G2

Reference Manual

Version 2015

G2 Reference Manual, Version 2015

November 2018

The information in this publication is subject to change without notice and does not represent a commitment by Gensym Corporation.

Although this software has been extensively tested, Gensym cannot guarantee error-free performance in all applications. Accordingly, use of the software is at the customer's sole risk.

Gensym Corporation
52 Second Avenue
Burlington, MA 01803 USA
Telephone: (781) 265-7100
Fax: (781) 265-7101

Part Number: DOC014-1200

Contents Summary

Contents

Preface

Describes this manual and the conventions that it uses.

About this Manual

This reference manual presents G2, a development environment for creating intelligent, real-time, knowledge-based applications.

Audience

This manual is written for G2 application developers and system integrators. It addresses the application developer or system integrator as *you*, and refers to a G2 end-user as *the user*.

This manual assumes that you have done one or more of the following:

- Taken one or more G2 courses provided by Gensym.
- Gone through the *Getting Started with G2 Tutorials*.
- Otherwise become somewhat familiar with G2.

Organization

This manual contains the following chapters and appendixes:

	Title	Description
Part I	**Introduction to G2**	
1	Overview of G2	Presents a summary of and orientation to G2's major features.
2	The Developer's Environment	Introduces features and strategies for developing a G2-based application.
Part II	**Global G2 Components**	
3	Knowledge Bases	Shows how to work with the current KB, save the current KB, and load a KB.
4	Workspaces	Shows how to use workspaces to organize your KB's items.
5	Modularized KBs	Describes how to partition your KB into modules.
6	System Tables	Describes the use of system tables to set global preferences.
7	Configurations	Describes how configurations override the default behavior of items.
8	G2-Windows	Describes how G2 associates g2-window items with visible windows.
Part III	**Knowledge Representation**	
9	Values and Types	Describes the role of values and types in a knowledge base.
10	G2 Items	Presents the characteristics that are common to all G2 items.
11	Attributes and Tables	Shows you how to use item attributes and the attribute tables that display them.
12	Attribute Access FacilityAttribute Access Facility	Presents the capabilities of the attribute access facility.

	Title	Description
13	Classes and Class Hierarchy	Describes the principles, structure, and use of the G2 class hierarchy.
14	Definitions	Describes class definitions and shows you how to use them.
15	Variables and Parameters	Describes variables and parameters and how to use them within a KB.
16	Lists and Arrays	Describes how to use lists and arrays.
17	Hash Tables and Priority Queues	Describes how to use hash tables and priority queues.
18	Connections	Describes connections, connection posts, and junction blocks.
19	Relations	Describes how to associate items in a non-graphical way.
Part IV	**Computational Capabilities**	
20	Actions	Describes each G2 action and shows you how to use it.
21	Expressions	Describes the purpose and syntax of each G2 expression.
22	Procedures	Shows how to define, customize, and use G2 procedures.
23	Methods	Shows how to define and use G2 methods.
24	Rules, Inferencing, and Chaining	Describes how G2 invokes rules to perform actions.
25	Formulas	Describes generic and specific formulas and their use.
26	Text Parsing and Manipulation	Describes capabilities for manipulating text and substrings, parsing and tokenizing text using regular expressions, and interconverting text between the Gensym and Unicode character sets.
27	XML Parsing	Describes how to parse XML code and make callbacks to user-defined procedures.

	Title	Description
43	Custom Windows Dialogs	Describes the API procedures for creating custom Windows dialogs.
44	Windows Views, Panes, and UI Features	Describes the API procedures for creating Windows views.
Part VII	**Editors and Facilities**	
45	The Text Editor	Describes how to create text items and how to use text inserters.
46	The Icon Editor and Icon Management	Describes the G2 Icon Editor and its icon-description language.
47	The Inspect Facility	Describes how to use the Inspect facility to search for items.
48	Natural Language Facilities	Describes the facilities for using non-English languages in a KB.
49	G2 Character Support	Presents a description of the G2 character support through Unicode.
Part VIII	**Debugging & Optimization**	
50	Error Handling	Describes the G2 error-handling capabilities.
51	Debugging and Tracing	Describes G2 facilities that can assist in debugging your KB.
52	Explanation Facilities	Describes the facilities that collect and display data about rules and formulas and the objects they reference.
53	Profiling and KB Performance	Describes techniques for evaluating and improving KB performance.
54	G2-Meters	Shows how to create, configure, and use G2-meters.
55	Memory Management	Describes G2's memory regions and shows how to manage them.
56	Task Scheduling	Describes the G2 scheduler, the G2 clock, and task queues.

	Title	Description
D	Syntax Conventions	Describes the notation and user-specified terms used in G2 syntax.
E	G2 KBs and GIF Files	Describes the demonstration, sample, and utility KBs, and the GIF files that ship with G2.
F	Superseded Practices	Describes G2 capabilities that are obsolete and may not be supported indefinitely.

Conventions

This guide uses the following typographic conventions and conventions for defining system procedures.

Typographic

Convention Examples	Description
g2-window, g2-window-1, ws-top-level, sys-mod	User-defined and system-defined G2 class names, instance names, workspace names, and module names
history-keeping-spec, temperature	User-defined and system-defined G2 attribute names
true, 1.234, ok, “Burlington, MA”	G2 attribute values and values specified or viewed through dialogs
Main Menu > Start KB Workspace > New Object create subworkspace Start Procedure	G2 menu choices and button labels
conclude that the x of y ...	Text of G2 procedures, methods, functions, formulas, and expressions
new-argument	User-specified values in syntax descriptions

Convention Examples	Description
text-string	Return values of G2 procedures and methods in syntax descriptions
File Name, OK, Apply, Cancel, General, Edit Scroll Area	GUIDE and native dialog fields, button labels, tabs, and titles
File > Save Properties	GMS and native menu choices
workspace	Glossary terms
`c:\Program Files\Gensym\`	Windows pathnames
`/usr/gensym/g2/kbs`	UNIX pathnames
`spreadsh.kb`	File names
`g2 -kb top.kb`	Operating system commands
`public void main()` `gsi_start`	Java, C and all other external code

Note Syntax conventions are fully described in the *G2 Reference Manual*.

Procedure Signatures

A procedure signature is a complete syntactic summary of a procedure or method. A procedure signature shows values supplied by the user in *italics*, and the value (if any) returned by the procedure *underlined*. Each value is followed by its type:

g2-clone-and-transfer-objects
(*list*: class item-list, *to-workspace*: class kb-workspace,
delta-x: integer, *delta-y*: integer)
-> *transferred-items*: g2-list

Related Documentation

G2 Core Technology

- *G2 Bundle Release Notes*
- *Getting Started with G2 Tutorials*
- *G2 Reference Manual*
- *G2 Language Reference Card*
- *G2 Developer's Guide*
- *G2 System Procedures Reference Manual*
- *G2 System Procedures Reference Card*
- *G2 Class Reference Manual*
- *Telewindows User's Guide*
- *G2 Gateway Bridge Developer's Guide*

G2 Utilities

- *G2 ProTools User's Guide*
- *G2 Foundation Resources User's Guide*
- *G2 Menu System User's Guide*
- *G2 XL Spreadsheet User's Guide*
- *G2 Dynamic Displays User's Guide*
- *G2 Developer's Interface User's Guide*
- *G2 OnLine Documentation Developer's Guide*
- *G2 OnLine Documentation User's Guide*
- *G2 GUIDE User's Guide*
- *G2 GUIDE/UIL Procedures Reference Manual*

G2 Developers' Utilities

- *Business Process Management System Users' Guide*
- *Business Rules Management System User's Guide*
- *G2 Reporting Engine User's Guide*
- *G2 Web User's Guide*
- *G2 Event and Data Processing User's Guide*

- *G2 Run-Time Library User's Guide*
- *G2 Event Manager User's Guide*
- *G2 Dialog Utility User's Guide*
- *G2 Data Source Manager User's Guide*
- *G2 Data Point Manager User's Guide*
- *G2 Engineering Unit Conversion User's Guide*
- *G2 Error Handling Foundation User's Guide*
- *G2 Relation Browser User's Guide*

Bridges and External Systems

- *G2 ActiveXLink User's Guide*
- *G2 CORBALink User's Guide*
- *G2 Database Bridge User's Guide*
- *G2-ODBC Bridge Release Notes*
- *G2-Oracle Bridge Release Notes*
- *G2-Sybase Bridge Release Notes*
- *G2 JMail Bridge User's Guide*
- *G2 Java Socket Manager User's Guide*
- *G2 JMSLink User's Guide*
- *G2 OPCLink User's Guide*
- *G2 PI Bridge User's Guide*
- *G2-SNMP Bridge User's Guide*
- *G2 CORBALink User's Guide*
- *G2 WebLink User's Guide*

G2 JavaLink

- *G2 JavaLink User's Guide*
- *G2 DownloadInterfaces User's Guide*
- *G2 Bean Builder User's Guide*

G2 Diagnostic Assistant

- *GDA User's Guide*
- *GDA Reference Manual*
- *GDA API Reference*

Customer Support Services

You can obtain help with this or any Gensym product from Gensym Customer Support. Help is available online, by telephone, by fax, and by email.

To obtain customer support online:

➔ Access G2 HelpLink at `www.gensym-support.com`.

You will be asked to log in to an existing account or create a new account if necessary. G2 HelpLink allows you to:

- Register your question with Customer Support by creating an Issue.
- Query, link to, and review existing issues.
- Share issues with other users in your group.
- Query for Bugs, Suggestions, and Resolutions.

To obtain customer support by telephone, fax, or email:

➔ Use the following numbers and addresses:

	Americas	Europe, Middle-East, Africa (EMEA)
Phone	(781) 265-7301	+31-71-5682622
Fax	(781) 265-7255	+31-71-5682621
Email	service@gensym.com	service-ema@gensym.com

Part I

Introduction to G2

Chapter 1: Overview of G2

Presents a summary of and orientation to G2's major features.

Chapter 2: The Developer's Environment

Introduces features and strategies for developing a G2-based application.

1

Overview of G2

Presents a summary of and orientation to G2's major features.

Introduction

G2 is a complete development environment for creating and deploying intelligent real-time applications. You can use G2 to develop applications that solve many problems commonly encountered in business, scientific, and industrial markets.

While G2 is flexible enough to use for almost any intelligent application, G2 users typically apply G2's capabilities to complex situations that require:

- Monitoring, diagnosis, and alarm handling.
- Scheduling and logistics.
- Supervisory and advanced control.
- Process design, simulation, and re-engineering.
- Intelligent network management.
- Decision support for enterprise-wide operations.

This overview does not attempt to provide detailed descriptions of G2's many features, nor does it offer technical insight into the underlying capabilities of G2. The remainder of this manual accomplishes those tasks. Instead, the overview provides an orientation to G2 that does two things:

- Presents a top-level view of the major features of the G2 development environment.
- Provides a reference for each main topic.

For a more extensive overview of G2 than this chapter provides, see:

- *G2 for Application Developers: An Introduction*. This document is available on request from Gensym. It provides a technical overview suitable for evaluating G2's applicability to particular needs.
- *G2 Developer's Guide*. This guide is included in the G2 Core Technology documentation kit and online. It provides guidelines and techniques for using G2 to develop knowledge-based applications.

Basic Components

G2 provides a complete, graphical development environment for modeling, designing, building, and deploying intelligent applications. To create a G2 application, you interact with a number of basic components.

Knowledge Bases

An application you develop in G2 is called a **knowledge base**, or **KB**. You create a new KB by adding items to the current G2 and then saving your work in one or more KB files. You can load an existing KB, then edit its contents or use it as needed.

A KB can be running, paused, or stopped. When a KB is running, reasoning and computation occur. When a KB is paused, transient data is maintained, but processing halts. While G2 is running, you can load a KB programmatically and

can save a KB either interactively or programmatically. G2 must be paused or reset to load a KB interactively.

Knowledge bases are described in Knowledge Bases.

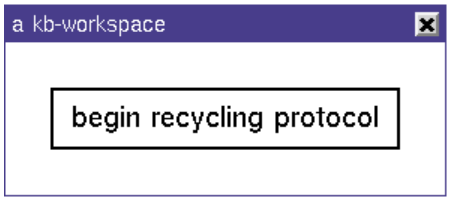
All components of a KB exist as **items** which have attributes. Items can appear graphically as **icons**. For information on items, see G2 Items.

Workspaces

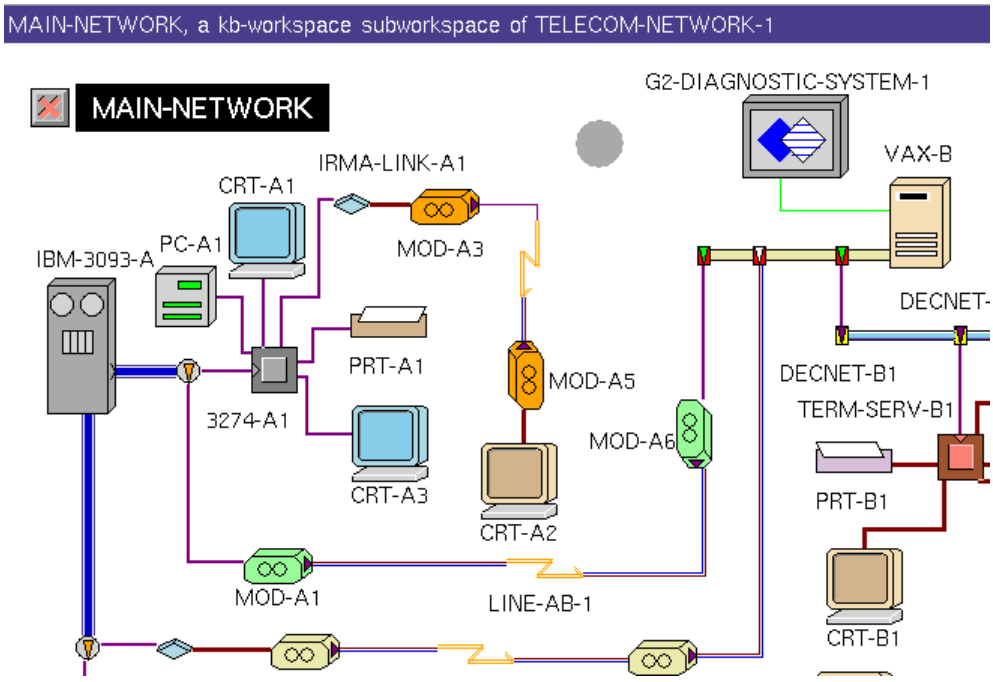
G2 calls the *blank pages* upon which you create, maintain, and organize items **workspaces**. A KB can contain one or many workspaces. The items upon workspaces are capable of having their own subsidiary workspaces. Thus, you can create a logical hierarchy of items and workspaces to group and organize your KB data.

Workspaces can contain anything from text messages to entire schematics that model real-time activity.

This workspace contains a single simple message:



This workspace displays part of a networking schematic from a telecommunications application:



To display and capture workspace knowledge, you can scale and print all workspaces. Among other things, you can:

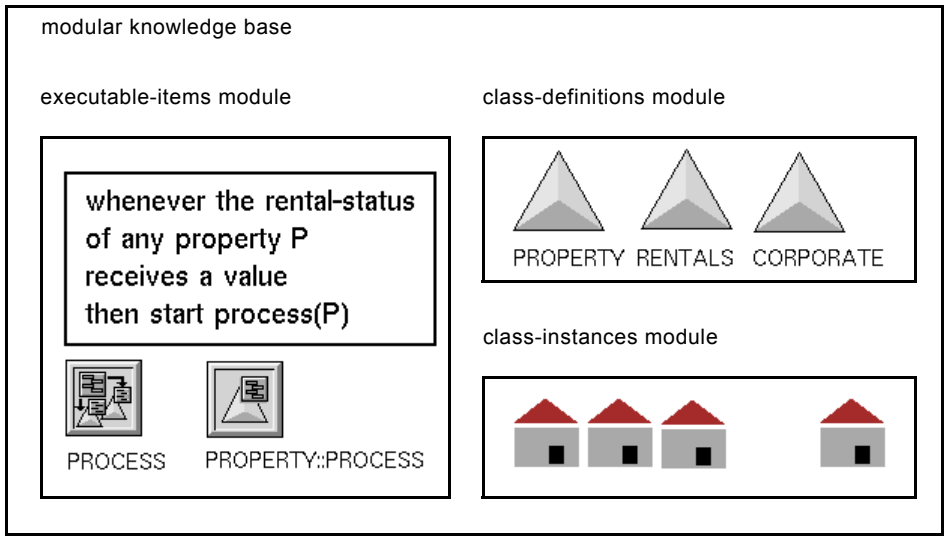
- Scale workspaces up to four times their full size, or down to a very small size, in effect iconizing the workspace.
- Hide workspaces.
- Print workspaces using PostScript files, or encapsulated PostScript files if the workspace is no larger than one physical sheet of paper.
- Print workspaces onto arbitrarily large paper sizes.

Workspaces are described in Workspaces.

Modules

You can develop a large KB from smaller, more manageable pieces called **modules**. Each module contains a set of related items that together comprise a KB.

You might begin to build an application by populating an empty KB, organizing the knowledge that pertains to certain classes of items into different modules. For example, you could define a module for class definitions, define another module for instances of the classes, and define a third module for executable items that manipulate class instances as represented in the following figure.



Modules facilitate modular development and reusability. When several developers are working on a single application, each can work on a separate module These can later be combined to form the entire application. Class definitions and other knowledge can be saved in a single module and used across multiple applications.

Using modules you can:

- Specify an alternate search path for locating module KB files.
- Locate items on a per module basis, using the Inspect facility.
- Analyze the module consistency of a KB, using the Inspect facility.
- Delete all related workspaces when deleting a module.
- Merge, delete, and save modules programmatically, using system procedures.

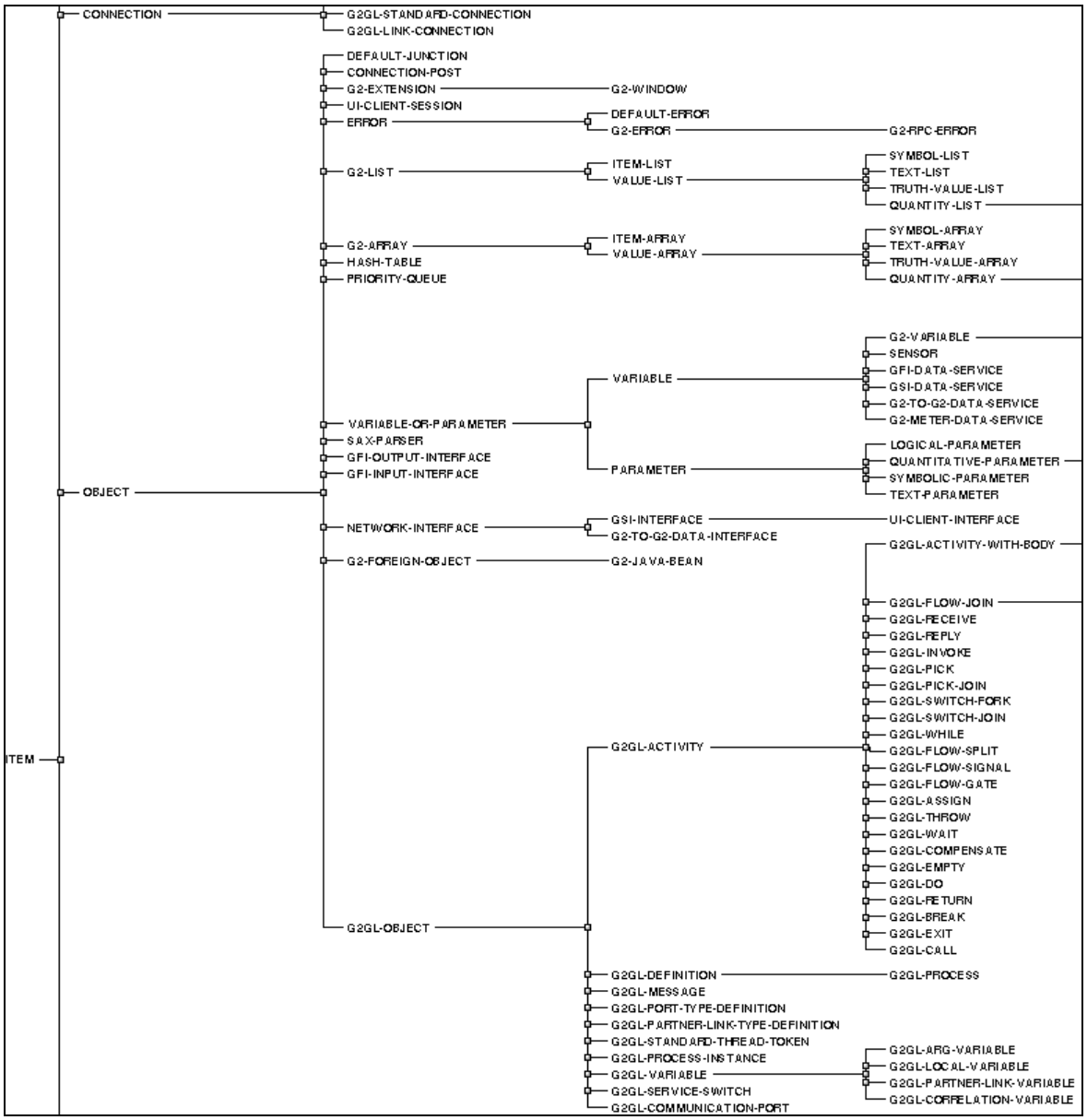
For a complete description of using modules, see Modularized KBs.

Classes and Class Hierarchy

G2 development is based on object-oriented design. Knowledge representation is maintained and extended through **classes** in the G2 class hierarchy.

G2 includes a large set of system-defined classes, many of which you can use as the foundation of customized, user-defined classes. You can add to the G2 class hierarchy through the use of its extensible classes. Every class within the G2 class hierarchy is either a system- or user-defined class.

The next figure shows a small portion of the G2 class hierarchy as displayed by the **Inspect** facility, G2's tool for accessing and browsing KB knowledge. The hierarchy begins with the item class on the left, which is the highest class level, and extends to the right. All classes shown are system-defined classes.


ITEM
CONNECTION
G2GL-STANDARD-CONNECTION
G2GL-LINK-CONNECTION
OBJECT
DEFAULT-JUNCTION
CONNECTION-POST
G2-EXTENSION
G2-WINDOW
UI-CLIENT-SESSION
ERROR
DEFAULT-ERROR
G2-ERROR
G2-RPC-ERROR
G2-LIST
ITEM-LIST
VALUE-LIST
SYMBOL-LIST
TEXT-LIST
TRUTH-VALUE-LIST
QUANTITY-LIST
G2-ARRAY
HASH-TABLE
PRIORITY-QUEUE
ITEM-ARRAY
VALUE-ARRAY
SYMBOL-ARRAY
TEXT-ARRAY
TRUTH-VALUE-ARRAY
QUANTITY-ARRAY
VARIABLE-OR-PARAMETER
SAX-PARSER
GFI-OUTPUT-INTERFACE
GFI-INPUT-INTERFACE
VARIABLE
PARAMETER
G2-VARIABLE
SENSOR
GFI-DATA-SERVICE
GSI-DATA-SERVICE
G2-TO-G2-DATA-SERVICE
G2-METER-DATA-SERVICE
LOGICAL-PARAMETER
QUANTITATIVE-PARAMETER
SYMBOLIC-PARAMETER
TEXT-PARAMETER
NETWORK-INTERFACE
GSI-INTERFACE
G2-TO-G2-DATA-INTERFACE
UI-CLIENT-INTERFACE
G2-FOREIGN-OBJECT
G2-JAVA-BEAN
G2GL-OBJECT
G2GL-ACTIVITY
G2GL-ACTIVITY-WITH-BODY
G2GL-FLOW-JOIN
G2GL-RECEIVE
G2GL-REPLY
G2GL-INVOKE
G2GL-PICK
G2GL-PICK-JOIN
G2GL-SWITCH-FORK
G2GL-SWITCH-JOIN
G2GL-WHILE
G2GL-FLOW-SPLIT
G2GL-FLOW-SIGNAL
G2GL-FLOW-GATE
G2GL-ASSIGN
G2GL-THROW
G2GL-WAIT
G2GL-COMPENSATE
G2GL-EMPTY
G2GL-DO
G2GL-RETURN
G2GL-BREAK
G2GL-EXIT
G2GL-CALL
G2GL-DEFINITION
G2GL-PROCESS
G2GL-MESSAGE
G2GL-PORT-TYPE-DEFINITION
G2GL-PARTNER-LINK-TYPE-DEFINITION
G2GL-STANDARD-THREAD-TOKEN
G2GL-PROCESS-INSTANCE
G2GL-VARIABLE
G2GL-SERVICE-SWITCH
G2GL-COMMUNICATION-PORT
G2GL-ARG-VARIABLE
G2GL-LOCAL-VARIABLE
G2GL-PARTNER-LINK-VARIABLE
G2GL-CORRELATION-VARIABLE

Classes have **attributes**, which define the inherited and locally defined properties of the class. G2 maintains class attributes within **attribute tables**. Here is the iconic representation and the attribute table of a G2 integer variable class:

an integer-variable

Options	do not forward chain, breadth first backward chain
Notes	OK
Item configuration	none
Names	none
Tracing and breakpoints	default
Data type	integer
Initial value	none
Last recorded value	no value
History keeping spec	do not keep history
Validity interval	supplied
Formula	none
Simulation details	no simulation formula yet
Initial value for simulation	default
Data server	inference engine
Default update interval	none

Classes may have associated **methods**. These define the operations characteristic of each class. Methods allow generic operations to be implemented in class-specific ways (**polymorphism**). Code that invokes a method needs only to know the method's name: the details of how to perform the operation exist in the method, not in the code that invokes it (**encapsulation**).

G2 permits **multiple inheritance** in its class hierarchy: any user-defined class can inherit the attributes and methods of any number of superior classes. To facilitate modular design, classes can inherit attributes with identical names that are defined by different superior classes.

The G2 class hierarchy is presented in Classes and Class Hierarchy. The ability to extend the class hierarchy to create custom classes is described in Definitions.

Attributes and their tables are covered in Attributes and Tables. Programmatic access to G2 system-defined attributes and their data structures is described in Attribute Access Facility.

Methods are described in Methods.

Knowledge Representation

Items are the fundamental data structures within G2 that you use to represent knowledge. You use items to collect and organize knowledge about real objects, processes, and relationships. Items are described in G2 Items.

G2 represents knowledge within items as values, which are data structures that are generated as the result of expression evaluations and are associated with item attributes. G2 supports a variety of value types including integers, floats, text values, truth values, symbols, and composite types. These are described in Values and Types.

G2 supports a variety of other types of knowledge representation:

- Variables and parameters, which keep histories of values, described in Variables and Parameters.
- Lists and arrays, which consist of a series of elements of a particular type, described in Lists and Arrays.
- Hash tables, which consist of a collection of key-value pairs, and priority queues, which consist of a collection of items, each with a priority, described in Hash Tables and Priority Queues.
- Connections, which are graphical items that create a logical relationship between two or more, objects, described in Connections.
- Relations, which are non-graphical items that create a logical relationship between two or more, objects, described in Relations.

Configurations

G2 provides a unique capability, called configurations, for creating KB user modes and controlling the behavior of KB items. You can use configurations to define the behavior of single items, or hierarchically to specify the behavior of groups and classes of items that you designate in various ways.

Typical uses of configurations include:

- Adding capabilities and restrictions of many different kinds to any item.
- Defining how items respond to particular user actions, such as mouse clicks.
- Allowing and prohibiting network access to an entire KB, or to any of its individual components.

Configurations are explained in Configurations.

System Tables

G2 provides system tables that define global parameters applicable to an entire KB, including parameters related to KB configuration, modules, menus, editor, fonts, color, drawing, printer setup, saving, G2 server, data server, inference engine, language, logbook, message board, log file, simulation, G2 graphical language, timing, and debugging.

For details, see System Tables.

G2 Windows

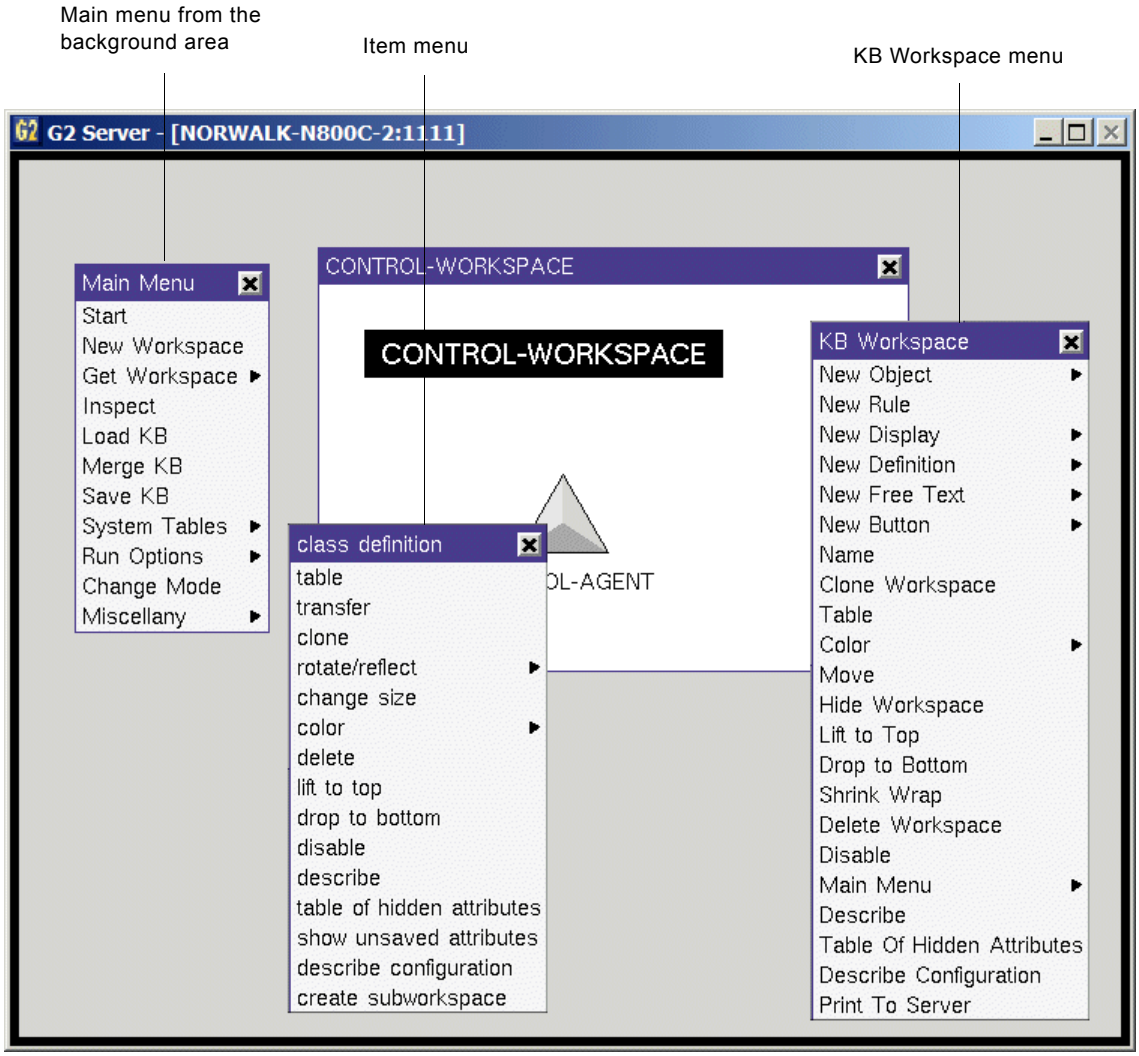
A G2 window represents knowledge about the window within which you interact with G2. G2 can automatically associate a G2 window item either with the local G2 window or with a remote G2 window, which is the window displayed by a Telewindows connected to your G2. A G2 window provides a variety of information, including connected users, language, user mode, and so on.

For details, see G2-Windows.

G2 Developer's Environment

G2 provides a rich development environment for building application, which includes:

- Menus, which provide access to G2's interactive capabilities. You can access G2 menus by clicking the G2 window background, any workspace, an individual item. Here are some examples of menus:



- G2 Message Board, which displays user-generated messages.
- G2 Operator Logbook, which displays system-generated messages and errors.

For more information, see The Developer's Environment.

Computational Capabilities

At the core of the developer's environment lies G2's structured natural language. G2 uses this language in all programmatic statements. Since the G2 language is similar to ordinary human language, it is easier to read statements that are written in the G2 language than it is to read other programming languages.

For example, the following rule scans all refrigerators as G2 executes, tests each one for a specified temperature condition, and performs an action on any refrigerator for which the condition is true:

```
if the temperature of any refrigerator R > 40 degrees
    then start adjust-temperature-procedure(R)
```

For a summary of the G2 programming language, see the *G2 Language Reference Card*.

Procedures, Methods, and Rules

Programmatic control over a KB and its corresponding real-time external events occurs within:

- **Procedures**, covered in Procedures.
- **Methods**, described in Methods.
- **Rules**, presented in Rules, Inferencing, and Chaining.

Each of these items contains G2 statements. **Statements** consist of **expressions**; expressions can include **actions**.

Expressions

You can use G2 expressions to:

- Obtain information about items.
- Specify actions to be executed on items.

Expressions are described in Expressions.

Actions

You can use G2 actions to perform many different tasks, including:

- Creating, moving, deleting, and showing items.
- Controlling the position of any workspace in the current G2 window.
- Accessing the position of items upon a workspace.
- Obtaining the current size of any item.

Actions describes all actions.

Formulas

G2 provides formulas for creating equations that provide values for a variable or parameter. G2 computes a formula only when a value is needed. Formulas are described in Formulas.

Text and XML Parsing

G2 provides a variety of functions and expressions for manipulating and parsing text strings, described in Text Parsing and Manipulation.

G2 also provides a facility for parsing XML code and executing user-defined callbacks, using the SAX (Simple API for XML) standard, described in XML Parsing.

Functions

G2 provides the ability to define user-defined functions, which are named operations that return a value, with or without an argument. Functions are similar to procedures except they are invoked differently. G2 also defines a set of system-defined functions for a variety of operations including arithmetic, character manipulation, time operations, and more.

Functions are described in Functions.

System Procedures

System procedures are a group of G2-provided procedures, contained in the `sys-mod.kb` file. G2 includes hundreds of system procedures.

You use system procedures by merging or requiring the `sys-mod.kb` file into your current KB, and then calling system procedures as needed from user-defined code.

System procedures let you complete a variety of different tasks, and, in some cases, provide a programmatic access to items that is unavailable through expressions or actions.

Using system procedures, you can:

- Obtain and set various graphical properties of any KB item.
- Manipulate item and workspace layering.
- Obtain an item's system predicate status (permanent, transient, or showing).
- Determine the position of any item.

- Register items for item passing.
- Perform profiling operations.
- Sort lists and arrays directly or through keys.
- Determine memory usage.

G2 system procedures are described in the *G2 System Procedures Reference Manual* and the *G2 System Procedures Reference Card*.

G2 Graphical Language

The G2 Graphical Language (G2GL) allows the execution of processes, including business, industrial, and general reasoning processes, directly within G2. It provides a self-contained graphical programming environment for the specification of any type of process, which fully integrates with G2.

The process activities are generally based on the Business Process Execution Language for Web Services (BPEL4WS or BPEL for short) language. BPEL is an industry initiative, now managed by OASIS, to establish an effective standard framework for describing and defining high-level business processes that are offered as Web services.

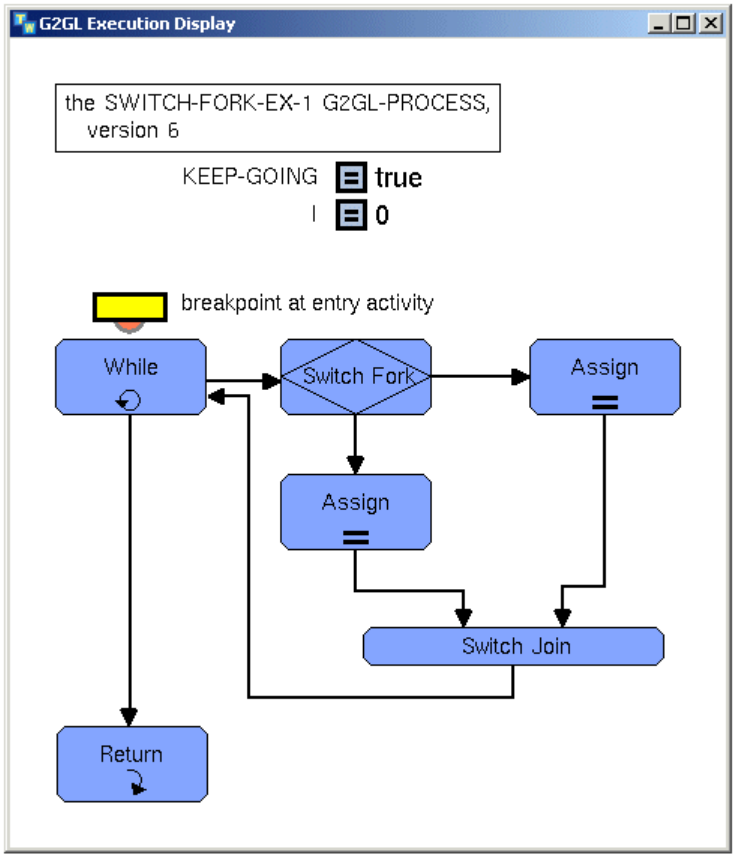
G2GL provides a tightly integrated environment for developing business processes that includes process modeling, compilation, execution, debugging, and animation. G2GL supports importing and exporting processes, based on the BPEL4WS XML specification.

G2GL provides a variety of process activities for expressing the logic of the process, including activities that perform sequential processing, branching and concurrency, synchronization, and looping. G2GL supports most BPEL activities, as well as activities beyond those within the BPEL specification including waiting, debugging, and breakpoints. A G2GL process can include local variables for holding data. G2GL activities use G2GL expressions, which are similar to G2 expressions.

G2GL provides communication between two linked partner processes. A partner is a series of connected elements that mediate communication between two linked partners.

G2GL allows for scope activities, which have bodies that specify subprocesses. You can also have scope-like fault, alarm event, or message event handlers.

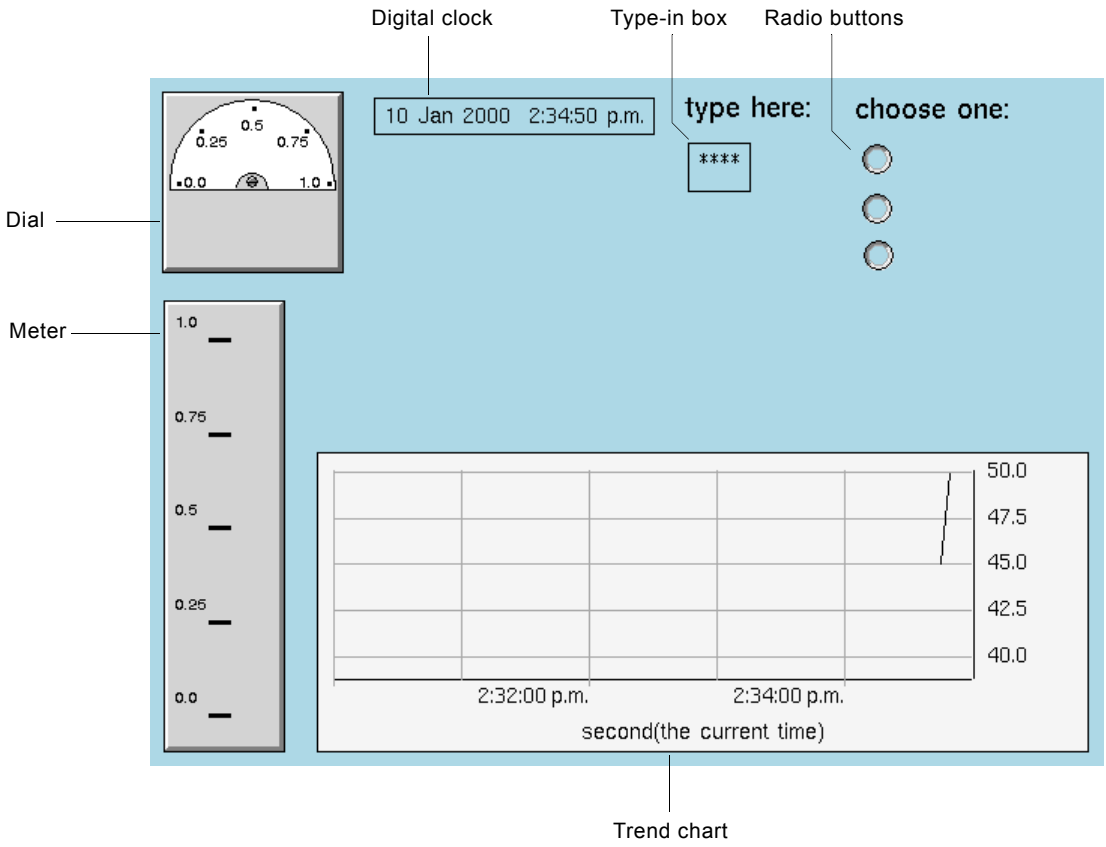
For example, this figure shows a G2GL process execution with a breakpoint:



For information using G2GL, see G2 Graphical Language (G2GL).

Extensible and Graphical Components

The G2 environment is both graphical and extensible. Almost everything in G2 has a graphical representation. You can use system-defined display items to show the state of your application as it changes over time, and system-defined buttons to send commands to G2 or the outside world. You can extend G2's graphics in various ways to provide a customized visual environment.



The preceding figure shows some system-defined and user-defined display items. G2 provides and allows you to customize many such items, including:

- Radio buttons, type-in boxes, check-boxes
- User menu choices
- Readout displays and digital clocks
- Meters and dials
- Charts, graphs, and trend charts

You can use system procedures to get information about, and then change, many graphical aspects of items, as described in System Procedures.

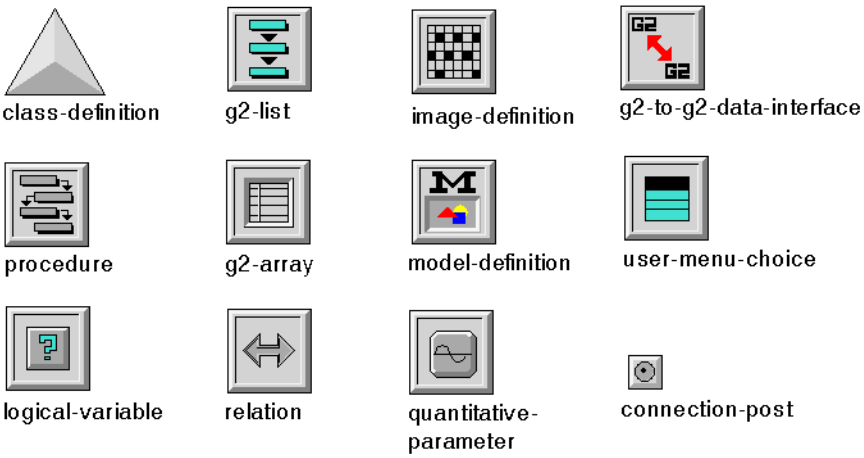
For details on these graphical components, see:

- Buttons.
- User Menu Choices
- Readout Tables, Dials, and Meters.
- Freeform Tables.

- Charts.
- Graphs.
- Trend Charts.

Icons

All G2 items are represented graphically or textually. The iconic representation of items supports a full range of colors. G2's **Paint drawing mode** permits polychrome icons to overlap and maintain their color. G2 includes a large palette of system-defined colors that you can apply to various KB items, including workspaces, icons, and textual items. Some examples of system-defined icons are:



For information on creating and modifying icons, see The Icon Editor and Icon Management.

Images

G2 supports the use of JPEG, X Bit Map (XBM) and Graphics Interchange Format (GIF) images within a KB. The G2 Icon Editor, which you can use to create new icons or edit existing ones, allows you to use images as icon components, where they appear in monochrome.

You can also use images to provide full-color workspace backgrounds. The following workspace has a frame style defined by a frame-style-definition and a color background image:

For information on importing and using externally defined images, see External Images.

Textual Items

Textual items, such as messages and free text, use an outline font technology, making fonts more readable at smaller scales, and providing enhanced typographical detail for larger font sizes.

For information on creating textual items, see Text Items and Messages.

Custom User Interfaces

G2 provides extensive tools for building custom Windows user interfaces for display in Telewindows, including:

- Custom menus, including menu bars, popup menus, localization, dynamic menus, and callbacks.

  For details, see Windows Menus.

- Basic Windows dialogs, including basic, query, notification, file, and print dialogs.

  For details, see Windows Dialogs.

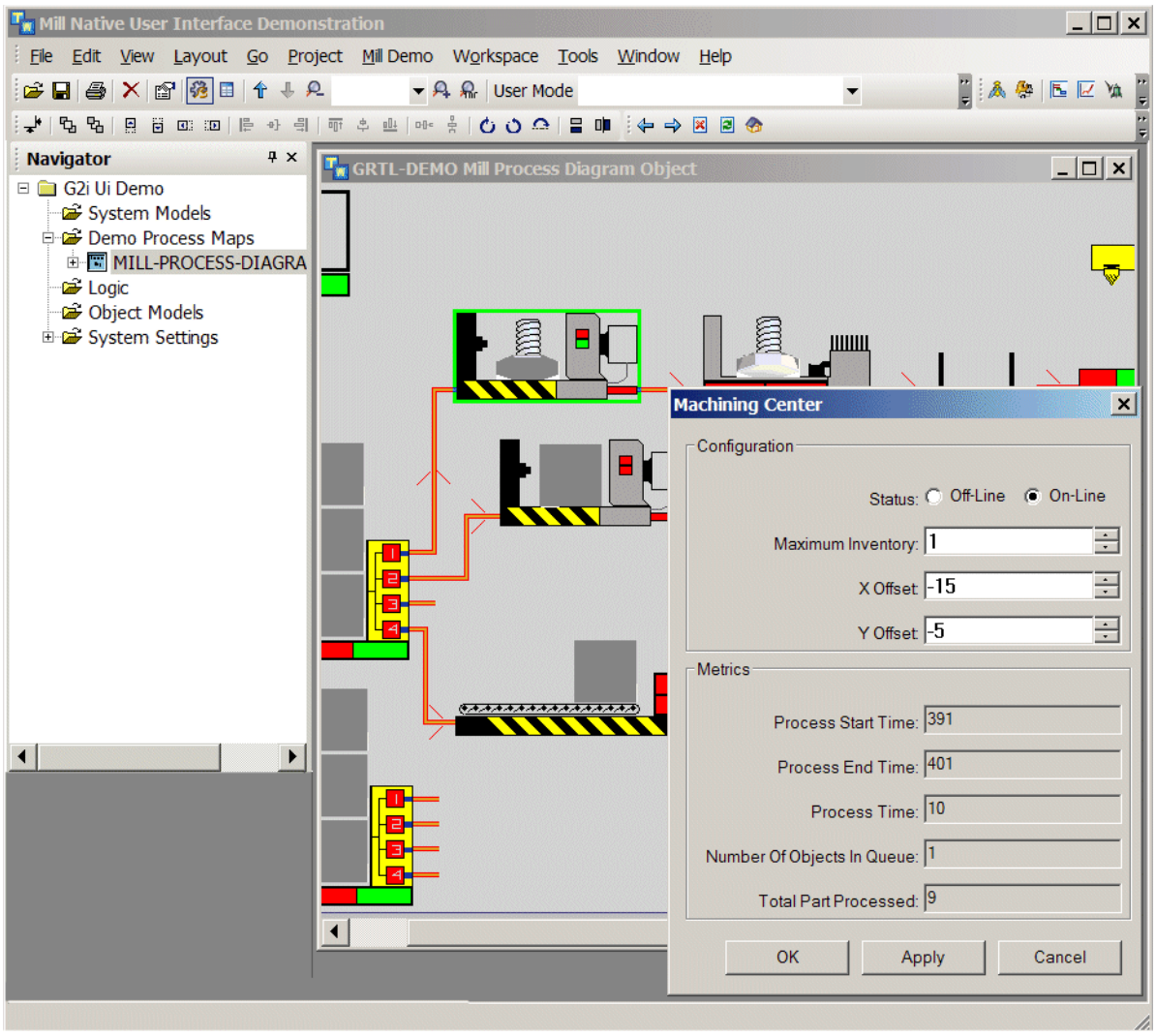
- Custom Windows dialogs, including posting, modifying, and querying custom dialogs, callbacks, with numerous standard windows controls such as text, buttons, lists, color, time and date, progress bars, tabular views, grouping, images, and workspaces.

  For details, see Custom Windows Dialogs.

- Windows views, including chart views, HTML views, shortcut bars, and tree views.

  For details, see Windows Views, Panes, and UI Features.

Here is an example of a custom user interface that shows some of these features:



Editors and Facilities

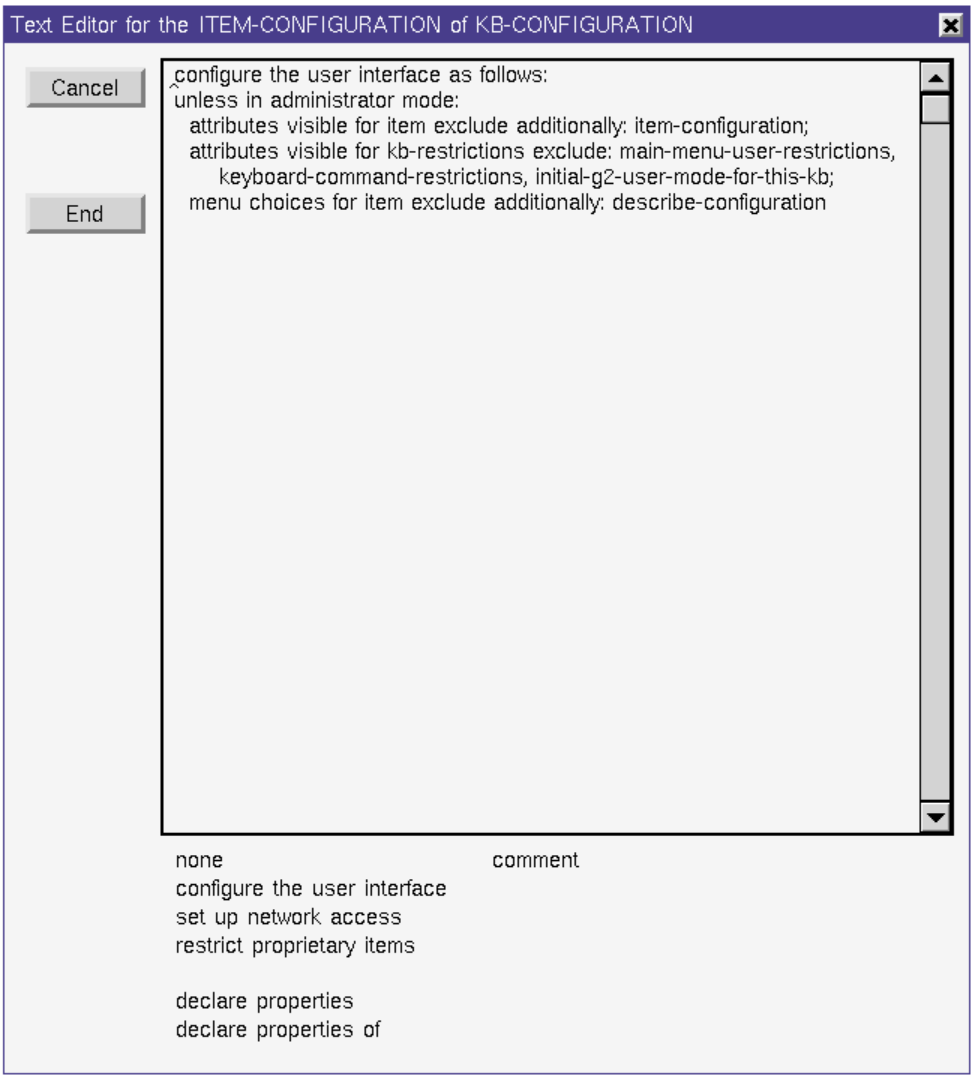
G2 provides various editors and facilities for interacting with text, icons, the overall KB, languages, and characters.

Text Editor

G2 includes two text editors, the standard Text Editor for entering and editing limited amounts of texts, such as item names and short statements, and the scrollable Text Editor. Both editors are described in The Text Editor.

For information about the Windows text editor available through Telewindows, see the *Telewindows User's Guide*.

The scrollable Text Editor is useful with multi-line text entries, such as procedures, methods, and complex configurations. For example:



Icon Editor

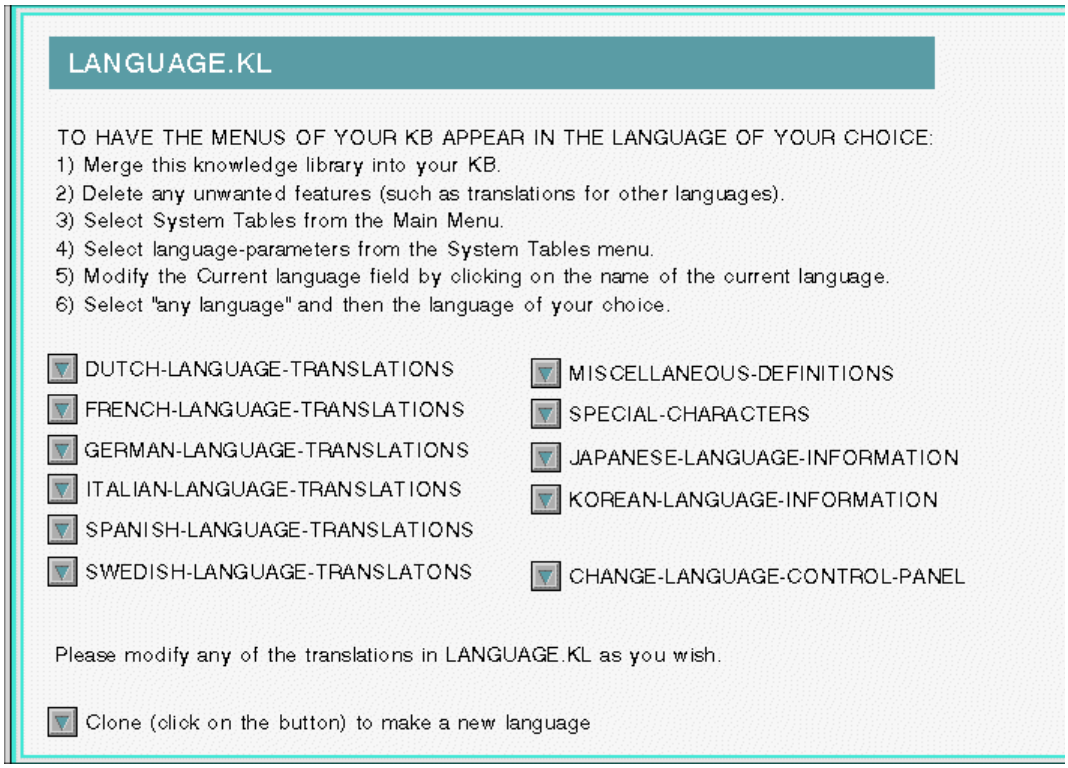
Icon Editor allows you to define a class icon with graphic tools. The Icon Editor converts the resulting graphical description into G2 code, and sets this code as the value of the icon-description attribute of the class definition. The Icon Editor is described in The Icon Editor and Icon Management.

Inspect Facility

The Inspect facility allows you to search a knowledge base (KB) for items based on their type, class, attributes, and location. The Inspect Facility is described in The Inspect Facility.

Natural Language Facilities

G2's natural language facilities let you create your own menu translations for non-European languages. Additionally, G2 includes a `language.kl` KB with several complete sets of European language menu translations, as shown here:



Every G2 license includes the:

- European language facilities.
- Japanese language facilities.
- Korean language facilities.
- Chinese language facilities.
- Russian language facilities.

Japanese, Korean, and Chinese outline fonts require additional authorization. For details, see Natural Language Facilities.

G2 Character Support

G2 character representation is provided by the Unicode Worldwide Character Standard, which supports the storage, exchange, processing, and display of text for most of the world's modern and classical written languages. Supported characters cover the principal languages of the Americas, Europe, Middle East, Africa, India, Asia, and Pacifica. G2 character support is described in G2 Character Support.

Development and Deployment

G2 provides an incremental development and deployment environment. As development progresses, you can add capabilities to your KB at virtually any stage of the development cycle. Techniques and guidelines for G2 application development and deployment appear in the *G2 Developer's Guide*.

For a detailed overview of the G2 development environment, see The Developer's Environment.

Compilation

Compilation occurs each time you select the End button or type Ctrl-Enter when editing a procedure, rule, function, or any attribute containing a compatible expression in the text editor.

You can use configuration statements to declare certain items as stable-for-dependent-compilations. Declaring items this way informs G2 that certain parts of the item's knowledge will not change, letting G2 compile dependent items more efficiently. In large KBs, the more items you can declare as stable, the more performance will improve.

Error Handling and Debugging

G2 supports various error handling capabilities, including a system-defined class for errors, error handling statements within procedures for catching, signalling, and handling errors. For details, see Error Handling.

G2 provides various debugging capabilities, including displaying error and warning messages, displaying source-code error location, displaying trace messages, setting breakpoints and dynamic breakpoints, stepping through procedure code, displaying disassembled code for procedures, methods, and rules, and writing G2-state information and logbook messages to a file. For details, see Debugging and Tracing.

Explanation Facilities

G2's explanation facilities allow you to display including forward and backward chaining for a variable, invocations of backward-chaining rules for a variable, invocations of rules for an object that contain a generic reference to that object, invocations of a particular rule. For details, see Explanation Facilities.

Profiling a KB

As KB development nears completion, you can use the KB profiling facility to collect and analyze data about its performance during execution. After you identify which parts of your KB can benefit from further optimization, you can apply compilation configurations to help G2 to compile those parts more efficiently.

A complete description of G2 compilation and profiling appears in Profiling and KB Performance.

G2 Meters and Memory Management

G2 meters are specialized quantitative variables that monitor G2 and compute statistics about its performance, such as how much memory it is using, and how fast it is processing. For details, see G2-Meters.

G2 provides various tools for managing and allocating memory. For details, see Memory Management.

Task Scheduling

G2 supports subsecond time. You can specify a subsecond time interval that affects the G2 clock and thus the scheduler, certain intervals, and history collection specifications. To allow subsecond timing, G2 represents time as a float, rather than an integer.

The G2 scheduler directs task processing in G2. While a user never interacts with it directly, the scheduler controls all of the activity that the user sees, as well as many of G2's background activities. The scheduler is the G2 time keeper and task master; it is responsible for scheduling and prioritizing all tasks, executing tasks between clock ticks, and ticking the G2 clock.

For information about scheduling and time, see Task Scheduling.

Package Preparation

When deploying an application, you use G2's package preparation tools to remove source code and make a KB proprietary. You do this by marking items for text stripping, removing change logging and version information, and configuring proprietary workspaces. For details, see Package Preparation.

Licensing and Authorization

G2 provides licenses for offline and online use, and for development and deployment environments. It provides separate licensing for the Telewindows client, using dedicated or floating licenses.

You can configure G2 to be secure, which requires users to login with a password. You can also limit network access to a KB.

For details, see Licensing and Authorization and Network Security.

Networking and Interfacing

G2 offers these network and interfacing capabilities:

- Network security
- Telewindows
- G2-to-G2 interface
- G2 Gateway (GSI)
- Item passing
- Publish/subscribe
- Java interface
- Foreign functions support
- G2 as data service

Network Security

You can secure a KB from unauthorized network access through the use of special network-oriented configuration statements.

Using configurations, you can apply network security at any level you need to permit or disallow KB access across a network connection. Network security is described in Configurations.

Telewindows

Telewindows allows more than one user to access the same G2 independently. Each Telewindows user can open a **telewindow**, or remote view, into a running G2 process. Telewindows provides a client-server based capability in which a single G2 process, acting as a server, executes a KB, to which any number of authorized Telewindows clients users can connect. On Windows platforms, Telewindows provides a standard, Windows-based developer and end user interface for G2 applications.

For information about Telewindows, see Telewindows Support, and the *Telewindows User's Guide*.

G2-to-G2 Interface

The G2-to-G2 interface lets two or more G2 processes connect for the purpose of exchanging data. G2 supports the TCP/IP protocol only.

Once two systems are connected, you can:

- Use a remote G2 as the data server of one or more variables.
- Exchange various types of data, including any value.
- Pass entire items and their user- or system-defined attributes.

G2 also permits the dragging of single items between two G2 or Telewindows processes on Windows platforms.

For information about using the G2-to-G2 interface, see G2-to-G2 Interface.

G2 Gateway

The G2 Gateway standard interface (GSI) is a network-oriented toolkit used for developing software interfaces, or **bridges**, between G2 and other, external systems. G2 Gateway allows KBs to exchange various types of data between a G2 process and the bridge.

The G2 Gateway bridge is itself a process that communicates with G2 over the TCP/IP protocol, using a gsi-interface item.

For information about using G2 Gateway, see G2 Gateway, and the *G2 Gateway Bridge Developer's Guide*.

Item Passing

Item passing is supported across the G2-to-G2 and the G2 Gateway interfaces, through the use of remote procedure calls. G2 supports item passing by allowing you to:

- Pass any KB item by reference, using a network handle.
- Pass entire items, including complex items that contain attributes given by objects such as variables and parameters, or attributes that consist of lists or arrays of values or items.

Several system procedures support item passing. For details on item passing and the procedures that support it, see G2-to-G2 Interface. Information about item passing is also available in the *G2 Gateway Bridge Developer's Guide*.

Publish/Subscribe

G2 provides a publish/subscribe facility, which allows application developers to implement scalable, distributed applications that can respond dynamically to changes in the application, including changes in item attribute values, item deletion or creation, and custom events.

For information, see Publish/Subscribe Facility

Java Interface

G2 JavaLink provides a set of Java components and classes that you can use to communicate with Java/RMI applications.

For information on the Java interface classes and references for more information, see Interfacing with Java Applications.

Foreign Functions Support

G2 supports the use of **foreign functions**, which are functions written in C or C++ that you can call from within your KB as if they were local functions. The foreign function interface is platform-independent. You can start a foreign function either as an external process, or as a spawned process from within a KB.

To use foreign functions, you collect existing C source files into an executable **foreign image** to which G2 connects. Gensym provides sample files to help you create and use a foreign image.

Foreign functions and images are described in Foreign Functions.

G2 as Data Service

GService allows you to install and manage G2 and G2 bridges as services under Windows. You may use this utility to install any number of G2 or bridge processes as services as long as you provide a unique service name for each installed service. GService runs each service as a separate process.

For information on running GService as well as examples, see Windows Services.

Additional Capabilities and Information

G2 provides the following additional capabilities and information in appendices:

- Command-line options

  G2 provides a variety of command-line options for use when launching the G2 server, which are described in the Appendix A, Launching a G2 Process.

- G2 reserved words

G2 reserved words are symbols that cannot serve as a user-defined name in G2. For a complete list of reserved words, see Appendix B, Reserved Symbols.

- Mouse gestures, key bindings, and shortcut keys

  G2 supports standard mouse gestures for selection, where "standard" implies the Windows standard. They also support a number of other mouse gestures, key bindings, and shortcut keys for interacting with selection, workspaces, and items. G2 uses a selection style user interface where commands apply to the current selection. For the complete list, see Appendix C, Mouse Gestures, Key Bindings, and Shortcut Keys.

- Syntax conventions

  For a description of the notation and user-specified terms used to describe the G2 language, see Appendix D, Syntax Conventions.

- G2 KBs

  For a list of the demonstration, sample, tutorial, utility, and graphics files that are included with G2, see Appendix E, G2 KBs and GIF Files.

- Superseded practices

  For a description of the G2 features that have been superseded, see Appendix F, Superseded Practices.

G2 Utilities

G2 provides a number of utilities for developers to achieve uniformity, compatibility, and reliability in their applications. The G2 utilities are:

- **G2 ProTools (ProTools) —** Provides advanced G2 developer tools for speeding up development, testing, debugging, documenting, and deployment. See the *G2 ProTools User's Guide*.
- **G2 Foundation Resources (GFR) —** Establishes standard approaches to several important design and implementation issues commonly encountered in building inter-operable modules. GFR helps to assure the compatibility of modules with modules written by other authors who also use GFR. See the *G2 Menu System User's Guide*.
- **G2 User Interface Development Environment/User Interface Library (GUIDE/UIL) —** Provides a library of user interface components from which you can build dialogs from pre-built components. GUIDE includes a basic button library for navigation buttons. Once you have built a GUIDE application, you can remove the development modules of GUIDE from the application. See the *G2 GUIDE User's Guide* and *G2 GUIDE/UIL Procedures Reference Manual*.

- **G2 Menu System (GMS) —** Provides a way of implementing menu bars. All applications that need menu bars and popup menus in G2 should use GMS. See the *G2 Menu System User's Guide*.
- **G2 Dynamic Displays (GDD) —** Provides a number of attractive dials, meters, and displays, based on G2 power icons, which you can use directly or as direct superior classes. Use GDD to enhance the visual appeal of your application. See the *G2 Dynamic Displays User's Guide*.
- **G2 Developer's Interface (GDI) —** Provides menu templates and dialogs for standard menu layout and menu- based activities. You can use GDI as the basis for developing your own custom menu layout, or simply use one of the many useful GDI dialogs for selecting files, printing, manipulating modules, and the like. GDI is based on GMS and GUIDE. See the *G2 Developer's Interface User's Guide*.
- **G2 XL Spreadsheet (GXL) —** Provides a way of creating scrolling tabular displays for viewing and editing a wide variety of lists, arrays, and complex data structures. See the *G2 XL Spreadsheet Reference Manual*.
- **G2 OnLine Documentation (GOLD) —** Provides a set of related modules that implement online documentation based on external browsers and HTML. GOLD is the standard way to deliver context-sensitive help and to access documentation via keyword, index, and table-of-contents searches. See the *G2 OnLine Documentation User's Guide* and *G2 OnLine Documentation Developer's Guide*.

G2 Developer's Utilities

G2 provides the following developer's utilities, which provide a consistent development framework for building G2 decision management applications:

- **Business Process Management System (BPMS) —** Provides a user interface, classes, methods, and built-in services that are based on the G2 Graphical Language (G2GL). See the *Business Process Management System Users' Guide*.
- **G2 Business Rules and Management System** (BRMS) **—** Provides a mechanism for easily editing, organizing, analyzing, and executing complex business rules. See the *Business Rules Management System User's Guide*.
- **G2 Web Services** (GWEB) defines out-of-the-box Web pages and SOAP services, as well as classes and APIs enabling G2 to implement an HTTP server and serve HTML pages, XML structures, SOAP services, and files. See the *G2 Web User's Guide*.
- **G2 Reporting and Processing Engine** (GRPE) provides a consistent approach for defining reports and charts, collecting values, displaying tabular values in reports, and charting those values. See the *G2 Reporting Engine User's Guide*.

- **G2 Event and Data Processing** (GEDP) is a multi-purpose graphical language composed of graphical blocks that can be connected together to express a flow of data, perform calculations, execute functions, generate messages, and events. See the *G2 Event and Data Processing User's Guide*.
- **G2 Event Manager** (GEVM) provides tools that support highly scalable, distributed operator-advisory applications by providing an event "black board" and alarm management capabilities, as well as associated message queues, message browsers, and logging. See the *G2 Event Manager User's Guide*.
- **G2 Run-Time Library** (GRTL) provides a wide variety of development tools for the runtime environment. These include support for object models, which includes object keys, event notification, and support for localization, configuration files, command-line options, publish/subscribe, XML, and a variety of general runtime utilities. See the *G2 Run-Time Library User's Guide*.
- **G2 Dialog Utility** (GDU) extends the custom Windows dialog functionality that G2 provides to enable the rapid building and deployment of native Windows dialogs. This module also includes the **G2 Dialog Conversion Utility** (GDUC), which generates custom Windows dialog specifications from GUIDE/UIL dialogs and the **G2 Dialog Configuration Editor** (GDUE), which provides a native Windows editor for a native Windows dialog specification. See the *G2 Dialog Utility User's Guide*.
- **G2 Data Source Manager** (GDSM) provides tools for managing network connections and for pooling connections to improve throughput in large-scale applications, including UIL and native configuration dialogs. See the *G2 Data Source Manager User's Guide*.
- **G2 Data Point Manager** (GDPM) provides functionality to configure, log, replay, and simulate datapoints, typically related to external sensors such as temperature, pressure, and flow. These external values are represented in GDPM as external datapoints and obtain their values typically via an OPC or PI interface and bridge. *G2 Data Point Manager User's Guide*.
- **G2 Engineering Unit Conversion** (GEUC) provides a way of specifying the engineering units for entering and displaying values, as well as a large number of synonyms for those conversions in both the English and metric systems. *G2 Engineering Unit Conversion User's Guide*.
- **G2 Error Handling Foundation** (GERR) provides tools for error handling as an extension to G2 error and G2 Foundation Resources (GFR). See the *G2 Error Handling Foundation User's Guide*.
- **G2 Relation Browser** (GRLB) provides tools for displaying related items in a graphical layout. See the *G2 Relation Browser User's Guide*.

G2 Bridges

G2 provides the following bridges for communication with external systems and standards:

- Databases:
  - **G2-Oracle Bridge —** Provides communication with Oracle. See the *G2-Oracle Bridge Release Notes*.
  - **G2-Sybase Bridge —** Provides communication with Sybase. See the *G2-Sybase Bridge Release Notes*.
  - **G2-ODBC Bridge —** Provides communication with any relational database on any platform for which there is an ODBC driver. See the *G2-ODBC Bridge Release Notes*.

  For general information, see the *G2 Database Bridge User's Guide*.
- Devices and data historians:
  - **G2-PI Bridge —** Provides communication with the PI data historian. See the *G2-PI Bridge User's Guide*.
  - **G2-OPC Client Bridge** (OLE for Process Control) **—** Provides communication with data supplied by any OPC-compliant server. See the *G2 OPCLink User's Guide*.
- Distributed object standards and protocols:
  - **G2 ActiveXLink —** Provides communication with Microsoft ActiveX/COM. See the *G2 ActiveXLink User's Guide*.
  - **G2 JavaLink —** Provides communication with Java/RMI. See the *G2 JavaLink User's Guide*, *G2 DownloadInterfaces User's Guide*, and *G2 Bean Builder User's Guide*.
  - **G2 JMail Bridge —** Provides communication with JavaMail (JMail). See the *G2 JMail Bridge User's Guide.*
  - **G2 JMSLink —** Provides communication with Java Message Service (JMS). See the *G2 JMSLink User's Guide.*
  - **G2-SNMP Bridge —** Provides communication with devices that support the Java SNMP (Simple Network Management Protocol). See the *G2-SNMP Bridge User's Guide*.
  - **G2 Java Socket Manager —** Provides communication with Java Sockets. See the *G2 Java Socket Manager User's Guide.*
  - **G2 CORBALink —** Provides communication with CORBA. See the *G2 CORBALink User's Guide.*

- **G2 WebLink —** Provides communication with HTTP, the protocol of the World Wide Web. See the *G2 WebLink User's Guide*.
- **G2-HLA Bridge —** Provides an interface to the Modeling and Simulation (M & S) High Level Architecture (HLA). See the *G2-HLA Bridge Users' Guide*.

2

The Developer's Environment

Introduces features and strategies for developing a G2-based application.

Introduction

This chapter shows you how to interact with G2 as an application developer, and how to design an application that uses G2's major computational features.

Capturing Knowledge in a Knowledge Base

You implement a G2 application by using G2 to develop one or more knowledge bases (KBs). These KBs will be delivered with G2 licenses (and perhaps with other Gensym products) to provide an intelligent solution, dedicated or distributed, to a knowledge-management need.

G2's **developer's environment** refers to the default set of features that are available when you use G2 under a development license. You use these features to define items, as well as their properties and behaviors, and to organize them into a **knowledge base** (or KB). We use the word *knowledge* to mean information that is structured and specified so that a running G2 can reason about it.

The set of knowledge that a running G2 contains is called the **current KB**. After G2 starts, it always has a current KB. That is, a portion of G2's memory is always reserved to hold the items that currently represent the knowledge you have collected and organized. At all times, the current KB contains a set of **system tables**, which represent your current preferences for how G2 works with the KB.

G2 executes, or runs, the current KB. You can start, pause, resume, reset, and restart (that is, reset and start as one command) the current KB.

You can save the current KB's knowledge into a new or existing **KB file,** or, more typically, into multiple KB files which capture your KB knowledge in configurable modules. G2 does not alter KB files until you save the current KB into it.

You can also load or merge a KB into G2 from a KB file that you previously saved. You can load one KB, or more than one KB, into G2 at the same time.

For more information about the features of KBs, see Knowledge Bases.

Using Computational Features in G2

You interactively operate the overall execution of the current KB, while G2 automatically maintains the KB's execution-related knowledge. By *execution-related knowledge* we mean the current knowledge of the KB items, the communications status of interface items to external systems, and the state of each executable item that has been invoked. G2's **executable items** include procedures and methods, rules, action buttons, and user menu choices.

The G2 **scheduler** schedules and manages all of the activities required to execute the current KB. The scheduler has settable properties, many of which reside in the

Timing Parameters system table. G2's scheduler also queries the real time via your computer's own clock.

The G2 **inference engine** and other G2 components perform the KB's rules, provide data service for the KB's variables, call foreign functions in other processes, and support remote procedure calls (RPCs) to and from other processes across your computer's network.

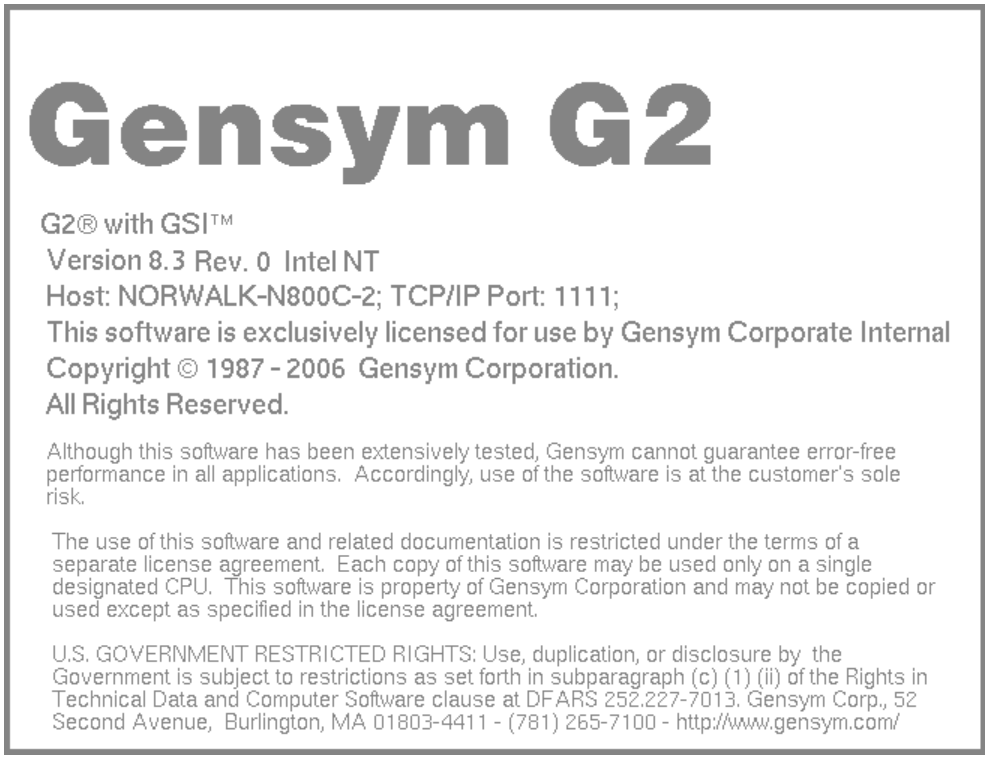
Starting G2

For details on starting G2, see Appendix A, Launching a G2 Process.

For platform-dependent information, see the `readme-g2.html` file and the *G2 Bundle Release Notes*.

The G2 Title Block

By default, G2 displays a **title block** during startup, as the following figure shows:



The title block displays:

- The version of G2.

- Your platform (or combination of computer model and operating system), identified when G2 was installed.
- The network identifier for the host machine.
- The TCP/IP port number on which this G2 listens for connections from other processes across your network.
- The machine ID of the host machine (unless a site license is in use).
- The expiration date of the license (unless a permanent license is in use).

For information about displaying the G2 title block, see Displaying the Title Block.

Customizing the Gensym Background

By default, G2 and Telewindows display a light gray background. You can change the background color and pattern of your local window to a solid color or to a gray-and-white tiling pattern derived from an image file you specify. The image file must contain fewer than 128x128 pixels.

To change the background pattern of your local window to a solid color:

➔ Launch your G2 or Telewindows process with the `-background` command-line option followed by a color symbol.

Examples are:

```
g2 -background red

tw -background dark-slate-blue
```

To change the background pattern of your local window to a gray-and-white tiling pattern:

➔ Launch your G2 or Telewindows process with the `-background` command-line option followed by the file path of a GIF or XBM image file.

Examples are:

```
g2 -background /home/ghf/gifs/tile.gif

tw -background C:\development\kbs\system.xbm
```

If you find that your full-color image file does not result in an acceptable pattern, try reducing the image to black and white by applying a graphics program technique such as ordered dithering before you import the image into your G2 or Telewindows process.

Note G2 and Telewindows display the default gray pattern when you specify a color that is not in the G2 color palette, or when you specify a file that is not in GIF or XBM format or contains more than 16,384 pixels.

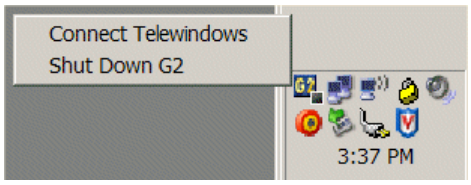
Interacting with the G2 Server Icon on Windows Platforms

On Windows platforms, when the G2 server is running, an icon appears in the taskbar notification area. The tool tip shows the host and port of the G2 server, for example, TCP_IP:host:1111. The icon has a popup menu with these choices:

- Connect Telewindows, which locates Telewindows in the registry and automatically connects a Telewindows session to the G2 server.
- Shut Down G2, which shuts down the G2 server and any connected Telewindows sessions.

If Telewindows Next Generation (`twng.exe`) is registered, the Connect Telewindows menu choice uses that. If it cannot find Telewindows Next Generation, it uses the registry location of Telewindows (`tw.exe`).

This figure shows the icon with its menu:



The G2 server icon looks like this, depending on the G2 run state:

Telewindows Next Generation and Telewindows are both registered automatically when you install the G2 Bundle. However, if G2 cannot find either Telewindows for some reason, you can run the `-regserver` command-line option.

To connect Telewindows instead of Telewindows Next Generation when using the Connect Telewindows menu choice, you can run the `-unregserver` command-line option to unregister Telewindows Next Generation. For details, see regserver.

To start G2 without the icon, use the `-no-tray` command-line option.

Note When running G2 as a service, you must start G2 with the `-no-tray` command-line option to suppress the icon; otherwise, an error occurs when you start G2 as a service.

Exiting from G2

You can quit G2 only if it is paused or reset.

To quit from G2:

➔ Choose Main Menu > Miscellany > Shut Down G2.

The G2 process closes its window and terminates execution.

Interacting with G2

After G2 starts, you can:

- Begin entering new knowledge interactively,
- Load a stored KB file, then
- Work with the loaded knowledge.

G2 provides a highly interactive, graphical, and customizable environment for collecting and organizing knowledge:

- It is graphically *interactive* because you use your computer's keyboard and mouse to work with items that are visible on the screen.
- It is *customizable* because G2 supports three window styles and provides several groups of features that suppress or replace the default behavior of the KB's items and of the menus and non-menu operations of the developer's environment itself.

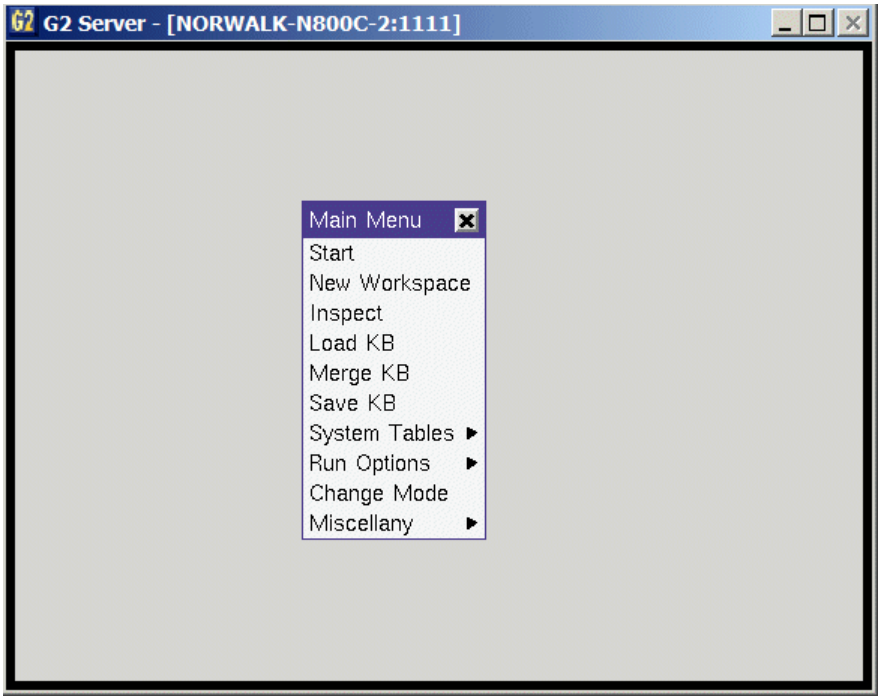
By default, after selecting an item that appears on the screen, you choose from a **menu**. The menu lists operations that are relevant to that item. You can also select from other menus that affect developer's environment settings, such as whether G2 displays long menus or short menus or automatically highlights invoked rules.

In various ways you can customize how the application's users, and how other G2 developers, work with the items in the current KB:

- You can code your KB so that it programmatically changes the appearance and features of the items that the user interacts with.
- You can also declare configurations on items that refer to a **user mode**, which is an identifier that specifies a level of access or degree of functionality that is associated with particular users.

For details, see Actions and Configurations.

G2 Window Styles



The style of the Main Menu above is called standard. It is the default window style. G2 supports two other styles: a large version of the standard style called standard-large and the G2 style before version 6.0, g2-5.x. Your style choice determines the appearance of workspaces, menu and attribute tables, text and icon edit boxes, and temporary workspaces such as Inspect workspaces. Item icons are unaffected by window style.

The standard window styles are characterized by:

- A blue title bar which displays either its Workspace's name, if it has one, or the class of the workspace, unhyphenated, with mixed case (e.g., its menu-style text not its prior attribute-table header style text), and a delete/hide button. Selecting the button hides workspaces, and deletes tables, edit boxes, and temporary workspaces. Clicking outside the button brings the item to the top of the display hierarchy.
- A light-gray background, except for workspaces which, by default, have a white background.

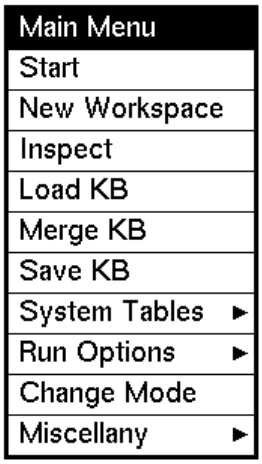
Window-Style Menu Examples

Here is the Main Menu shown in the three styles:

standard

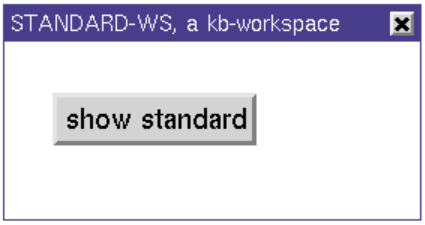
standard-large

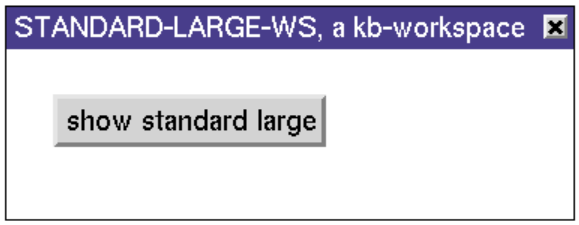
g2-5.x

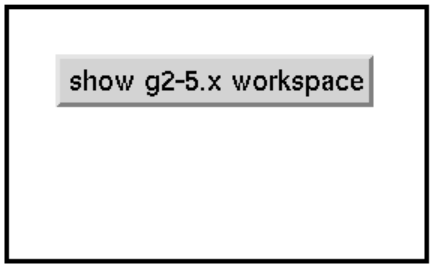


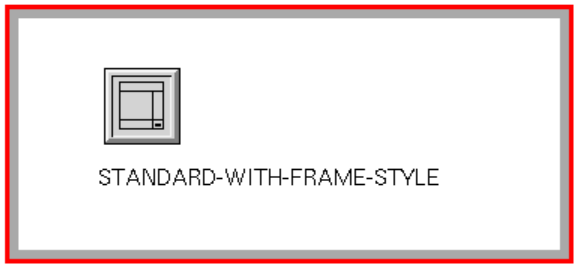
Window-Style Workspace Examples

Here are examples of workspaces in the three window styles, as well as a standard-style workspace that has a user-defined frame-style which replaces the title bar:





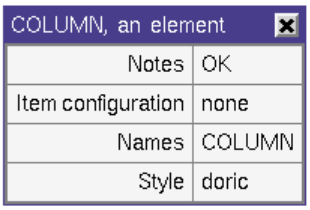




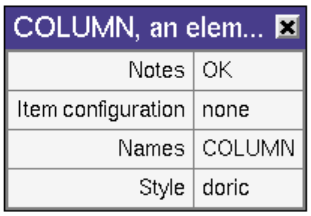
Window-Style Attribute Table Examples

This example shows an attribute table in the three styles:

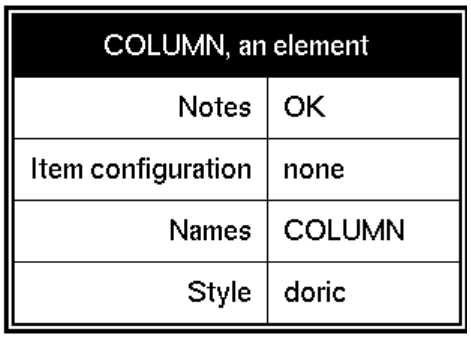
standard



standard-large



g2-5.x



Specifying Window Styles

G2 provides you with the ability to control the window style at several levels. The options are to:

- Use the default window style, which is standard.
- Specify the default window style for the G2 process by setting the system table Server Parameters g2-window-style attribute. This sets the window-style for the G2 process.

  You can edit this attribute either interactively, programmatically, or by using the `-window-style` command-line option when you launch G2.

  The g2-window-style attribute setting persists in the G2 process until you explicitly change it because, unlike other system tables, the Server Parameters table does not lose its non-default attribute values when the KB is cleared.
- Specify the window style for a particular KB by setting the system table Miscellaneous Parameters default-window-style attribute. This sets the window style for the KB, overriding the Server Parameters setting and the default setting for the KB.

  Using the new standard and standard-large window styles with KBs or modules that were saved in G2 5.x. could break them. To avoid this, you should set the window-style to g2-5x. To preserve maximum compatibility with KBs previously saved in G2 5.1 or earlier, G2 automatically sets the default-window-style to g2-5x whenever you load a 5.x KB or module. Although not recommended, you can override this setting.
- Specify the window style for the Telewindows connection or local G2 window by setting the system table This Window g2-window-style attribute. This sets

the window style for the g2-window item associated with your interaction with G2, making it possible for each process that is interacting with G2 to establish its own window style.

The order of precedence for the window style setting is:

1. The Telewindows connection or local G2 window.
2. A KB.
3. The G2 process.
4. G2 default, which is standard.

If any of the above is set to default, G2 uses the window style setting of the next item down.

For example:

- If the local G2 window (1) and the KB (2) are set to default, and the G2 process (3) is set to g2-5.x, then G2 will use the g2-5.x window style.
- If the local G2 window (1) is standard-large, the KB (2) is default, and the G2 process (3) is g2-5.x, G2 will use the standard-large window style.

Note Setting the window-style affects newly created windows only. It does not affect existing windows.

Establishing a Default Window Style for the G2 Process

You change the default style for the G2 process by editing the g2-window-style attribute of the Server Parameters system table. You can edit this attribute either interactively, programmatically, or by using the `-window-style` command-line option when you launch G2. You can also launch Telewindows, using this command-line option.

You can also specify this command-line option when launching a Telewindows process. For more information, see the *Telewindows User's Guide*.

To specify the default window style interactively:

1. Select Main Menu > System Tables > Server Parameters.
2. Edit the g2-window-style attribute to one of these four values:

   default, standard-large, g2-5.x, or standard

To specify the default window style by using a command-line option:

➔ Launch G2 with the `-window-style` command-line.

For example:

```
g2 -window-style standard-large
```

Overriding the Default Window Style for a Particular KB

You specify the window style for a particular KB by setting the system table Miscellaneous Parameters default-window-style attribute.

To specify the window style for a KB:

1 Select Main Menu > System Tables > Miscellaneous Parameters.

2 Edit the default-window-style attribute to one of these four values:

default, standard-large, g2-5.x, or standard

Overriding the Default Window Style for the Current Window

You specify the window style for the Telewindows connection or local G2 window by setting the system table This Window g2-window-style attribute.

To specify the window style for your interaction with G2:

1 Select Main Menu > System Tables > This Window to access the g2-window associated with your G2 or Telewindows process.

2 Edit the g2-window-style attribute on your g2-window item to one of these four values:

default, standard-large, g2-5.x, or standard

Editing Title Bar Text

You can edit the text of the title bar by editing the title-bar-text attribute of a kb-workspace. The text can be entered as a string, with quotes, or as an expression to display the workspace name, class, or table header. When no name exists, the expression can use a default.

Here are some examples:

- "My Workspace Title" shows the text My Workspace Title.
- "" shows a blank title bar.
- the name if any otherwise "unnamed" shows the workspace name, if any, or the text unnamed.
- the name if any otherwise the class shows the workspace name, if any, or the class name, the default.

If a workspace does not show a title bar, this attribute has no effect.

You can override this attribute for user-defined subclasses of kb-workspace. You can get and set the exported internal representation of the title-bar-text attribute, using the attribute access facility.

Syntax

The syntax for title-bar-text has this format:

default |
simple-option |
conditional-option [if any, otherwise *simple-option*]

The symbol default indicates that G2 should use its default setting, which you cannot currently change. The default is equivalent to the name if any, otherwise the class. In a future release, we may allow the default setting to be changed by the user.

simple-option is one of:

string | the class | the table header

The *string* option displays the literal text string in title bar.

The phrase the class displays the workspace class name in the title bar, which is formatted without hyphens and in title case, that is, with initial capitalization and most other characters in lowercase, except KB, which always appears in upper case.

The phrase the table header displays the table header of the workspace in the title bar. This is the same text that would appear in the header of a table for the workspace.

conditional-option is:

the name

The phrase the name displays the workspace name, if any, in the title bar. If there is more than one name, the first name is used. If there are no names, then the title bar is blank, unless an addition option is specified, using the phrase if any, otherwise *simple-option*.

Note that you cannot include expressions to evaluate in any of the options, using the [ ] syntax.

Attribute Access

You can get and set values for the title-bar-text internal attribute, using the attribute access facility. You can set the value by using a simple conclude statement or by concluding the value into a sequence. You must use a sequence when concluding the value, using a phrase such as the name if any otherwise "unnamed". Otherwise, the use of sequences is optional.

For more information about using this facility, see Attribute Access Facility.

The attribute access format for setting the title-bar-text attribute value is one of the following:

Title Bar Value Type	Setting Attribute Access Value
Blank	conclude that the title-bar-text does not exist
Text string	conclude that the title-bar-text = *string*
Class name, table header, name	conclude that the title-bar-text = the symbol class conclude that the title-bar-text = the symbol table-header conclude that the title-bar-text = the symbol name
Empty sequence	conclude that the title-bar-text does not exist
Text, class name, table header, name as a sequence	conclude that the title-bar-text = sequence(*string*) conclude that the title-bar-text = sequence(the symbol class) conclude that the title-bar-text = sequence(the symbol table-header) conclude that the title-bar-text = sequence(the symbol name)
the name if any otherwise *simple-option*	conclude that the title-bar-text is sequence(the symbol name, "unnamed")

Using Menus to Operate the Current KB

After G2 has started and if the current KB contains knowledge that you want to work with, you can **operate** the KB, which means to use G2's default menus to start, pause, resume, reset, or restart the current KB.

Because you can easily operate the current KB, you can quickly test and determine the effects of changes in the KB's items. Note, also, that you can make many changes to your KB's items, including in the definitions of **classes**, while the KB is running. For more information, see Knowledge Bases.

Starting the current KB causes G2 to perform several standard tasks, all related to **activating** some or all of the KB's knowledge. Activating an item causes G2 to do something with it, based on the item's class. Activation of items is described in Workspaces.

Note To perform an operation *programmatically* means that you perform it by invoking an executable item. To perform an operation programmatically requires that the current KB is running.

Pausing and *resuming* the current KB does just that. No knowledge about the status of executing items is lost due to pausing the KB.

Resetting the KB means to restore all knowledge in the KB to its initial state.

Restarting the KB means to reset the KB then start it, in one command.

Using Menus to Operate on an Item in the KB

By default, you use menus to interact with items in the current KB. We say "by default," because you can use **configurations** to suppress the display of any default menu or any default menu choice available in the developer's environment.

To work interactively with a particular item in the current KB, click the mouse on the item to display its menu. The menu choices that are common to the KB's **items** are described under Using Item Menus.

Using Menus to Affect the Developer's Environment

You also use menus to interact with the G2 developer's environment. To change a feature or setting in the developer's environment, select from the G2 Main Menu and from the Miscellany menu.

To display the G2 Main Menu:

➔ Click the mouse on the background of the G2 window.

Choices on the Main Menu

By default, the G2 Main Menu displays these choices:

- Change Mode: Displays the login dialog.
- Get Workspace: Brings an existing kb-workspace to the top of the display hierarchy.
- Inspect: Opens the Inspect facility.
- Load KB, Merge KB, and Save KB: Loads or merges a KB or save the current KB.
- Miscellany: Displays the Miscellany menu.
- New Workspace: Creates a new kb-workspace.

- Run Options: Displays a menu from which you can select options that affect how G2 runs the current KB.
- System Tables: Displays a menu from which you can select the system tables for the top-level module.
- Start, Pause, Resume, Reset, and/or Restart: Changes the G2 run state, depending on the current state.

Choices on the Miscellany Menu

By default, the Miscellany menu displays these choices:

- Clear KB: Clears the current KB.
- Create New Module: Creates a new module in the current KB.
- Connect to Foreign Image/Disconnect from Foreign Image: Connects this G2 to or disconnects this G2 from a separately developed C or C++ program.
- Delete Module: Deletes a module from the current KB.
- Enter Package Preparation Mode and Simulate Package Preparation Mode: Enters or simulates G2's package preparation mode.
- Load Attribute File: Loads a file, called an attributes file, that populates the attributes of existing items in the current KB. Attribute files are a superseded capability. For more information see Appendix F, Superseded Practices.
- Neatly Stack Windows: Relocates the currently displayed application windows, or **workspaces**, into a cascading arrangement.
- Network Info: Displays this G2 process's network information.
- New Title Block: Display G2's title block.
- Short Menus/Long Menus: Selects long menus or short menus.
- Shut Down G2: Exits G2.
- Write G2 Stats: Creates a statistics file related to memory usage.

Clearing the KB

Clearing the current KB means to delete all knowledge from the current KB and to reset all system tables, except the Server Parameters system table, to their default values.

Creating a New Module and Deleting a Module

Creating a new module means to add a new module item and its associated set of system tables to the current KB's **module hierarchy**. This module *cannot* serve as the current KB's top-level module. Deleting a module means deleting a module

item, and, optionally, the workspaces associated with the module. For more information about modules, see Modularized KBs.

Connecting to and Disconnecting from a Foreign Image

Connecting to a foreign image means to establish a network connection with a running executable image, whose procedures the current KB's procedures can invoke. Disconnecting from a foreign image means to break a network connection that was previously established. For more information, see Foreign Functions.

Entering or Simulating Package Preparation Mode

Entering **package preparation mode** means to set G2's developer's environment so that you can prepare the current KB for customer distribution. Simulating package preparation mode means to set G2's developer's environment so that your G2 behaves as if it were authorized to run the proprietary current KB.

For more information about using these menu choices, see Package Preparation.

Neatly Stacking Windows

Neatly stacking windows means to relocate the current KB's visible workspaces so that they appear to cascade from the upper left corner of G2's window.

Displaying Network Information

Displaying network information means to display the host name and TCP/IP port number on which this G2 listens for connections from other processes across your network. This information also appears in the G2 title block.

You can start G2 with the network information in the title bar of the window by using a command-line option. For details, see `name`.

Displaying the Title Block

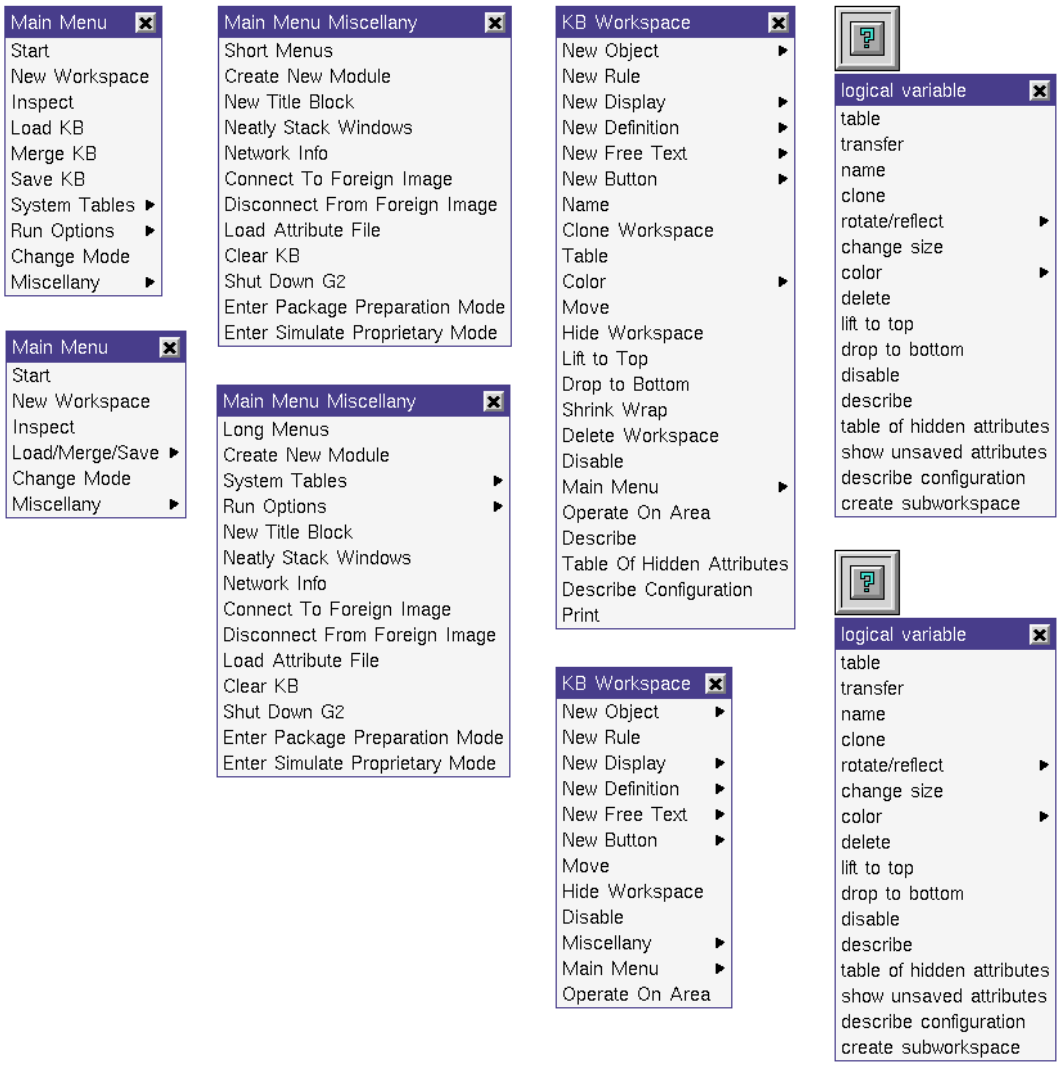
This means to display G2's title block, as shown in Starting G2.

Selecting Long or Short Menus

By default, G2's developer's environment presents all menu choices on **long menus**. You can alternatively select G2's default menu choices from **short menus**, which display more of the default choices in submenus.

The next figure shows the default G2 menus when long menus and short menus are in effect. Notice in the figure that selecting long menus or short menus affects the display of the Main Menu, the Miscellany submenu, and the KB Workspace menu, but does not affect the display of the item's menu.

Default long menus are on top with their equivalent short menus below them.



To use long menus to interact with G2:

➔ Select Main Menu > Miscellany > Long Menus.

To use short menus to interact with G2:

➔ Select Main Menu > Miscellany > Short Menus.

Shutting Down G2

Shutting down G2 means to **exit G2**. Shutting down causes G2 to interrupt and end all its processing, close any open files, and release its resources to your computer's operating system.

Navigating KB Knowledge

After the current KB contains some number of items, you will need a convenient way to navigate the KB's class and workspace hierarchies and to find particular items. G2's **Inspect facility**, another feature of G2's developer's environment, provides this capability.

The Inspect facility is described in The Inspect Facility.

Notifying the User of Errors

An **error condition** is any unintended or unexpected discrepancy that occurs while G2 is handling information. G2 can detect some error conditions whether the current KB is running or not.

In general, G2 responds to error conditions by invoking an error handler. An **error handler** manages error signals and produces error messages, if necessary.

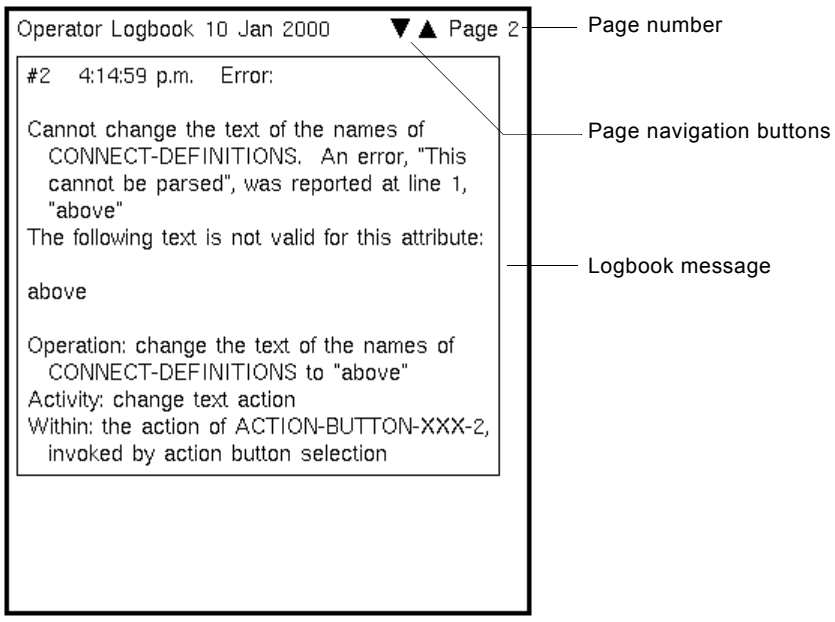
G2 includes a default error handler. When the current KB is reset or paused, G2's default error handler responds to most error conditions by posting a message on the Operator Logbook. After you start the current KB, if an error condition occurs, G2 responds by invoking either its default error handler or a custom error handler defined in the current KB.

G2 error handling is described in Error Handling.

Working with the Operator Logbook

The Operator Logbook is a collection of workspaces that receive error messages from G2's default error handler. The Operator Logbook appears as a set of pages, each of which can contain one or more error and informational messages produced by G2.

The next figure shows one Operator Logbook page. Notice that each page has a header that displays today's date and a page number. Clicking the mouse on one of the two triangles causes G2 to display either the preceding or subsequent Logbook page.



Note Operator Logbook messages are internal to G2, rather than being posted by the user, and therefore are not included in the values of expressions such as the count of each message.

By default, the Operator Logbook displays as a native pane in Telewindows. For more information, see Displaying the Native Logbook.

Hiding and Showing Logbook Pages

You can specify Logbook pages interactively or programmatically.

Specifying Logbook Pages on a G2-Window

The g2-window class includes the show-operator-logbook-in-this-window? attribute, whose value is yes (the default) or no. Leaving the value of this attribute as yes causes the Operator Logbook to be displayed as specified in the Logbook Parameters system table.

Changing the attribute value to no for a window hides all existing Operator Logbook pages in that window. Subsequent messages are recorded in the logbook, but all pages remain hidden. Changing the value to yes again shows logbook pages as specified in the Logbook Parameters system table. The pages look as they would if they had never been hidden.

To hide and show operator logbook pages:

➔ Edit the window's table or use the conclude action to set the show-operator-logbook-in-this-window? attribute of the relevant G2 window to yes to display pages or no to hide them.

For example, the following procedures programmatically show and hide the Operator Logbook pages in the g2-window for the user ghw:

```
hide-pages(window: class g2-window)
begin
if the g2-window-user-name-in-operating-system of window = "ghw"
    then conclude that the show-operator-logbook-in-this-window@? of
        window is false
end

show-pages(window: class g2-window)
begin
if the g2-window-user-name-in-operating-system of window = "ghw"
    then conclude that the show-operator-logbook-in-this-window@? of
        window is true
end
```

Specifying Using the Hide and Show Actions

To execute the show and hide actions on logbook pages:

1. Enable the executable item containing the show or hide statement to refer to inactive items by setting the may-refer-to-inactive-items attribute of the evaluation-attributes attribute to true. You can do this from the hidden attributes table of the executable item.
2. Execute hide every log-book page or show every log-book page.

Limiting the Number and Size of Logbook Pages

You can conserve G2's use of memory (specifically, its region 1 memory) by keeping fewer Operator Logbook pages in memory, and by limiting the number of logbook messages allowed per page. The number of Operator Logbook pages can quickly accumulate, because G2 automatically writes a message to the Operator Logbook workspace each time you start, pause, or reset the current KB.

To limit the number of Operator Logbook pages:

➔ Set the maximum-number-of-pages-to-keep-in-memory attribute in the Logbook Parameters system table to a small number, such as 4 or 5.

To limit the size of Operator Logbook pages:

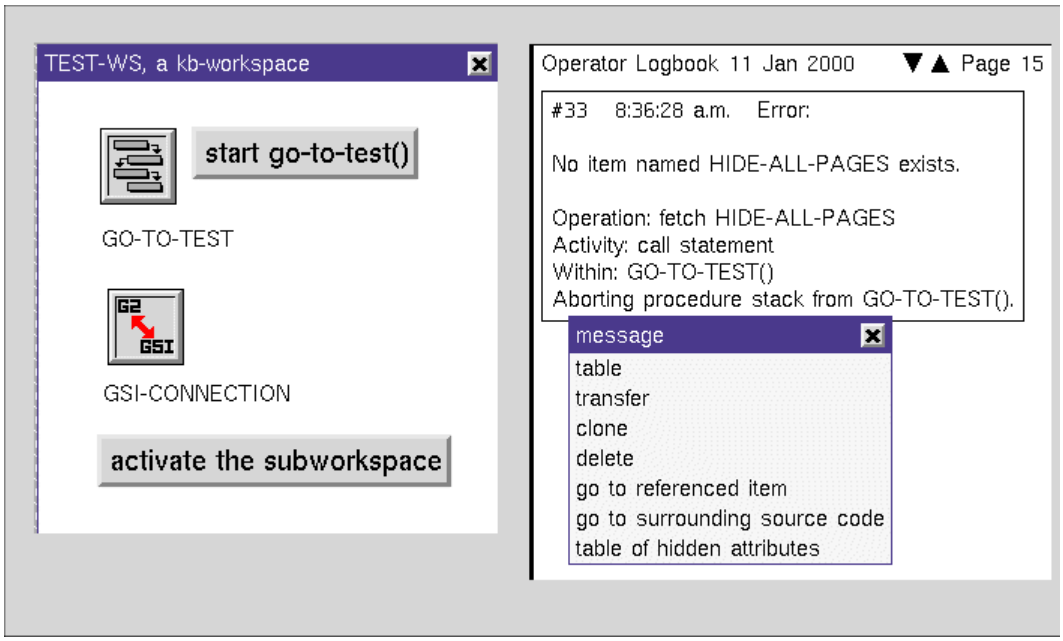
➔ Set the width-for-pages and height-for-pages attributes in the Logbook Parameters system table.

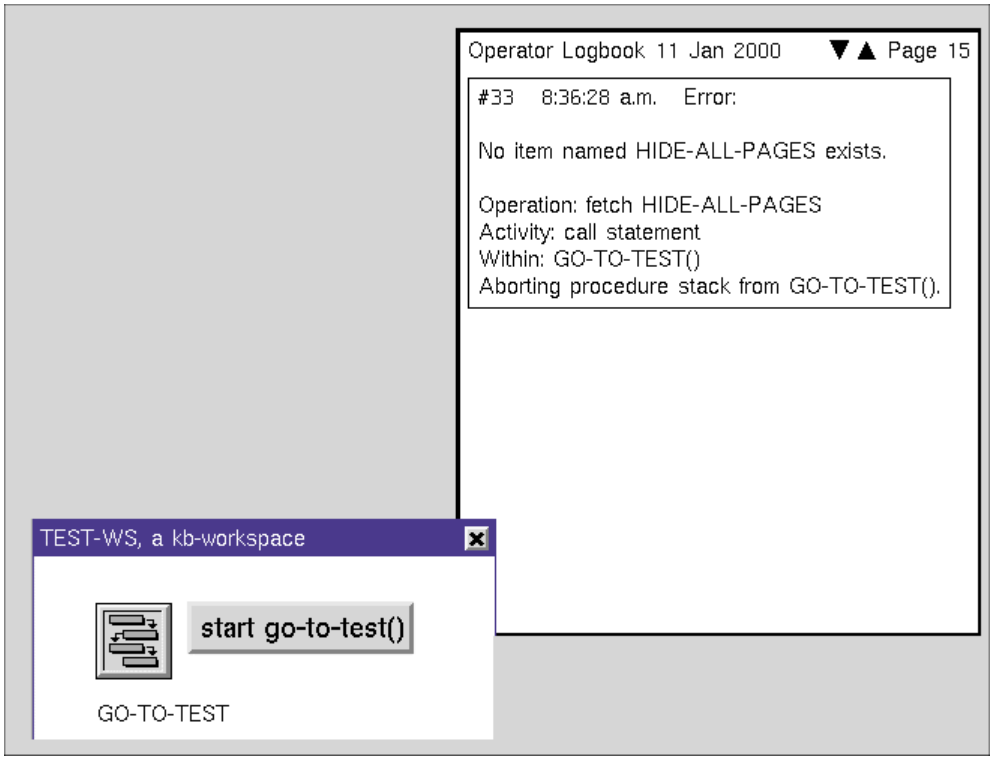
Navigating to an Item Referenced in an Operator Logbook Message

Each Operator Logbook message that references at least one item in the current KB includes the go to referenced item menu choice. Selecting go to referenced item causes G2 to display the workspace that contains the item referenced in the message, and displays the referenced item (shown within its own workspace) in the center of the window.

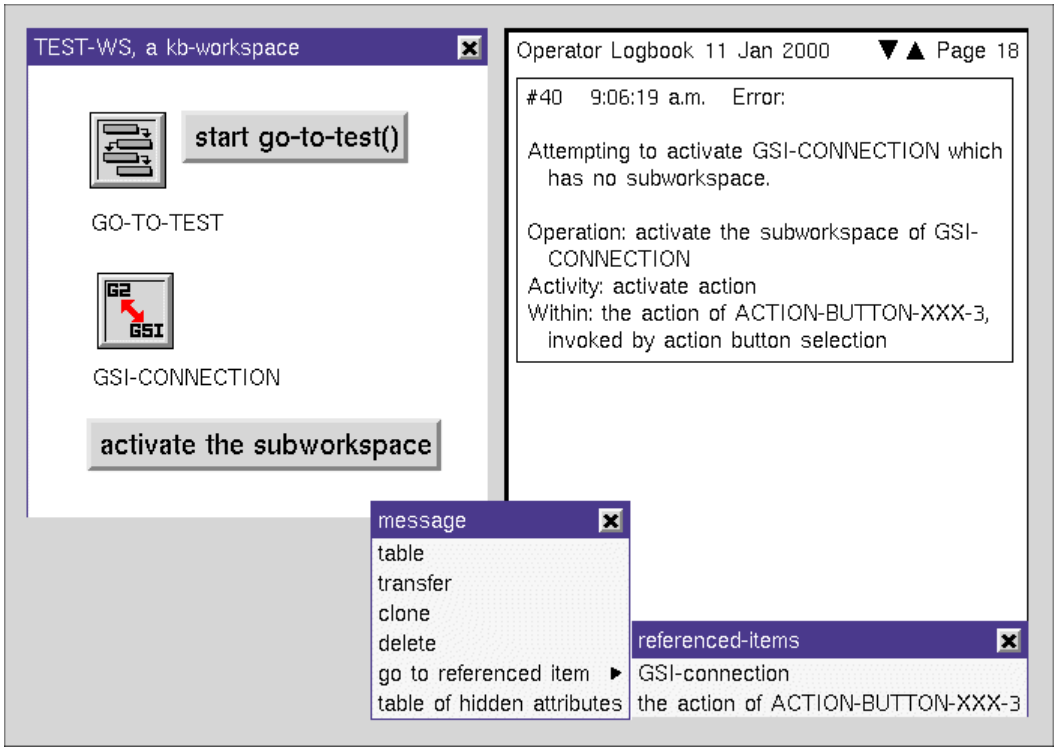
Note The go to referenced item menu choice appears only if the Operator Logbook message references an item on a workspace that is not configured to be **proprietary**. Package Preparation describes proprietary workspaces.

Use the go to referenced item menu choice with care. As shown below, G2 relocates the referenced item's parent workspace so that the item appears at the center of the window. This might disrupt the visual organization of a KB whose workspaces have been carefully positioned. G2 also displays that item's parent workspace at its full scale, as described under Scaling a Workspace.Before selecting go to referenced item, your window might appear like the top grouping; and after selecting go to referenced item, G2 displays the referenced action button at the center of the window.


TEST-WS, a kb-workspace
start go-to-test()
GO-TO-TEST
G2
GSI
GSI-CONNECTION
activate the subworkspace
Operator Logbook 11 Jan 2000
Page 15
#33 8:36:28 a.m. Error:
No item named HIDE-ALL-PAGES exists.
Operation: fetch HIDE-ALL-PAGES
Activity: call statement
Within: GO-TO-TEST()
Aborting procedure stack from GO-TO-TEST().
message
table
transfer
clone
delete
go to referenced item
go to surrounding source code
table of hidden attributes


Operator Logbook 11 Jan 2000
Page 15
#33 8:36:28 a.m. Error:
No item named HIDE-ALL-PAGES exists.
Operation: fetch HIDE-ALL-PAGES
Activity: call statement
Within: GO-TO-TEST()
Aborting procedure stack from GO-TO-TEST().
TEST-WS, a kb-workspace
start go-to-test()
GO-TO-TEST

If you select an Operator Logbook message that references more than one item, the go to referenced item menu choice leads to a submenu, from which you can choose a particular item, as shown in this figure:



If an Operator Logbook message does not refer to an item, G2 does not offer the go to referenced item menu choice in the message's menu.

Navigating to the Procedure Code That Causes an Error

By default, G2 generates compiled-code to source-code identification information when it compiles your procedure code. When your procedure code causes an error, you can select the go to source menu choice from the message. G2 then opens a text editor on the procedure, and places a cursor within the statement that caused the error. For the details of this facility see Obtaining Procedure Source-Code Error Location Information.

Shadowing the Operator Logbook Message Handler

When G2 posts a message to the Operator Logbook, it does so by calling an Operator Logbook message handler and passing it the message. The system-defined handler posts the message to the logbook.

You can shadow the system-defined handler with any procedure that takes one argument of type text. Such a procedure is called a **user-defined Operator Logbook message handler**. Once registered, such a handler receives all messages that would otherwise go to the system-defined handler and be posted to the Operator Logbook. Such messages do not appear on the logbook, and are not recorded in the log file, if any.

If any activity of a user-defined Operator Logbook message handler causes G2 to post a message to the Operator Logbook, the request goes to the system-defined handler, which posts the message as if no user-defined handler had been registered.

Resetting G2 does not affect handler shadowing: any handler registered remains in effect when G2 restarts.

Note The following procedures provide low-level operator logbook message handling. More sophisticated techniques are available through GFR. See the *G2 Foundation Resources User's Guide* for details.

To register a logbook message handler:

➔ g2-register-logbook-message-handler
(*procedure*: class procedure)

Registers the procedure to handle all logbook messages.

To deregister a logbook message handler:

➔ g2-deregister-logbook-message-handler
( )

Deregisters the currently registered logbook message handler. The system-defined handler is again in effect.

To get the logbook message handler:

➔ g2-get-logbook-message-handler
( )
-> {*handler*: class procedure | false: truth-value}

Returns the procedure currently registered as the logbook message handler, or false if none is registered.

Each of these procedures is described in more detail in the *G2 System Procedures Reference Manual*.

Working with the Message Board Workspace

The Message Board is a system generated workspace (not a kb-workspace) that G2 creates automatically the first time that G2 executes a post or inform the operator action.

G2 automatically activates the Message Board when it is created, and it remains active whether the current KB is running or paused.

The currently installed Message Board Parameters system table determines the size and settings for the Message Board.

By default, the Message Board displays as a native pane in Telewindows. For more information, see Displaying the Native Message Board.

Shadowing the Message Board Message Handler

When G2 posts a message to the Message Board, it does so by calling a Message Board message handler and passing it the message. The system-defined handler posts the message to the board. The posted messages are deleted when you reset G2. The messages on the Message Board, but not the Message Board itself, can be manually transferred to a kb-workspace.

You can shadow the system-defined handler with any procedure that takes one argument of type text. Such a procedure is called a **user-defined Message Board message handler**. Once registered, such a handler receives all messages that would otherwise go to the system-defined handler and be posted to the Message Board.

If any activity of a user-defined Message Board message handler causes G2 to post a message to the board, the request goes to the system-defined Message Board handler, which posts the message as if no user-defined handler were in effect.

A user-defined Message Board message handler can itself post messages to the Message Board. G2 passes such a request to the system-defined handler, which posts the message just as if no user-defined handler had been registered.

Resetting G2 does not affect handler shadowing: any handler registered remains in effect when G2 restarts.

Note The following procedures provide low-level message board message handling. More sophisticated techniques are available through GFR. See the *G2 Foundation Resources User's Guide* for details.

To register a message board handler:

➔ g2-register-message-board-message-handler
(*procedure*: class procedure)

Registers the procedure or method to handle all message board errors.

To deregister a message board handler:

➔ g2-deregister-message-board-message-handler
( )

Deregisters the currently registered message board message handler.

To get the message board handler:

➔ g2-get-message-board-message-handler
( )
-> *handler*: class procedure | false: truth-value

Returns the procedure or method currently registered as the message board message handler, or false if none is registered.

Each of these procedures is described in more detail in the *G2 System Procedures Reference Manual*.

Organizing KB Knowledge

As you add knowledge to your KB, you will find it important to organize that knowledge in various ways and for different purposes.

G2 provides three ways to organize knowledge globally in a KB: by class, by workspace, and by module. Each of these organizing techniques is global, because you can reference each item in your KB only by its class, only by its location within the KB's workspace hierarchy, or only by its association with a module.

Distinguishing Functional Behavior by Class

Use the organization of the classes defined in your KB as a primary determinant of how your KB behaves. That is, your KB's programmatic behavior should take advantage of the object-oriented feature of inheritance that is built into G2's class hierarchy. Because a G2 developer has control over the organization of only the KB's user-defined classes, it is especially important that the organization of these classes reflect your application's functional requirements.

A standard way to use the class-orientation of your KB's items is to use methods. By coding your KB's programmatic activities as methods, you can reuse the procedural knowledge that is associated with a more generic class as you define

the procedure knowledge required for a more specific class. For more information about using G2's methods, see Methods.

You can also use the inheritance paths defined in the KB's class hierarchy to configure the behavior of the KB's items. Instance configurations affect the default behavior of the KB's items, based on each item's class. For more information about instance configurations, see Configurations.

Using Workspaces to Organize KB Knowledge

For your own convenience, you can organize your KB's items into collections by placing each upon a workspace (that is, upon items of the kb-workspace class). A workspace both contains a set of items (some of which can have their own subworkspaces) and establishes their arrangement as a set when the workspace is displayed. For more information, see Workspaces.

Although you can organize your KB's items in any manner you prefer, you should develop guidelines for how you arrange your KB's items among a set of workspaces.

For instance, if you are developing your KB to use methods as the primary programmatic items, you might create one workspace for each user-defined class. Upon this workspace you might place:

- The definition item for the class.
- Each method that you define for the class.
- The rules that refer only to instances of the class.
- Free texts and other items that contain the unchanging information for this workspace's methods and rules.

Partitioning Knowledge into Modules

G2 supports a further level of organization for your knowledge base, called **modules**. Each module is an item that is associated with its own set of system tables and with one or more top-level workspaces. Partitioning a large knowledge base into modules allows you and others to develop and maintain the modules in a more manageable fashion.

After you create a module in the current KB, you can save, as a unit, the module item, its system tables, its associated workspaces, and all items below those workspaces in the KB's workspace hierarchy, into a distinct KB file.

You can design your modules to have well-defined dependencies upon each other, or to have no dependencies upon each other at all (other than directly required module dependencies). For more information, see Modularized KBs.

Planning Your Work

G2 offers many features that support building real-time applications. As you design your G2-based application, you must evaluate how to put these features to work. This section takes a broad view of G2's features and indicates why and how they are relevant to your work as an application developer.

Configuring the Default Developer's Environment

As you develop your KB, keep in mind that you can customize the interface to the KB's items either by suppressing or by supplementing the default features of G2's developer's environment. G2 allows you to accomplish this by declaring configurations that are stored with your KB.

Whether you do this, and to what degree, depends upon the intended users of your KB. For instance, you might develop a set of KBs for use only by other G2 developers; whereas, another G2 developer might develop a set of KBs that are intended to work together as a complete application. The features that another G2 developer requires (such as editing procedures, changing the definitions of classes, and so on) are probably inappropriate to deliver to users of most applications.

Prototyping or Engineering

You can use G2 to very quickly develop a working prototype of an application. G2 easily supports a prototyping development approach to developing your application. G2's key features, such as its syntax-driven Text Editor, its structured-English programming language, its iconic and object-oriented class hierarchy, and its incremental development environment, all support rapid development and deployment of applications. You can complete your work in a small fraction of the time required using other software development technologies.

However, your application will also benefit from your taking a more disciplined, engineering-oriented approach to your application development project. For instance, developing a robust class hierarchy, with the associated data servers, methods, rules, and configurations that support the application's items, can consume a significant portion of a project's time and effort.

Identifying Roles for Workspaces

You can create a set of workspaces where you capture and organize the definitions and items that form the backbone of your application. Typically, these workspaces should not be available to the application's users.

Thus, while designing your application's user interface, you must determine how to use workspaces to display the KB's knowledge to the application's users. For

instance, your KB might be designed to contain static workspaces: workspaces of fixed sizes and with more or less fixed relationships to each other. The application uses these workspaces as the areas within with to display the KB's knowledge and within which to allow user interaction.

Alternatively, you could design your KB to create and display workspaces and their contents dynamically. That is, the stored KB that you deliver as the application might not include any static, user-visible workspaces at all. Rather, at run-time the application creates the workspaces that the user must see based only on run-time conditions.

Identifying the User Interface Paradigm

Your application can utilize the user interface features of the G2 developer's environment, or you can develop a different user interface paradigm. That is, your KB can display menus for its items in the same manner as G2 does by default. Or, you can design new kinds of operations and items that implement the same features in what is called a **direct-manipulation paradigm**.

One approach is to create items that display their primary knowledge graphically and directly. For many kinds of applications, you can create items with which the user interacts directly, without the need for selecting commands from menus.

For example, assume that you have developed a class of items that can be the target of three operations: move this item, create a copy of this item, and delete this item. You can use G2 actions and configurations to accomplish this, without requiring the use of a menu-based interface. That is, you can configure the items to respond to these user actions:

- Move this item operation: When the user clicks the mouse pointer over an item, attach that item to the mouse pointer. Drop the item upon the workspace where the user next clicks the mouse.
- Create a copy of this item operation: When the user clicks the mouse pointer over an item while also holding down the Shift key, create a copy of that item and attach it to the mouse pointer. Drop the new item upon the workspace where the user next clicks the mouse.
- Delete this item operation: When the user clicks the mouse pointer over an item while also holding down the Control key, delete that item.

Under this interface paradigm, where the user drops the item, such as within a particular workspace, can also signify an operation on that item.

User Interface Utilities

G2 includes a number of utilities that provide specific user interface capabilities:

- GUIDE/UIL helps you implement your application's dialogs, navigation buttons, and data validation features with its own time-saving graphical interface. For information see the *G2 GUIDE User's Guide* and the *G2 GUIDE/UIL Procedures Reference Manual*.
- G2 Menu System (GMS) provides extensive capabilities for defining menu bars that appear on workspaces and popup menus associated with items. For information see the *G2 Menu System User's Guide*.
- G2 Dynamic Displays (GDD) provides a number of attractive dials, meters, and displays, based on G2 power icons, which you can use directly or as direct superior classes. For more information, see the *G2 Dynamic Displays User's Guide*.
- G2 Developer's Interface (GDI) enables you to develop a Windows-like GUI using classic G2 capabilities. For information see the *G2 Developer's Interface User's Guide*.
- G2 XL Spreadsheet (GXL) enables you to develop scrolling tabular displays for viewing and editing lists, arrays, and complex data structures. For more information see the *G2 XL Spreadsheet User's Guide*.
- G2 Online Documentation (GOLD) is a set of related modules that implement online documentation and context-sensitive help, based on external browsers and HTML. For more information, see the *G2 OnLine Documentation User's Guide* and the *G2 OnLine Documentation Developer's Guide*.

Other Developer Utilities

G2 provides a number of other useful utilities for the G2 developer:

- G2 ProTools provides advanced G2 developer tools for speeding up development, testing, debugging, documenting, and deployment.
- G2 Foundation Resources (GFR) establishes standard approaches to several important design and implementation issues commonly encountered in building inter-operable modules. GFR helps to assure the compatibility of modules with modules written by other authors who also use GFR.

Identifying Data Servers for Variables

Typically, the **variables** in your KB represent a data point, or series of data points, collected over regular time intervals. Also, some variables represent data points that are collected outside of G2. One important part of the design of your G2-based application is to identify the sources of data, or **data servers**, for each class of variables in your KB.

Variables can have either *internal* or *external* data servers. External data service is a more complex task for your KB to perform than internal data service. For this reason, determining the minimum number of variables that must use an external data server is also a key design decision.

For more information about variables, see Variables and Parameters.

Using Internal Data Servers

G2's internal data servers are the G2 inference engine and the G2 Simulator. If a variable's data server is the inference engine, G2 allows the variable to receive a new value via conclude actions (including those that result from chaining among rules) and from specific formula items. If a variable receives data service from the simulator, G2 associates the variable with a simulation formula, which provides its value.

When a variable has been defined to receive internal data service, your KB can dynamically alternate its service between the G2 inference engine and the G2 Simulator. This allows your KB to respond more flexibly to situations in which a simulated value is just as useful to the application as a value obtained from another source.

The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices.

Using External Data Servers

G2's external data servers are other G2 processes, G2 Gateway bridge applications, and GFI files. If a variable receives external data service, your KB cannot dynamically change its data service. However, your KB can conclude the value of an internally data-served variable from another variable that is externally data-served.

External data service takes place *asynchronously* to the rest of your KB's processing; thus, the activity that your KB must perform to accommodate external data service is inherently more complex than internal data service. For instance, to support external data service, you must create and include interface items in your KB.

For more information, see G2-to-G2 Interface, G2 Gateway, and the *G2 Gateway Bridge Developer's Guide*. GFI is a superseded capability. For information about it, see Appendix F, Superseded Practices.

Using Timekeeping Features

Keeping time is central to how G2 performs its tasks. G2 has awareness of three streams of time: real-time, scheduled time, and simulated time.

Querying the Real Time

After G2 starts and as long as it runs, G2 has awareness of the real time. G2 has this awareness by querying your computer's own clock. G2 uses the capabilities of your computer's operating system to perform these queries. G2 has as precise a grasp of the real time as your computer and its operating system provide.

Scheduling G2's Work

G2 schedules its own work, and the work it performs when running the current KB, using a second time stream. You can adjust the *granularity* of this time stream in the minimum-scheduling-interval attribute of your KB's Timing Parameters system table. Its granularity determines how often G2 checks whether there is more work for it to do.

G2 manages its own work by dividing it into **tasks**, by assigning a priority to each task, and by scheduling a given set of prioritized tasks to be performed at a particular point in future time. G2 calculates that future point in time as the current time plus a multiple of the minimum scheduling interval.

G2 performs only the work that has been scheduled for the current scheduling interval. After G2 performs that work, G2 waits, by default, for the remainder of the minimum scheduling interval (if any) to pass, then it begins performing the tasks scheduled for the next interval.

G2 keeps this time using the **G2 clock**. G2 increments the G2 clock each time it moves from one minimum scheduling interval to the next.

G2 manages its work in this way, because it must manage several threads (independently running G2 tasks) of data processing that take place more or less simultaneously. The tasks that G2 must constantly schedule and perform include, but are not limited to:

- Responding to input from the user.
- Performing each procedure that has been started.
- Performing a procedure that has been called by a started procedure.
- Checking whether a running procedure has exceeded its execution time limit.
- Invoking each rule that is defined to be scanned during this time interval.
- Invoking a rule due to detecting an event.
- Invoking a rule due to chaining.
- Determining whether any variable's value has expired.

- Determining whether any variable expects a new value.
- Inputting data received from an external data server.

Thus, you can set your KB's knowledge so that G2 checks more often or less often whether there is new work waiting to be done.

Determining the Minimum Scheduling Interval

You determine the right minimum scheduling interval for your KB by determining the minimum interval of real time that is significant to your application. For instance, perhaps one class of variables in your KB must be updated as often as once per 0.5 seconds, but none of its rules must be invoked more often than once per 0.5 seconds. Thus, for this KB there is no need to set the minimum scheduling interval to a value less than 0.5 seconds. For this KB, doing so would increase G2's own overhead while adding no more capability to the KB's time-based processing.

You can also adjust the *rate* at which this second time stream elapses. The scheduler-mode attribute of the Timing Parameters system table contains this setting. Specifically, this determines whether G2 waits for the entire minimum scheduling interval to elapse, before moving on to its set of tasks that are scheduled for the next interval.

Use this feature to cause G2 to run your KB unconstrained by the granularity of the minimum scheduling interval. For a KB whose processing does not depend, or depends only minimally, upon the occurrence of events upon which G2 must wait an entire minimum scheduling interval, setting the KB to run in the as fast as possible scheduler mode allows G2 to move from minimum scheduling interval to interval (and thereby increment the G2 clock) as fast as G2 can perform its scheduled work without waiting for real-time based events.

For more information about the G2 scheduler and the G2 clock, see Task Scheduling.

Establishing Simulated Time

G2 can also maintain a distinct time stream for each **simulation model** item defined in the KB. Your KB can establish a distinct current time for each simulation model, to allow the G2 Simulator to represent the occurrence of events independently of the real time. You use G2 system procedures to set and manipulate the time streams of simulation models.

The granularity at which G2 increments this time stream is determined by the default-simulation-time-increment attribute of the Simulation Parameters system table. The rate at which G2 allows this time stream to elapse is determined by the scheduler-mode attribute of the Timing Parameters system table.

The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices.

Establishing Naming Conventions

As for applications coded in other programming languages, you must assign names and identifiers to many entities in your KB, and you should establish conventions for how you derive and assign those names.

G2 allows more than one item to have the same name. However, G2 requires that each user-defined class and relation definition have a unique name.

If you are developing a modularized KB, you might prefer to establish module-based naming conventions for classes, attribute names, user interface items, procedures, and methods.

For more information about naming conventions, see the *G2 Developer's Guide*.

Considering Natural Language Support

As you use G2 to implement an application, you can configure G2's developer's environment to display G2's default menu choices and to display the default text of buttons in the Text Editor and other G2 facilities, using a particular natural language.

You can also use the features of G2's Text Editor to enter a text value that contains any character in any natural language that G2 supports. Using this feature, you can also define language-specific versions of each of your application's custom menu choices.

You can make this Text Editor feature available to your application's users, so that they can enter text into the application in a particular language (English or not), or in any language that G2 supports.

These features are described in Natural Language Facilities.

You can also use ASCII characters to signify any character in any natural language that G2 supports. This scheme for signifying non-English characters is defined as the Gensym character set, which is described in G2 Character Support. Use the Gensym character set to specify text outside of G2 that must be input to G2, and to translate text containing non-English characters that G2 outputs for use by other applications.

Also, Telewindows users can connect to G2 and can operate its current KB such that each displayed Telewindow displays its menu text and G2's system-defined text in a distinct G2-supported natural language. This feature is described in G2-Windows, and in Telewindows Support.

Part II

Global G2 Components

Chapter 3: Knowledge Bases

Shows how to work with the current KB, save the current KB, and load a KB.

Chapter 4: Workspaces

Shows how to use workspaces to organize your KB's items.

Chapter 5: Modularized KBs

Describes how to partition your KB into modules.

Chapter 6: System Tables

Describes the use of system tables to set global preferences.

Chapter 7: Configurations

Describes how configurations override the default behavior of items.

Chapter 8: G2-Windows

Describes how G2 associates g2-window items with visible windows.

3

Knowledge Bases

Shows how to work with the current KB, save the current KB, and load a KB.

Introduction

A **knowledge base**, or **KB**, is the container in which you collect and organize a set of knowledge about real or virtual entities.

A KB contains knowledge in the form of items which have attributes. Attributes can also be items as well as simple or component values. The items in a KB represent a set of application knowledge. G2 Items describes the purposes and features of items.

A running G2 process provides an interactive environment that a developer uses to work with knowledge bases. The Developer's Environment describes the features provided by the G2 developer's environment.

You use G2 to work with a KB as follows:

- A running G2 process reserves part of its memory to contain one KB, called the **current KB**. A G2 process always works with one and only one current KB.
- You can direct G2 to save the KB into one or more files called **KB files.**

A KB can contain *executable* items, which are items that specify actions that G2 performs on the information contained in the KB.

You use the G2 developer's environment to *operate* a KB. To operate a KB means to start, pause, resume, reset, and restart the KB. You can operate your KB programmatically, for example, within rules and procedures.

You work with the current KB and with KB files in these ways:

- You can *load* a previously saved KB file, so that it replaces the contents of the current KB.
- You can *merge* the contents of a saved KB file into the current KB.
- You can *clear* the current KB by removing all application knowledge from it.
- You can *save* the current KB to KB files that G2 stores on a storage device.
- You can *commit* and *update* source-code controlled files in the current KB.
- You can *capture* the current KB, along with all its run-time information, to a snapshot file.
- You can *warmboot* a KB from a KB snapshot file to run a captured KB as if resumed at the time its snapshot file was written.

There are other ways to work with a modularized KB, such as saving the top-level module to a file. Modular KBs are described in Modularized KBs.

Finally, each KB module has a set of system tables. These tables store preferences that affect how G2 uses the KB's application knowledge when the KB is loaded.

Contents of a KB

A KB contains knowledge in the form of items which have attributes. A KB itself is *not* an item.

Items

A KB can contain any number of items, subject to the memory and disk storage limitations of your computer.

The items in a KB can represent both permanent and transient knowledge. For a description of how items represent knowledge, see Understanding the Knowledge Contained in Items.

System Tables

Each KB contains at least one set of system tables. System tables contain information that determines the default behavior of a G2 process. You can specify new values for many system table parameters, or you can use the default values. For a description of each system table and their attributes, see System Tables.

The system tables that are currently active for a KB are called the **installed system tables**. The behavior of G2's run-time environment is largely defined by the set of installed system tables.

A modularized KB can contain more than one set of system tables when the KB consists of more than one module. Each module has its own set of system tables.

- If you save a modularized KB into separate KB files, each KB file contains its own set of system tables.
- If you save an inconsistently modularized KB into a single file, or if you intentionally save a modularized KB into a single file, the KB file contains one set of system tables for each module in the KB.

For more information on modularized KBs, see Working with Modularized KBs.

Operating the Current KB

A running G2 process always contains a KB, called the **current KB**, in its memory. You operate the current KB by starting, pausing, resuming, resetting, or restarting it. You can operate the current KB by using the G2 menus or programmatically.

The Initial Contents of a KB

By default, a new G2 process contains the following system-defined items:

- A g2-window item associated with the G2 process.

- A g2-window item for each connected Telewindows process, if any.
- A ui-client-session item for each connected G2 JavaLink process, if any.
- A complete set of system tables.

When a new G2 process starts, G2 initializes all the attributes of the system-table and window items to their default values. You can customize the attributes of these items, and you can add knowledge to the KB by interactively creating items. When your KB contains executable items, you can programmatically add and change KB knowledge.

You can save your KB at any point. When you next load your KB, G2 restores your customized settings and added KB knowledge.

Clearing the Current KB

Sometimes you may wish to clear the knowledge you have added to the current KB, and begin KB development again. The clear action reverts the contents of the current KB back to its initial set of system-table and window items. The attribute values of these items regain their default values with the exception of the Server Parameters system table which retains its non-default values because it is associated with the G2 process and not with a particular KB.

Caution Clearing the current KB cannot be undone. To save the knowledge in the current KB before clearing it, you must save it to one or more KB files.

To clear the current KB:

➔ Select Main Menu > Miscellany > Clear KB.

A clear-KB action is implicit when you specify a KB load because G2 automatically clears the current KB before loading the new one. When you merge a KB, the current KB is not cleared.

Starting the Current KB

Starting the current KB initializes all executable items so they can run. You can start the current KB after launching a new G2 process or after resetting the KB. After you start the current KB, it continues to run until you pause, reset, or restart it.

To start the current KB:

➔ Select Main Menu > Start.

Pausing and Resuming the Current KB

Pausing a KB means temporarily suspending the execution of all items. You can pause the current KB only if it is already running.

You can still create items and interact with them in most ways when a KB is paused. However, certain interactions are restricted when a KB is paused.

Once you have paused a KB, you can resume running it to continue execution. You can pause and resume a KB interactively or programmatically.

To pause the current KB interactively:

➔ Select Main Menu > Pause.

To pause the current KB programmatically:

➔ Execute the pause knowledge-base action.

To resume a paused KB:

➔ Select Main Menu > Resume.

To resume the current KB programmatically:

➔ Execute the resume knowledge-base action.

Resetting the Current KB

Resetting the current KB:

- Stops the knowledge base from running.
- Reinitializes all variable and parameter values.
- Returns all items to their initial positions.
- Restores the default colors of all items.
- Deletes any transient items.
- Removes any relation instances that you have established, unless the relations are permanent.

You can reset the current KB at any time.

To reset the current KB interactively:

➔ Select Main Menu > Reset.

To reset the current KB programmatically:

➔ Execute the reset knowledge-base action.

Restarting the Current KB

Restarting the current KB starts the KB again as if it had been reset. Restarting is the same as selecting Main Menu > Reset, then selecting Main Menu > Start, in succession.

To restart the current KB:

➔ Select Main Menu > Restart.

Determining the Run-State of the Current KB

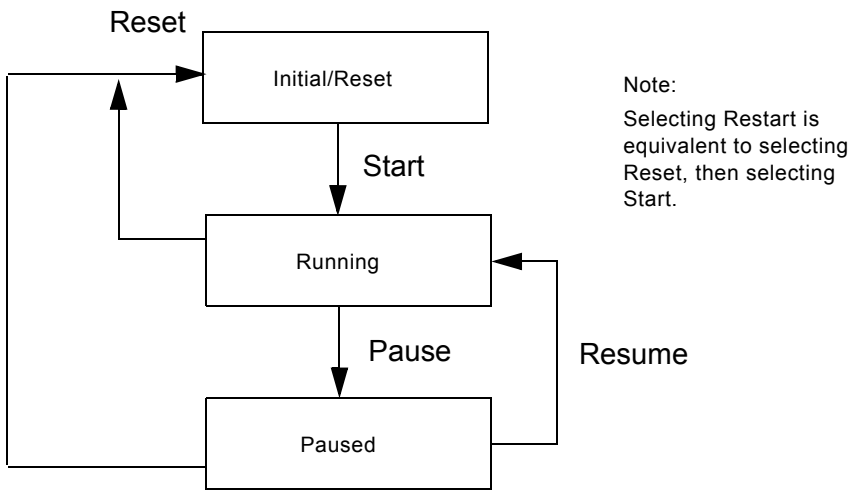
Because G2 executes, or *runs*, the current KB, we refer to the current KB's **run-state**. Run-states affect the contents of a KB.

The next table summarizes how menu choices on the Main Menu affect the current KB run-state:

Main Menu Choice	Purpose	Resulting Run-State
Start	Start executing the current KB after it is loaded or reset.	Running
Pause	Stop executing the current KB, but allow execution to be resumed.	Paused
Resume	Continue executing a paused current KB.	Running
Reset	Initialize all information in the current KB.	Initial/ Reset
Restart	Reset and start executing a current KB that is already running or paused.	Running

G2 displays only the Main Menu choices that are valid for the current run-state. For instance, if the current KB is paused, the Main Menu displays the Restart, Resume, and Reset choices, but not the Start and Pause choices.

This diagram shows the Main Menu choices that transition between run-states.



By default, G2 does not confirm run-state changes. Set the confirm-run-state-changes attribute in the Miscellaneous Parameters system table to yes to post a confirmation dialog for any attempt to start, restart, reset, resume, or pause G2.

The Initial/Reset Run-State

In the initial/reset run-state, a KB is ready for running. In this run-state, you can interactively change all knowledge in the KB.

In this run-state, a KB contains only permanent knowledge. For a description of permanent items and their initialized state, see G2 Items.

From the initial/reset run-state, a KB can transition only to the running run-state.

The Running Run-State

In the running run-state, G2 is performing the tasks specified in the current KB. G2 detects events that occur in real time, performs actions specified in rules, executes procedures, seeks data for variables, and so on. G2 performs these tasks as a series of operations dispatched and controlled by the scheduler, as described in Task Scheduling.

When the KB is running, it can contain both updated permanent knowledge and transient knowledge. Transient knowledge consists of a KB's transient items and the transient information associated with permanent items. For a description of transient items and the transient information associated with permanent items, see G2 Items.

From the running run-state, a KB can transition to the initial/reset run-state or to the paused run-state.

The Paused Run-State

In the paused run-state, G2 suspends the execution of all tasks specified by the KB. G2 retains all information about updated permanent and transient knowledge. When in this state, you can resume running, at which time G2 continues performing suspended tasks.

From the paused run-state, a KB can transition to the initial/reset run-state or to the running run-state.

Summary of Run-States

For each part of a KB's knowledge, the following table summarizes its condition under each run-state:

Status of Knowledge During Each Run-State

Item Knowledge	Initial/Reset	Running	Paused
Attributes of items	Initial: Contain default values. Reset: contain the most recently assigned values.	Contain most recently assigned values.	Contain most recently assigned values.
Variables	Values revert to initial values. Do not have collection times, expiration times, simulation values, histories, or simulation histories (as applicable).	Conclude initial values when activated. After activation, have current values, collection times, etc. (as applicable).	Have current values, collection times, etc. (as applicable).
Parameters	Values revert to initial values. Do not have histories.	After activation, have current values.	Have current values.
Arrays and lists	Can have contents if permanent.	Can have contents.	Can have contents.
Relation instances	Can exist if permanent.	Can exist.	Can exist.
Transient items	Do not exist.	Can exist.	Can exist.

Status of Knowledge During Each Run-State

Item Knowledge	Initial/Reset	Running	Paused
Rules, procedures, formulas, and functions	Do not execute, and cannot resume previous execution.	Execute.	Do not execute, but can be resumed.
Definitions	Can be edited.	Can be edited.	Can be edited.
User-interface items, such as, buttons	Show a menu when clicked.	Operational.	Show a menu when clicked.
Internal and external data service and polling	Neither data service nor polling takes place; external data service connection closed; new external data service connection not allowed.	All data service and polling takes place; new external data service connection allowed.	Data service does not take place; polling paused; external data service paused; new external data service connection allowed.
Item registration status for external data service	No new registrations; existing registrations removed.	New registrations allowed; existing registrations retained.	New registrations allowed; existing registrations retained.
Internal and external message service	Neither takes place.	Both take place.	Message input; message output paused.

The G2 Simulator, which can provide simulation values and simulation histories, is a superseded capability. For more information, see Appendix F, Superseded Practices.

Saving Your KB Knowledge

When you add knowledge to a KB, you should save it periodically to a knowledge base consisting of one or more files. Each module is saved in its own **KB file.** By default, the KB data that is saved in your knowledge base does not include transient knowledge; instead, it is the permanent knowledge that persists after a reset action has deleted runtime transient knowledge.

The mode most frequently used for saving KB knowledge is saving permanent data to modular KB files. G2 also supplies a system procedure you can use to save both the transient and permanent data in your running KB to a single file called a **KB snapshot file**. Permanent and transient knowledge is described in Distinguishing Permanent, Transient, and Current Knowledge. Information on saving a KB snapshot file is given in Saving Permanent and Transient Data in Snapshot KBs.

G2 saves your KB modules in a compressed format consisting of ASCII characters. KB files are fully portable across all G2-supported platforms.

The capability to save a KB depends upon the license associated with your G2 product. For information about G2 licenses, see Licensing and Authorization.

Saving the Current KB

You can save the current KB interactively or programmatically. You can save it whether it is running, paused, or reset.

To save the current KB programmatically:

➔ Use the saving system procedures described in KB and Module Operations in the *G2 System Procedures Reference Manual.*

To save the current KB interactively:

➔ Select Main Menu > Save KB.

The Save dialog that appears differs depending on whether your current KB is modularized or unmodualized. To be minimally modularized, a KB must have one named module, and all top-level workspaces must be assigned to that module.

You name a module by editing the top-level-module attribute of the Module Information system table, and you assign top-level workspaces by editing the module-assignment attribute of kb-workspaces. To organize your KB into separate modules, see Creating a Module Hierarchy.

Saving a Modularized KB

If the current KB was loaded from saved KB files, each module's Saving Parameters system table includes the current-file-for-module attribute, which tells you the file path from which the module was loaded.

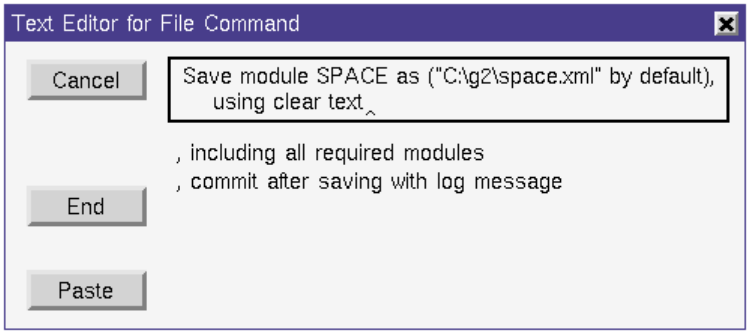
You save a modularized KB in the Save dialog.

To save a modularized KB:

➔ save module *module-name* as (*default-quoted-file-path* by default)
[{*overriding-file-name-symbol* | *overriding-quoted-file-path*}]
[, including all required modules]
[, using clear text]

The syntax in the first line is required; the other three lines contain optional phrases. When using clear text, the saved KB is in XML format using the `.xml` extension.

The following example Save dialogs are based on saving a newly developed modularized KB. The top-level module is named space, and space has a single required module called definitions. By default, the save dialog for an unsaved KB comes up with syntax to save the top-level module of your current KB to a file path that is either your home directory or the directory from which you launched G2. For example, the following unedited dialog saves the top-level module space to a default file path:

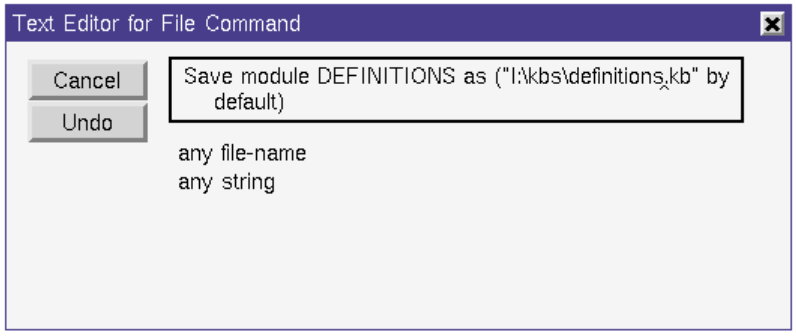


Tip Place KB modules you do not wish to have overwritten in read-only directories, and set your module search path to include all of the pathnames of your KB directories.

To save a required module instead of the top-level module:

➔ Edit the *module-name* and *default-quoted-file-path* in the first line of the edit box to another module name and another file path.

This example saves the definitions module to another file path:

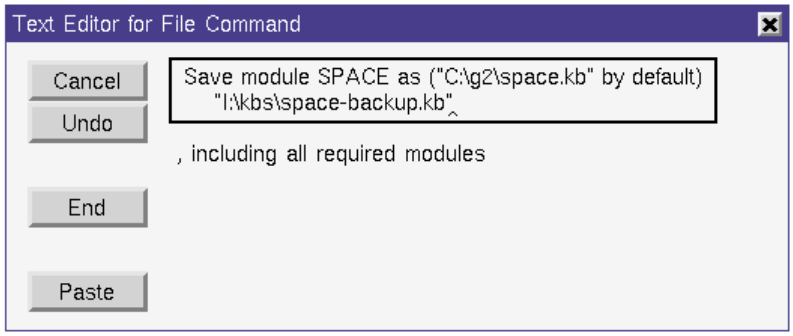


To specify alternative file paths using overriding grammar:

1 In the save dialog, place your cursor after the closing parenthesis.

2 Type in an alternative file name or an alternative quoted file path.

For example:



Although G2 allows you to enter any file-name extension, it actually saves your modules using the `.kb` extension (or `.xml` if clear text is used).

To additionally save all required modules:

➔ Select the including all required modules phrase.

Saving an Unmodularized KB

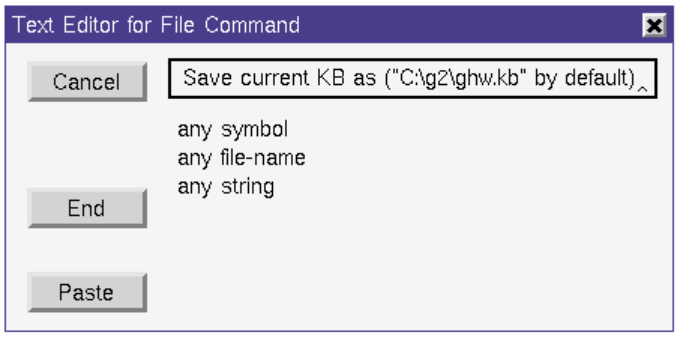
You save a modularized KB in the Save dialog.

To save an unmodularized KB:

➔ save current KB as (*quoted-file-path* by default)
[{*overriding-file-name-symbol* | *overriding-quoted-file-path*}]

You can accept the *default-quoted-file-path* in the edit box or edit it to another file path. Alternatively you can supply an overriding file-name symbol or quoted file path by typing it after the right parenthesis in the edit box.

For example:



Backup Copies of KB Files

When G2 writes a KB module to a filename that already exists in the same directory, it first appends a tilde (~) to the existing file before saving the current KB.

For example, suppose you save a module to a file named `classes.kb`. If you later save a module to that same filename and directory, G2 changes the original file to `classes.kb~` and saves the current module to `classes.kb`. G2 saves only one backup copy.

Platform File Systems and KB File Names

A KB's filename must be acceptable to the file system that stores the KB, and to the G2 dialogs that load, save, and merge KBs. To insure that KB filenames work under all conditions, they should:

- Contain only the characters A-Z, a-z, 0-9, dot (.) and underscore (_).
- Have at most eight characters in the filename proper, followed by a dot and a suffix of at most three characters.
- Use the dot character only to indicate a suffix.

Filenames that conform to the described syntax work anywhere. Depending on the uses of your KB, you may be able to relax these restrictions, but you should do so only if you are certain that no incompatibility with an unanticipated use can occur. For example, hyphens (-) can appear in a KB filename if the KB will never be stored on CD-ROM, where the ISO 9660 standard precludes them; and blank spaces in file and directory names are supported by the NTFS and FAT32 file systems on Windows platforms, but the parsing methods on Unix platforms discourage their use.

Using Comments

You can add comments to a KB by editing an attribute of the Saving Parameters system table. See Adding Comments to a KB.

Using Change Logging for Version Control

G2 provides a comprehensive version control system, which leverages the G2 change log facility, to allow:

- Tagging attributes of G2 objects that support change logging (for example, procedures, rules, class definitions) within a module, as well as tagging all the attributes that support change logging of all items in a module.
- Reverting change-loggable attributes of individual items or of all items in a module to a previous revision. Note that this only works on items that still exist; G2 does not preserve the change log of deleted items.
- Deleting change log entries.
- Commenting change log entries.

For information on...	See...
Enabling change logging for a KB	Using KB Change Logging.
System procedures you can use for version control	Version Control in the *G2 System Procedures Reference Manual*.
Using the Inspect facility for version control	Version Control.

The following examples refer to the following item named my-umc, whose change log shows edits to the names, label, action, and applicable-class attributes of the item, including three revisions of the label attribute:

Temporary Workspace

Attribute	Revision	Value	Module Version	Timestamp	Author	Tags
Label	2	post-hello-world	63	7 Feb 2007 3:09 p.m.	nrs	none
Applicable class	0	object	63	7 Feb 2007 3:09 p.m.	nrs	none
Action	0	post "Hello world"	63	7 Feb 2007 3:08 p.m.	nrs	none
Label	1	do-action	63	7 Feb 2007 3:08 p.m.	nrs	none
Label	0	do-something	63	7 Feb 2007 3:08 p.m.	nrs	none
Names	0	MY-UMC	63	7 Feb 2007 3:07 p.m.	nrs	none

Tagging Change Log Entries

Here is a generic procedure that tags the change log entry of an item with a given identifier:

```
tag-change-log-entry (item: class item, attribute-name: symbol, identifier: structure,
new-tag: symbol)
resulting-struct: structure;
begin
    resulting-struct = call g2-tag-change-log-entry(item, attribute-name, identifier,
        new-tag);
    post "[resulting-struct]"
end
```

This action button tags the change log entry for the names attribute of my-umc with the given timestamp with the symbol G283R0:

Here is the resulting change log and message board:

Temporary Workspace

Attribute	Revision	Value	Module Version	Timestamp	Author	Tags
Label	2	post-hello-world	63	7 Feb 2007 3:09 p.m.	nrs	none
Applicable class	0	object	63	7 Feb 2007 3:09 p.m.	nrs	none
Action	0	post "Hello world"	63	7 Feb 2007 3:08 p.m.	nrs	none
Label	1	do-action	63	7 Feb 2007 3:08 p.m.	nrs	none
Label	0	do-something	63	7 Feb 2007 3:08 p.m.	nrs	none
Names	0	MY-UMC	63	7 Feb 2007 3:07 p.m.	nrs	G283R0

```
#86   3:15:51 p.m.   structure
  (ATTRIBUTE: the symbol NAMES,
  REVISION: 0,
  COMMENT: "",
  TAGS: sequence (the symbol G283R0),
  TEXT-VALUE: "MY-UMC",
  MODULE-VERSION: 63,
  TIMESTAMP: structure (YEAR: 2007,
    MONTH: 2,
    DATE: 7,
    HOURS: 15,
    MINUTES: 7),
  AUTHOR: the symbol NRS)
```

Getting Change Log Entries

Here is a generic procedure that posts the change log entry for an attribute of an item with a given identifier:

```
post-change-log-entry (item: class item, attribute-name: symbol, tag: structure)
result: structure;
begin
 result = call g2-get-change-log-entry (item, attribute-name, tag);
 post "[result]"
end
```

This action button gets the change log entry for the names attribute of my-umc tagged with the symbol G283R0:

Find By Tag

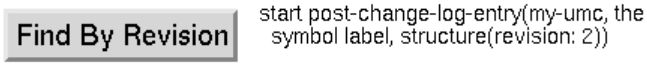
start post-change-log-entry(my-umc, the symbol names, structure(tag: the symbol G283R0))

POST-CHANGE-LOG-ENTRY

Here is the resulting message board:

```
#86   3:15:51 p.m.   structure
  (ATTRIBUTE: the symbol NAMES,
  REVISION: 0,
  COMMENT: "",
  TAGS: sequence (the symbol G283R0),
  TEXT-VALUE: "MY-UMC",
  MODULE-VERSION: 63,
  TIMESTAMP: structure (YEAR: 2007,
    MONTH: 2,
    DATE: 7,
    HOURS: 15,
    MINUTES: 7),
  AUTHOR: the symbol NRS)
```

This action button gets the change log entry for revision 2 of the label attribute of my-umc:



Here is the resulting message board:

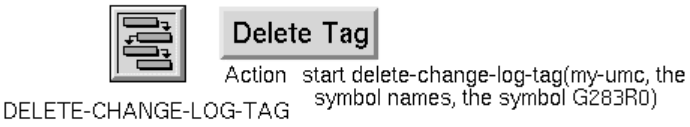
```
#89   3:21:00 p.m.   structure
  (ATTRIBUTE: the symbol LABEL,
  REVISION: 2,
  COMMENT: "",
  TAGS: sequence (),
  TEXT-VALUE: "post-hello-world",
  MODULE-VERSION: 63,
  TIMESTAMP: structure (YEAR: 2007,
    MONTH: 2,
    DATE: 7,
    HOURS: 15,
    MINUTES: 9),
  AUTHOR: the symbol NRS)
```

Deleting Change Log Entry Tags

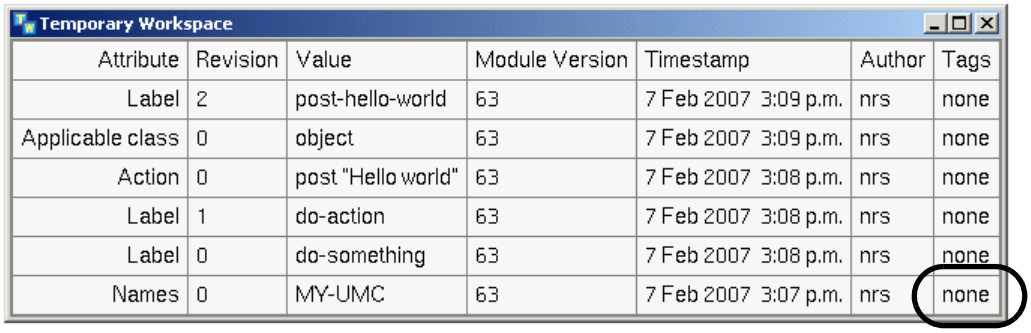
Here is a generic procedure that deletes the change log tag for an attribute of an item:

```
delete-change-log-tag (item: class item, attribute-name: symbol, new-tag: symbol)
resulting-struct: structure;
begin
    resulting-struct = call g2-delete-change-log-tag(item, attribute-name, new-tag);
    post "[resulting-struct]"
end
```

This action button deletes the change log entry tag G283R0 for the names attribute of my-umc:



Here is the resulting change log and message board:



Attribute	Revision	Value	Module Version	Timestamp	Author	Tags
Label	2	post-hello-world	63	7 Feb 2007 3:09 p.m.	nrs	none
Applicable class	0	object	63	7 Feb 2007 3:09 p.m.	nrs	none
Action	0	post "Hello world"	63	7 Feb 2007 3:08 p.m.	nrs	none
Label	1	do-action	63	7 Feb 2007 3:08 p.m.	nrs	none
Label	0	do-something	63	7 Feb 2007 3:08 p.m.	nrs	none
Names	0	MY-UMC	63	7 Feb 2007 3:07 p.m.	nrs	none

```
#100   4:09:18 p.m.   structure
  (ATTRIBUTE: the symbol NAMES,
  REVISION: 0,
  COMMENT: "",
  TAGS: sequence (),
  TEXT-VALUE: "MY-UMC",
  MODULE-VERSION: 63,
  TIMESTAMP: structure (YEAR: 2007,
    MONTH: 2,
    DATE: 7,
    HOURS: 15,
    MINUTES: 7),
  AUTHOR: the symbol NRS)
```

Deleting Change Log Entries

Here is a generic procedure that deletes a change log entry for an attribute of an item with a given identifier:

```
delete-change-log-entry (item: class item, attribute-name: symbol, identifier: structure)
succeeded: truth-value;
begin
    succeeded = call g2-delete-change-log-entry(item, attribute-name, identifier);
    if succeeded then
        post "deleting entry succeeded!"
    else
        post "deleting entry failed!"
end
```

This action button deletes revision 2 of the change log entry for the label attribute of my-umc:

Delete Change Log Entry

start delete-change-log-entry(my-umc, the symbol label, structure(revision: 2))

DELETE-CHANGE-LOG-ENTRY

Here is the resulting change log:

Temporary Workspace

Attribute	Revision	Value	Module Version	Timestamp	Author	Tags
Applicable class	0	object	63	7 Feb 2007 3:09 p.m.	nrs	none
Action	0	post "Hello world"	63	7 Feb 2007 3:08 p.m.	nrs	none
Label	1	do-action	63	7 Feb 2007 3:08 p.m.	nrs	none
Label	0	do-something	63	7 Feb 2007 3:08 p.m.	nrs	none
Names	0	MY-UMC	63	7 Feb 2007 3:07 p.m.	nrs	none

Commenting Change Log Entries

This action button adds a comment to revision 1 of the label attribute change log entry for my-umc:

Set Change Log Comment

start g2-set-change-log-entry-comment(my-umc, the symbol label, structure(revision: 1), "Changed label to do-action")

Here is a generic procedure that adds a comment to a change log entry for an attribute of an item with a given identifier:

```
post-change-log-entry-comment(item: class item, attribute-name: symbol,
    id: structure)
comment: text;
begin
    comment = call g2-get-change-log-entry-comment(item, attribute-name, id);
    post "[comment]"
end
```

This action button adds a comment to revision 1 of the change log entry for the label attribute of my-umc:

Post Change Log Comment

start post-change-log-entry-comment(my-umc, the symbol label, structure(revision: 1))

POST-CHANGE-LOG-ENTRY-COMMENT

Here is the resulting message board:

```
#96   3:28:12 p.m.   Changed label to do-
  action
```

Reverting Change Log Entries

Here is a generic procedure that reverts the change log entry for an attribute of an item with a given identifier:

```
revert-change-log-entry (item: class item, attribute-name: symbol, identifier: structure)
resulting-struct: structure;
begin
    resulting-struct = call g2-revert-change-log-entry(item, attribute-name, identifier);
    post "[resulting-struct]"
end
```

This action button reverts the label attribute of my-umc to revision 0:

Revert Attribute

start revert-change-log-entry(my-umc, the symbol label, structure(revision: 0))

REVERT-CHANGE-LOG-ENTRY

Here is the resulting change log and message board, thereby adding a new entry to the change log:

Temporary Workspace

Attribute	Revision	Value	Module Version	Timestamp	Author	Tags
Label	3	do-something	63	7 Feb 2007 3:34 p.m.	nrs	none
Applicable class	0	object	63	7 Feb 2007 3:09 p.m.	nrs	none
Action	0	post "Hello world"	63	7 Feb 2007 3:08 p.m.	nrs	none
Label	1	do-action	63	7 Feb 2007 3:08 p.m.	nrs	none
Label	0	do-something	63	7 Feb 2007 3:08 p.m.	nrs	none
Names	0	MY-UMC	63	7 Feb 2007 3:07 p.m.	nrs	none

```
#99   3:34:26 p.m.   structure
  (ATTRIBUTE: the symbol LABEL,
  REVISION: 0,
  COMMENT: "",
  TAGS: sequence (),
  TEXT-VALUE: "do-something",
  MODULE-VERSION: 63,
  TIMESTAMP: structure (YEAR: 2007,
    MONTH: 2,
    DATE: 7,
    HOURS: 15,
    MINUTES: 8),
  AUTHOR: the symbol NRS)
```

Tagging All Items in a Module

This action button tags the current version of all attributes of all items in the module named top with the symbol G283B0:

Tag Whole Module

```
start g2-tag-module(the symbol top,
  structure(), the symbol G283R0)
```

Here is the resulting change log for my-umc:

Temporary Workspace

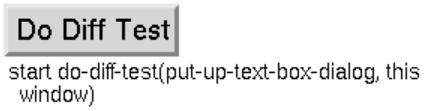
Attribute	Revision	Value	Module Version	Timestamp	Author	Tags
Label	3	do-something	63	7 Feb 2007 3:34 p.m.	nrs	G283R0
Applicable class	0	object	63	7 Feb 2007 3:09 p.m.	nrs	G283R0
Action	0	post "Hello world"	63	7 Feb 2007 3:08 p.m.	nrs	G283R0
Label	1	do-action	63	7 Feb 2007 3:08 p.m.	nrs	none
Label	0	do-something	63	7 Feb 2007 3:08 p.m.	nrs	none
Names	0	MY-UMC	63	7 Feb 2007 3:07 p.m.	nrs	G283R0

Performing “Diff” Operations

You can perform a “diff” operation on two texts or two change log entries.

For information on the system procedures you can use for text “diff” operations, see Version Control.

For example, this action button performs a “diff” on the change log for put-up-text-box-dialog:



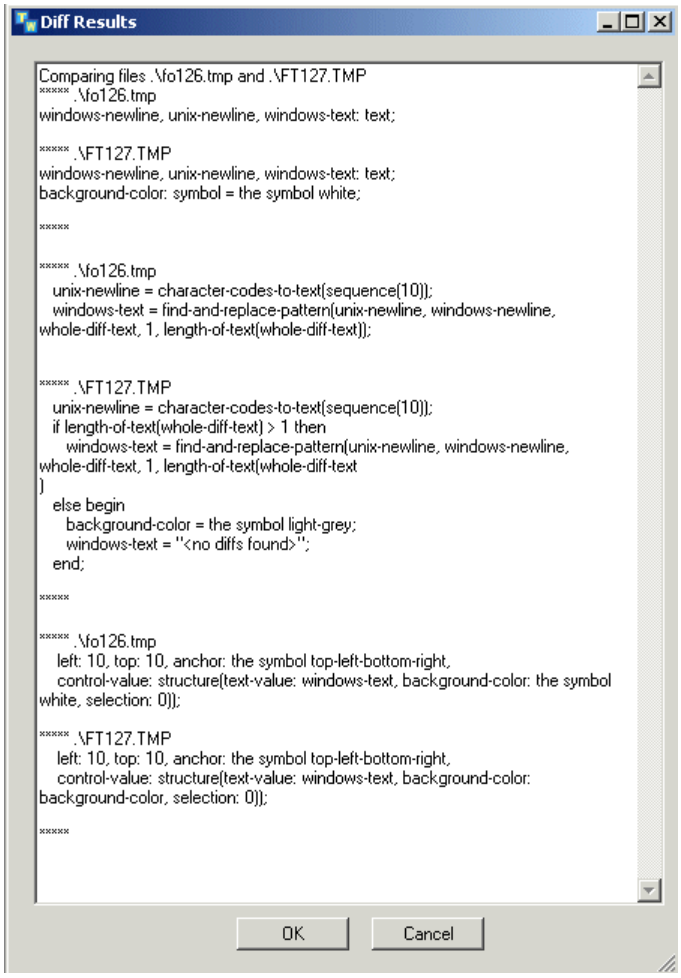
This procedure calls spawn-diff, which performs a “diff”, and show-results, which displays the results in a text box within a custom dialog:

```
do-diff-test (item-to-diff: class item, g2-win: class g2-window)
whole-diff-text: text;
begin
    whole-diff-text = call spawn-diff(item-to-diff);
    call show-results(whole-diff-text, g2-win)
end
```

This procedure calls g2-diff-texts on revision 0 and revision 1 of an item, and returns the diff-output of the return structure:

```
spawn-diff (item-to-diff: class item) = (text)
all-diffs: value;
user-name: value;
ndiffs: integer;
result: structure;
v0, v1, whole-diff-text: text;
begin
    all-diffs = the change-log of item-to-diff;
    ndiffs = the number of elements in all-diffs;

    user-name = call g2-name-for-item(item-to-diff);

    if ndiffs = 0 then
        post "[user-name] has no change-log"
    else if ndiffs = 1 then
        post "[user-name] has only one revision; cannot diff"
    else
        post "[user-name] has [ndiffs] revisions total";
    v0 = the text-value of all-diffs[0];
    v1 = the text-value of all-diffs[1];
    result = call g2-diff-texts(v1, v0);
    whole-diff-text = the diff-output of result;
    return whole-diff-text
end
```

Here is the result of doing the "diff" test on put-up-text-box-dialog:



Saving a Running Current KB

If you save a KB while it is running, G2 saves the permanent knowledge in the KB as of that moment in time, regardless of any changes made to the knowledge thereafter. The G2 scheduler allows KB processing to take place normally; G2 tasks of a higher priority take place before G2 tasks of lower priorities.

In addition, when G2 starts to save the KB, it delays any KB processing that changes any part of the permanent knowledge, such as deleting a workspace, until the KB is completely and successfully saved. It also postpones all other processing of lower priority than the delayed processing until the save is complete. In this way, G2 preserves consistency in the current KB.

The G2 scheduler manages the task of saving a KB while it is running. The scheduler sets the priority of tasks based on the value of the default-priority-for-runtime-saving attribute in the Saving Parameters system table.

Note The default value of the default-priority-for-runtime-saving attribute is priority 8, which causes saving while running to execute as a relatively low priority task, known as a background task.

For example, when you save a KB while it is running, G2 processes rules normally, because the default priority for processing rules is priority 1.

Using System Procedures that Pause G2 before Saving Your KB

There are three system procedures in `sys-mod.kb` that save your KB by first pausing G2, saving your KB, and then resuming G2. Refer to KB and Module Operations in the *G2 System Procedures Reference Manual*.

Saving the State of Workspaces

A KB file or KB snapshot file stores the following information about your KB workspaces:

- The scale and absolute position within the G2 window.
- Which workspaces are visible.
- The back-to-front ordering of the visible workspaces.

Tip For more information about how G2 manages the appearance of workspaces in windows, see the Positioning a Workspace within its Window.

Supporting Source-Code Control Systems

When you save a KB to the same KB file from which it was loaded, G2 updates only a portion of the KB file itself. This allows an industry-standard source-code control system (SCC) to detect which characters in the updated file represent the most recent changes.

Note When checking out a KB file using a SCCS, do *not* use keyword expansion; otherwise, the KB file will be corrupted. For example, if you use the RCS application, specify the `-ko` argument when checking out a KB file.

Loading a KB

Loading a knowledge base means *replacing* the current KB with knowledge read from any KB files.

Note When you load a KB, G2 replaces the entire current KB with the new KB.

G2 loads a KB by reading from saved KB files. Saving KBs is described in Saving Your KB Knowledge and Saving Permanent and Transient Data in Snapshot KBs.

The procedure for loading a KB file and a KB snapshot file are essentially the same, except for the options that you might want to specify, as outlined in Selecting Options when Loading a KB File.

Also, after loading a KB snapshot file, you can warmboot your KB. For information on warmbooting, see Warmbooting a KB Snapshot File.

To load a KB file interactively:

1 Pause or reset the KB by selecting Pause or Reset from the Main Menu.

2 Select Main Menu > Load KB.

G2 displays the Load KB dialog, described in the next section.

To load a KB file programmatically:

➔ Use the KB-loading system procedures described in KB and Module Operations in the *G2 System Procedures Reference Manual.*

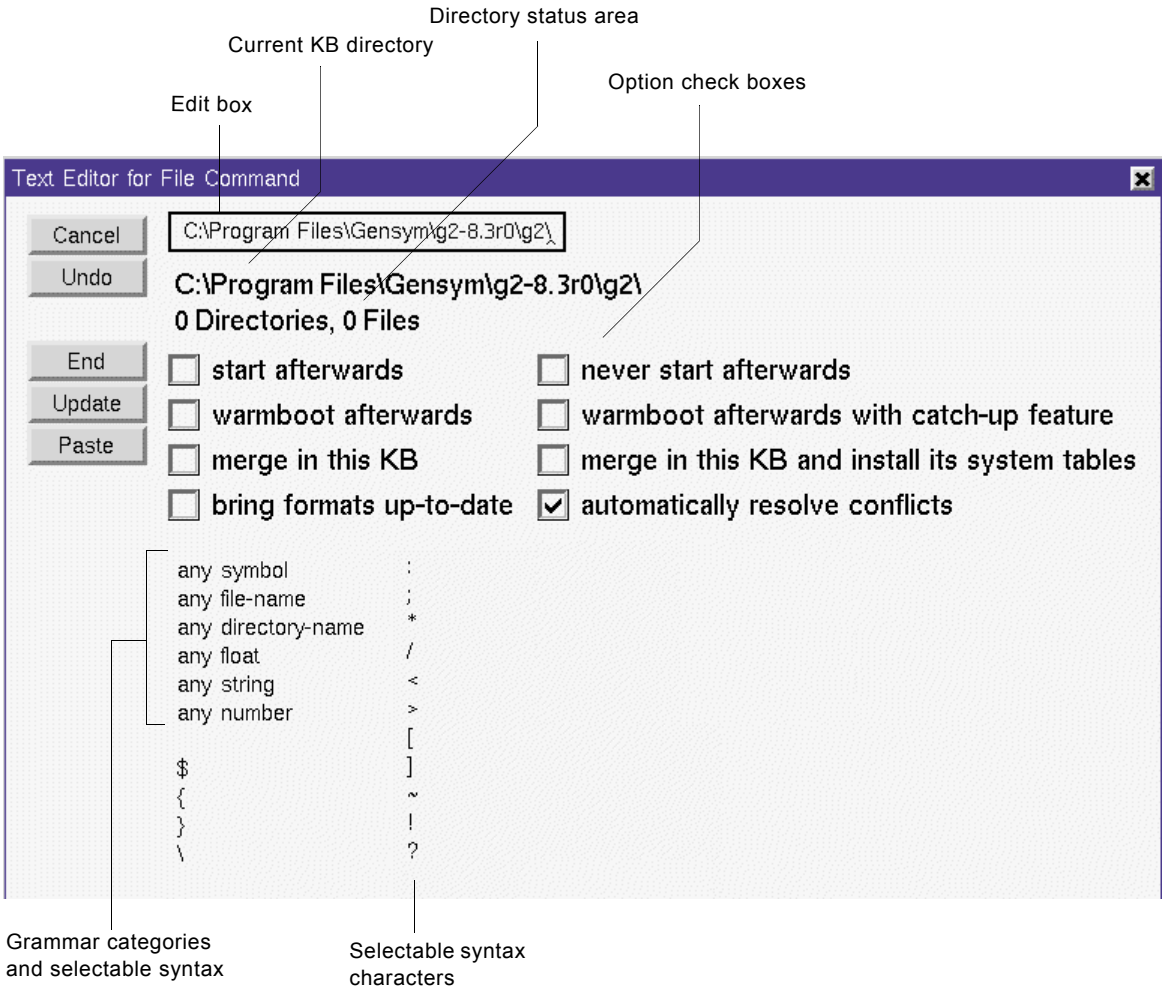
As G2 loads a KB file, it displays a table that informs you of its progress. The display also shows this progress as a percentage. It does not indicate the real size of the KB being loaded.

Note You can load KB files that you saved using previous versions of G2. However, you *cannot always* load KB files that you save using a later version of G2 into a previous version. See the *G2 Bundle Release Notes* for version-specific backward-compatibility details.

You can also load a KB file by using an initialization file. For details see Using an Initialization File.

Using the Load KB Dialog

When you load a KB file interactively, G2 displays the Load KB dialog, as shown in the next figure:



The first time you use the Load KB dialog, G2 displays in the edit area the directory from which you launched the G2 process. Thereafter, the default directory is the directory pathname most recently specified in a successfully executed Load KB, Merge KB, or Save KB operation.

By default, the automatically resolve conflicts option is selected so that intermodual class-definition differences are automatically resolved by G2. Resolving such conflicts by hand is not recommended because it is very time consuming; however, the option is deselectable.

In this dialog, you can navigate to any directory where KB files are stored. You can enter the name of the KB file to load, or select it from the list of files that appears at the bottom of the dialog.

To display the contents of a directory:

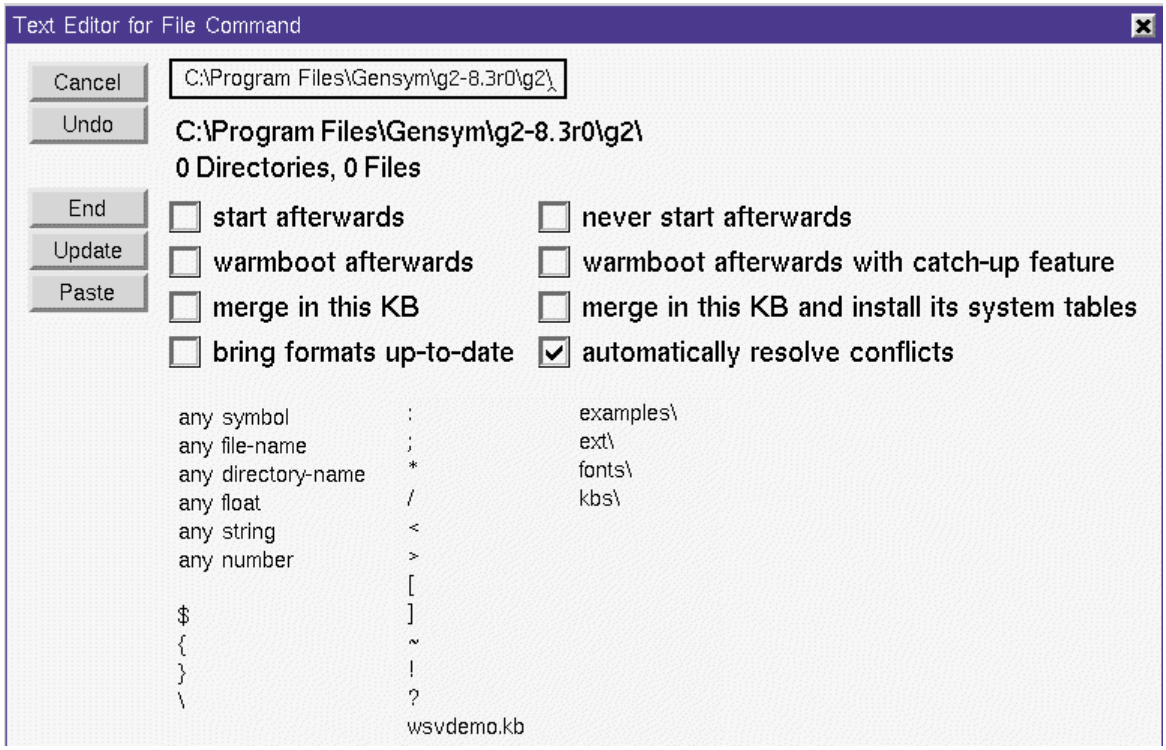
➔ Enter a pathname in the edit area, including a trailing delimiter character, and click End.

or

➔ Enter a pathname in the edit area, without a trailing delimiter character, and press Return.

The trailing delimiter character depends on your platform: / on UNIX platforms and \ on Windows platforms.

G2 displays a list of subdirectories and KB filenames contained in the specified directory. This figure shows how the Load KB dialog displays these lists:



At this point, you can select a KB file to load or another subdirectory. If you select a subdirectory, continue following the above procedure until you find the desired KB file.

Note G2 cannot load an empty file or a file that is not a KB file. If you attempt this, G2 signals an error.

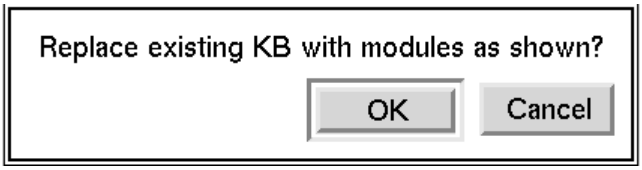
Loading the KB File

To load the specified KB file:

➔ Click End or press Return in the Load KB dialog.

If G2 contains a resident KB, it will be cleared before the new KB is loaded.

If the resident KB has unsaved permanent changes, G2 generates this confirmation dialog to notify you of the unsaved changes:



You can select the Cancel button and save the current KB before loading a new KB, or you can press the OK button to clear the resident KB and load a new KB. When G2 loads the new KB it reports on its load progress.

When loading is complete, G2 presents the contents of the loaded KB in the state in which it was saved. For information on what G2 saves in KB files and KB snapshot files, see Saving Your KB Knowledge.

In addition, G2 does the following:

- Sets the initial-value attributes of variables and the values of parameters.
- Displays some portion of each workspace that was visible when the KB was saved. G2 displays these workspaces in each G2 window that is connected to the G2 process. If the size of the G2 window is smaller than the size of the window at the time the KB was saved, G2 adjusts the absolute locations of the workspaces so that some portion of each workspace is visible.

After you load a KB file, the current KB contains the permanent knowledge that was stored in that file. For information on how items represent permanent knowledge, see Understanding the Knowledge Contained in Items.

Using Wildcards in Filenames when Loading a KB

You can enter a wildcard in the filename when loading a KB file. G2 displays a list of names that meet the specified criteria.

For instance, you can enter kb*s.kb to display a list of all KB files in the current directory, whose file names begin with the characters "kb" and end with the characters "s.kb".

To use wildcards in the filename, use combinations of the following characters:

Wildcard Character/ Purpose	Example
* (asterisk) Matches *zero or more* characters	Entering kb*s matches the files or directories named `kbfiles` and `kbs`.
? (question mark) Matches any one character	Entering kbfile? matches the files or directories named `kbfiles` and `kbfilez`.
{*abc*} (braces) Matches *one* occurrence of the character *a* or *b* or *c*, where *a*, *b*, and *c* each represents a character	Entering kb{ef}iles matches the files or directories named `kbfiles` and `kbeiles`.
{*abc*}* (braces and asterisk) Matches *zero or more* occurrences of the character *a* or *b* or *c*; where *a*, *b*, and *c* each represents a character	Entering kb{xyz}*files matches the files or directories named `kbfiles` and `kbzzzfiles`.
! (exclamation point) Escape (ESC) character allows use of other characters in the wildcard name	Entering kbfile!{s!} matches the file or directory named `kbfile{s}`.

You can also use these characters in the text of the argument passed to the g2-files-in-directory and g2-subdirectories-in-directory system procedures, as described in File Operations in the *G2 System Procedures Reference Manual*.

Selecting Options when Loading a KB File

You can modify how G2 loads the selected KB file by selecting one or more options on the Load KB dialog. To select a loading option, check its associated box.

Notice that the check boxes appear in pairs. For example, merge in this KB and merge in this KB and install its system tables pertain only to merging a KB file. You should not select both options in a pair at the same time.

This table explains each option on the Load KB dialog:

Load KB Option	Description
start afterwards	Begin running the new KB immediately after loading the KB file into a G2 that is in the initial/reset state. Selecting this option overrides the setting of the start-KB-after-load? attribute in the Miscellaneous Parameters system table of the loaded KB.
never start afterwards	Do not begin running the new KB after loading the KB file. Selecting this option overrides the setting of the start-KB-after-load? attribute in the Miscellaneous Parameters system table of the loaded KB.
warmboot afterwards	When loading a KB snapshot file, resumes running the KB from the point at which it was saved. This option has no effect if you are loading a normal KB file. For more information, see Warmbooting a KB Snapshot File.
warmboot afterwards with catch-up feature	When loading a KB snapshot file, sets the scheduler's internal current time to the current time saved in the snapshot file, and the scheduler-mode attribute of the Timing Parameters system to as fast as possible. This option has no effect if you are loading a normal KB file. For more information, see Warmbooting a KB Snapshot File.
merge in this KB	Merges the contents of the KB into the current KB. This is described in detail in Merging a KB File.
merge in this KB and install its system tables	Merges the contents of the KB into the current KB, and makes the merged module's system tables the installed system tables. This is described in Merging a KB File.

Load KB Option	Description
bring formats up-to-date	Applies the formatting defaults, which are specific to the current version of G2, to all loaded items. For example, the width of text items are based on system-defined defaults that might vary from version to version. **Note:** Selecting this option can significantly affect the appearance and layout of items when loading them into new G2 versions. In general, we do *not* recommended selecting this option unless you want to mix items developed under different G2 versions.
automatically resolve conflicts	When loading a modularized KB, G2 automatically checks for conflicts among class-definitions contained in the KB, and in any directly and indirectly required modules that G2 also loads. This option is selected by default. Using the automatically resolve conflicts feature is described in Detecting Conflicting Class-Definitions.

Searching for KB Files

When loading knowledge bases, G2 searches for module files in the specified directory and in the current G2 directory. The filename extension must be specified and it must be .kb or .kl. G2's module-saving scheme ensures that a directory has only one knowledge base with a particular base name and proper file extension. It does this by appending a tilde (~) to the backup copy. See Backup Copies of KB Files for more information. If you specify a file that is not a G2 knowledge base, G2 posts an error message to the Logbook.

Note The use of `.KL` and `.kl` files (known as knowledge libraries) is obsolete except for certain libraries supplied by Gensym to assist in localization, as described in Natural Language Facilities.

You can specify the home directory pathname for a G2 process using the `G2_HOME` environment variable. If no such specification exists, the home directory is the directory from which you launched G2.

Saving Permanent and Transient Data in Snapshot KBs

You can save and reload all of a KB's permanent and transient knowledge, including the real-time data associated with the G2 run-time environment, by

saving a **KB snapshot file** and reloading the snapshot KB with the warmboot option selected.

Saving a KB Snapshot File

G2 saves your snaphot KB knowledge in a single file.

To save a KB to a snapshot file:

➔ Execute the g2-snapshot or the g2-snapshot-without-other-processing system procedure, as described in KB and Module Operations in the *G2 System Procedures Reference Manual*.

Note G2 does not save a KB in which the attribute table of a transient item resides on a permanent workspace separate from the transient item.

The g2-snapshot system procedure writes the snapshot file with data as of the moment that it is invoked. It does this despite the fact that the task of writing the snapshot file allows interrupts for other processing. Thus, even if you modify or delete significant portions of a KB after invoking g2-snapshot, G2 writes that knowledge into the KB snapshot file as it existed at the time the procedure was invoked.

The g2-snapshot-without-other-processing saves your snapshot KB by first pausing G2, saving your KB, and then resuming G2.

See Warmbooting a KB Snapshot File for information about how to load KB snapshot file and resume running a KB from it.

Contents of a KB Snapshot File

A KB snapshot file records:

- All information necessary to present the KB as if it had been reset at the time of the snapshot, including information necessary to undo changes that are normally undone when a KB is reset.
- All transient items except transient g2-windows not on a workspace.
- The current values, collection times, expiration times, and histories of all variables and parameters when present.
- The simulation values and histories of all variables when present.
- The activation status of all KB workspaces.
- All instances of dynamic relations.
- The contents of all lists and arrays.

The G2 Simulator, which can provide simulation values and simulation histories, is a superseded capability. For more information, see Appendix F, Superseded Practices.

A KB snapshot file does *not* record:

- The current executing status of rules, button items and display items.
- Procedure invocations and their associated information.
- The Operator Logbook, the Message Board, menus, temporary workspaces, or tables, such as Inspect tables and attribute tables that have not been transferred to a workspace.

Naming Conventions for KB Snapshot Files

Snapshot filenames must include the `.kb` extension. In addition, you should follow naming conventions that distinguish snapshot files from KB files, such as including the suffix `-snapshot` in the filename. To identify the time at which the snapshot was saved, you might also prefer to include a timestamp in the filename, for example, `monitoring-snapshot-29may2000-02-14-23.kb`.

Warmbooting a KB Snapshot File

When loading a KB snapshot file, you must select one of the warmbooting afterwards options on the Load KB dialog to restore the snapshot file to its run-time state.

G2 loads and automatically resumes running the current KB as if it had merely been paused. This is called **warmbooting**. Loading a KB snapshot file restores both the stored KB's knowledge and the real-time data that existed at the time when the KB snapshot file was saved.

If you do not select the warmbooting afterwards check box when loading a KB snapshot file, G2 discards the run-time data that was loaded. The result is the same as if you had loaded a standard KB file.

To warmboot a KB snapshot file:

➔ Select the warmbooting afterwards option in the Load KB dialog.

To warmboot a KB snapshot file programmatically:

➔ Invoke the g2-warmboot-kb system procedure, as described in KB and Module Operations in the *G2 System Procedures Reference Manual*.

As when loading a KB file, when warmbooting a KB snapshot file, G2 sets the scheduler's *current time* to the *current real time*. When G2 resumes processing, G2 schedules its processing according to the value of the scheduler-mode attribute in the Timing Parameters system table of the loaded KB snapshot file.

After loading a KB snapshot file, G2 runs the current KB somewhat differently from its default behavior, as follows:

1. After the warmboot, G2 invokes no initially rules.
2. G2 looks for a procedure named warmboot, and executes it, if it exists.
3. After G2 has finished executing any warmboot procedure, it resumes executing all scanned rules.

Creating Warmboot Procedures

Warmbooting cannot automatically restore the context of procedures and rules that were executing when a KB snapshot file was written. To restore such context, you can provide one or more procedures called **warmboot procedures**. For example, you might want restart a procedure that was invoked just before the moment when the KB snapshot file was saved. You can accomplish this by using a warmboot procedure.

When the KB in a snapshot file contains a procedure whose name is warmboot, G2 invokes that procedure before beginning execution of the file. This invocation provides a hook that you can use to take whatever action is necessary to restore the needed context. If no procedure named warmboot exists, G2 continues without error. A modularized KB can also contain warmboot procedures that are not named warmboot, as described under Modular Warmboot Procedures.

Non-Modular Warmboot Procedures

When a KB is not modularized, or when one warmboot procedure suffices for all modules, you can provide a procedure named warmboot that does what is needed. This procedure must take no arguments and return no values.

Code the warmboot procedure so that it specifies a set of actions that are appropriate to execute after the KB snapshot file is warmbooted. For example, the sample warmboot procedure shown below duplicates some operations performed when the KB starts, but performs other operations that depend on state information saved as part of the KB snapshot file.

```
warmboot ( )
ND : class node ;
WS : class kb-workspace = the subworkspace of mill-welcome-screen;
begin
    { Notify the user that warmbooting has occurred. }
    show WS;
    post "Warm restart of MILL application";

    { Update the saved-state display for each manufacturing station. }
    for ND = each node upon WS
        do
            call reset-graphics ( ND );
            if the status of ND is processing then
```

```
                start process-material ( ND ) after
                    max ( 0 , the process-end-time of ND - the current time );
            end;

        { Display the menu bar and resume "production" of raw material items. }
        hide WS;
        start developer-package-initialization-rules ( );
        show the subworkspace of mill-process-diagram-object with its top left
            corner 2 units to the right of and 40 units below the top left corner of
            the screen ;
        start process-material ( warehouse );
    end
```

Modular Warmboot Procedures

When a KB contains modules that need to define their own warmboot procedures, some mechanism is needed that invokes them all in the correct order. You could write a procedure named warmboot that does this, but GFR provides a more general capability: it contains a system-defined procedure named warmboot that automatically executes any other warmboot procedures.

When you warmboot a snapshot of a KB that includes GFR, G2 invokes GFR's warmboot procedure just as it would a user-defined procedure with that name. The GFR procedure scans the KB for items of class gfr-startup-object, each of which can define a warmboot procedure for a module, and executes the procedures specified by the items in the order defined by the module hierarchy.

A warmboot procedure for use with GFR has a different signature than a procedure named warmboot, but otherwise does the same types of things in the same ways that a non-modular warmboot procedure does, as described under Non-Modular Warmboot Procedures. For further information about modular warmboot procedures, see the chapter on managing modules in the *G2 Foundation Resources User's Guide*.

Caution If a KB contains more than one procedure named warmboot, the duplicate names could cause G2 to invoke the wrong one. Therefore, a KB that requires GFR *must* use GFR to execute any warmboot procedures, and *must not* contain any procedure named warmboot except the one supplied by GFR.

Warmbooting with Catch-Up

G2's default behavior for initializing a warmbooted KB snapshot file might not be appropriate for a KB that is designed to run continuously. For this reason, you can direct G2 to warmboot a KB so that its processing can catch up from the current time saved in the snapshot file to the current real time.

To catch up to the current real time when warmbooting:

➔ Select the warmboot afterwards with catch-up feature option in the Load KB dialog.

G2 initializes the KB snapshot file as follows:

- G2 sets the scheduler's internal *current time* setting to the *current time* saved in the KB snapshot file.
- G2 sets the scheduler-mode attribute in the resulting current KB's Timing Parameters system table to as fast as possible.

After warmbooting a KB in this manner, and after the scheduler's *current time* catches up to become equal to the *current real time*, your KB should reset the scheduler-mode attribute to the value real time. If your KB does not reset the scheduler-mode attribute, then G2 continues to run with a setting of as fast as possible.

To reset the scheduler mode to use real time processing after a warmboot:

1 Create a procedure that changes the scheduler-mode attribute in the KB's Timing system table from as fast as possible to real time when the scheduler's *current time* is greater than or equal to the *current real time*.

For example, the following procedure restores the scheduler-mode attribute to the value at the time the KB snapshot file was saved. This procedure assumes that the warmbooted KB includes a text parameter named text-parameter-holding-saved-scheduler-mode, whose value is equal to the value of the scheduler-mode attribute in the Timing Parameters system table at the time the KB was saved to its snapshot file.

```
change-to-real-time-when-caught-up( )
begin
    repeat
        wait for the current real time = the current time;
        exit if the current time >= the current real time;
    end;
    change the text of the scheduler-mode of timing-parameters
      to the current value of
      text-parameter-holding-saved-scheduler-mode;
end
```

2 Create a warmboot procedure so that it starts change-to-real-time-when-caught-up.

G2 executes warmboot when warmbooting the KB. For example:

```
warmboot( )
begin
    . . .
    start change-to-real-time-when-caught-up ();
    . . .
end
```

3 Warmboot the KB snapshot file selecting the warmboot afterwards with catch-up feature option.

After G2 warmboots the KB, when the scheduler's *current time* becomes equal to the *current real time*, the change-to-real-time-when-caught-up procedure restores the value of the scheduler-mode attribute in the Timing Parameters system table.

For a KB that is warmbooted in this manner, if you reset the resulting current KB, G2 resets the scheduler-mode attribute to its value that is saved in the KB snapshot file.

Note You cannot use the warmboot afterwards with catch-up feature option to warmboot KB snapshot files saved under G2 Version 3.0 revision 0 or earlier.

Merging a KB File

Merging a KB file means adding the knowledge in that KB file to the current KB. Merging a KB is similar to loading a KB, as described under Loading a KB. The same syntax and options applies to both operations. When KBs merge, the KB that is already loaded is called the **primary KB**, and the KB that is merged into it is called the **secondary KB**.

You merge one secondary KB at a time into the primary KB. If the secondary KB is a modularized KB, G2 also merges the KB files that contain modules that are directly required by the secondary KB. This is described in Merging a Modularized KB into the Current KB.

When you merge one KB into another, G2 checks that the class-definitions in the two KBs are consistent. G2 provides a variety of techniques for resolving inconsistencies between merged KBs, as described in Detecting Conflicting Class-Definitions.

When two KBs are merged:

- The resulting KB contains all the information in both knowledge bases, except where conflicting class-definitions required changes.
- The visible workspaces from the secondary KB appear behind the visible workspaces of the primary KB.
- By default, the system tables of the primary KB remain in effect. You can choose to install the system tables of the secondary KB, thus replacing the currently installed system tables.

Note Loading a modularized KB actually performs a merge operation for each module that the loaded KB file directly requires. Thus, the entire discussion of merging KB files applies also to loading modularized KB files. For more information, see Working with Modularized KBs.

To merge a KB file interactively:

1 Pause or reset the current KB by selecting Pause or Reset from the Main Menu.

2 Select Main Menu > Merge KB.

G2 displays the Load KB dialog with the merge in this KB option automatically selected.

3 Navigate the directory structure and specify the filename to merge in the same manner as when you load a KB.

For information on interacting with the Load KB dialog, using wildcards in filenames, and specifying options, see Loading a KB.

To install system tables when merging a KB file interactively:

➔ Follow the preceding instructions, but choose the merge in this KB and install its system tables option in the Load KB dialog.

To merge a KB file programmatically:

➔ Execute the g2-merge-kb or g2-merge-kb-ex system procedure, as described in KB and Module Operations in the *G2 System Procedures Reference Manual*.

G2 reports its progress as it merges the KB. It does not indicate the real size of the KB being merged.

To merge a KB file using an initialization file.

➔ See Using an Initialization File.

Working with Duplicate Items in KBs

Most items store their names in the names attribute of the item. Some items store their name in an equivalent class-specific attribute. For example, the relation-name attribute stores the name of a relation. Items have the same name if their names attributes, or their class-specific equivalents, contains the same name.

Items that have the same name are called **duplicate items**. To detect duplicate items, G2 considers only their names; the items may or may not have the same class type or be functionally equivalent.

Duplicate Definitional Items

Definitional items include all definitions that you create interactively from the KB Workspace > New Definition menu, as well as a rule, which you create from the New Rule menu.

G2 allows a KB to contain duplicate definitional items for anything *except* class-definitions, which includes the following definitional items:

external-simulation-definition
procedure
foreign-function-declaration
image-definition
frame-style-definition
relation
function-definition
remote-procedure-declaration
generic-formula
rule
tabular-function-of-1-arg
language-translation
units-of-measure-declaration
method
user-menu-choice
tokenizer
text-conversion-style

Note G2 does *not* check method declarations for consistency.

When you merge a KB with a duplicate definitional item for anything except a class-definition, G2 creates duplicate items in the KB.

Caution When you merge a KB with a duplicate definitional item for anything except a class-definition, using the automatically resolve conflicts option, G2 deletes the duplicate item from the merged KB.

Where duplicate items exist in a KB, G2 places a warning in the Notes attribute of each item having the duplicate name, and posts no other notification. Here is an example of the notes attribute for two items with the same name:

OK, and note that this is one of 2 distinct items named input-1

When duplicate items exist and more than one of the items satisfies a reference, which item G2 chooses is not predictable. The choice may not be the same from one reference to the next, which can cause unintentional results.

To find items with the same name:

1 Select Main Menu > Inspect.

2 Enter this command in the Inspect edit box:

```
show on a workspace every item with notes
```

3 For the items returned that have the note:

```
this is one of integer items named name
```

enter this command in the Inspect edit box:

```
show on a workspace every item I such that
    the names of I exists and the names of I is name
```

Duplicate Class-Definitions

Class-definitions have the same name if their class-name attribute contains the same name. Class-definitions that have the same name are called **duplicate definitions**. To detect duplicate definitions, G2 looks only at the names of the classes they define; the definitions may or may not have the same type or be functionally equivalent.

Within a G2 process every class name must be unique. The G2 compiler does not allow you to specify an existing class name in the class-name attribute of a class-definition. However, class-definitions with duplicate names can occur when KBs contain modules not developed in the same G2 process, or when additional modules are merged into the current KB.

G2 generates backup class-definitions when writing a KB module. It saves all user-defined class-definitions that determine the inheritance of the items in the module, even when the definitions do not reside in the module. These class-definitions are called **backup definitions,** and G2 uses them to determine the inheritance of items in the module and to notice differences between the backup definition and a same-named class-definition in another module.

When a KB module that contains a class-definition is merged into a G2 process that already contains a class-definition of the same name, the existing class-definition is called the **primary definition,** and the class-definition being merged in is called the **secondary definition**.

At load time, G2 avoids all duplicate class-definitions in one of several ways, as described in the rest of this chapter.

Identical Duplicate Definitions

G2 considers two class-definitions to be **identical definitions** when they:

- Are of the same type (class-definition, object-definition, connection-definition, or message-definition).

- Have the same name, attributes, and initial and default values.
- Specify their attributes in the same order.

When two class-definitions are identical, their attribute tables look exactly the same. The value of an attribute that can contain multiple terms, such as class-specific-attributes, must list the same terms in the same order to be considered identical.

When two KB modules contain *identical* class-definitions, and one KB is merged into the other, the secondary definition is redundant. G2 therefore:

- Deletes the secondary definition. G2 also deletes any subworkspace hierarchy of the class-definition.
- Converts any instances of the deleted definition to be instances of the primary definition; the instances are otherwise unaffected.

The deletion of the secondary class-definition prevents it from existing as a duplicate class-definition in the combined KB. Since the class-definitions were identical, and the converted instances are unchanged, the deletion and conversion have no functional effect, so G2 does not post any notification that it has occurred.

Nonidentical Duplicate Definitions

When two KB modules contain duplicate class-definitions that are not identical, the definitions are in conflict: neither can be deleted in favor of the other without risking functional change. The rest of this chapter describes such situations and shows you what to do about them.

Detecting Conflicting Class-Definitions

In order to understand conflicting definitions and their resolution, you need to understand G2 classes, as described in Classes and Class Hierarchy, and G2 definitions for extending the class hierarchy, as described in Definitions. The rest of this chapter assumes that you understand those topics.

When you merge one KB module into another, either in the process of loading a multi-module KB or explicitly merging an additional module, G2 checks that each class-definition in the merging module is consistent with those in the resident KB modules. A merging KB module is consistent with the resident modules if it has no class-definitions with duplicate names, or if every duplicate class-definition is identical to the resident class-definition of the same name. Identical pairs of class-definitions are handled as described under Identical Duplicate Definitions.

If a secondary definition has a name that is also used in the resident KB modules, but has differing attributes, conflicting definitions exist. This section shows you how such conflicts can be resolved.

Automatically Resolving Conflicting Class-Definitions

You can direct G2 to resolve conflicts among definitions of classes automatically. This ability is sometimes referred to as **automerge**. The automatically resolve conflicts option on the Load KB dialog and Merge KB dialog determines whether G2 automatically resolves class-definition conflicts. By default, this option is selected because it is very time consuming to resolve them yourself.

To resolve conflicts automatically when loading and merging KBs:

➔ Make sure that the automatically resolve conflicts option in the Load KB dialog or Merge KB dialog is selected.

When merging KBs or loading modularized KBs with automerge selected, G2 checks each pair of conflicting definitions to see whether they can be merged automatically.

- Two definitions of the same type can be automerged if they have the same foundation classes OR no instances of the secondary definition exist.
- A class defined on an object-definition, connection-definition, or message-definition can be automerged into a class defined on a class-definition if they have the same foundation classes OR no instances of the secondary definition exist.
- A class defined on a class-definition *cannot* be automerged into a class defined on an object-definition, connection-definition, or message-definition, even if they have the same foundation classes.

When G2 automerges two KBs, it does the following for each pair of classes that can be merged automatically:

- For each attribute that differs, G2 changes the attribute in the secondary definition to be equivalent to its corresponding attribute in the primary definition.
- In the secondary definition item, G2 opens the change attribute and executes:

  merge all instances and subclasses into definition for *primary-definition*

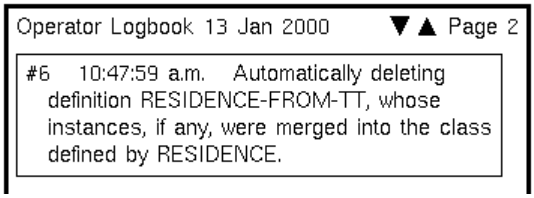
- After G2 automerges two definitions, it deletes the secondary definition.

Automerging a class defined on a object-definition, connection-definition, or message-definition into a class defined on a class-definition cannot be done by directly transferring attributes, because the syntax differs in the two types of definition. G2 carries out such a merge by changing the syntax of the information in the secondary definition as needed to fit into a class-definition.

Automerging two definitions resolves every difference between the primary and secondary definitions in favor of the primary definition:

- Attributes defined in the secondary definition but not the primary definition disappear from subclasses and instances of the secondary definition.
- Attributes defined in the primary definition but not the secondary definition are added to subclasses and instances of the secondary definition.
- Attributes that exist in both definitions but have different properties use the properties in the primary definition. Subclasses and instances of the secondary definition change accordingly.

For a successfully merged pair of class-definitions with duplicate names, G2 displays a message on the Logbook. For example:



If all conflicting definitions can be merged automatically, the KBs themselves have been successfully automerged. If any pair of definitions cannot be automerged, G2 treats them as it does all conflicting definitions during an ordinary merge, as described in the next section.

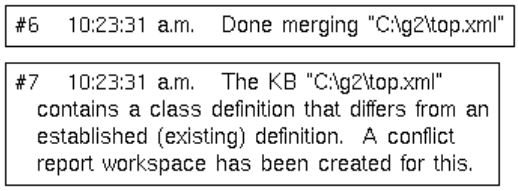
Manually Resolving Conflicting Class-Definitions

We recommend that you take advantage of G2's automerge facility which is selected by default on the Load KB and Merge KB dialogs. This section explains what happens when you turn off automerge or, when automerging, G2 encounters an unmergable conflict.

G2 Notification of Conflicting Class-Definitions

G2 does the following when it detects conflicting class-definitions:

- Displays a messages in the Operator Logbook indicating that conflicting definitions exist. For example:



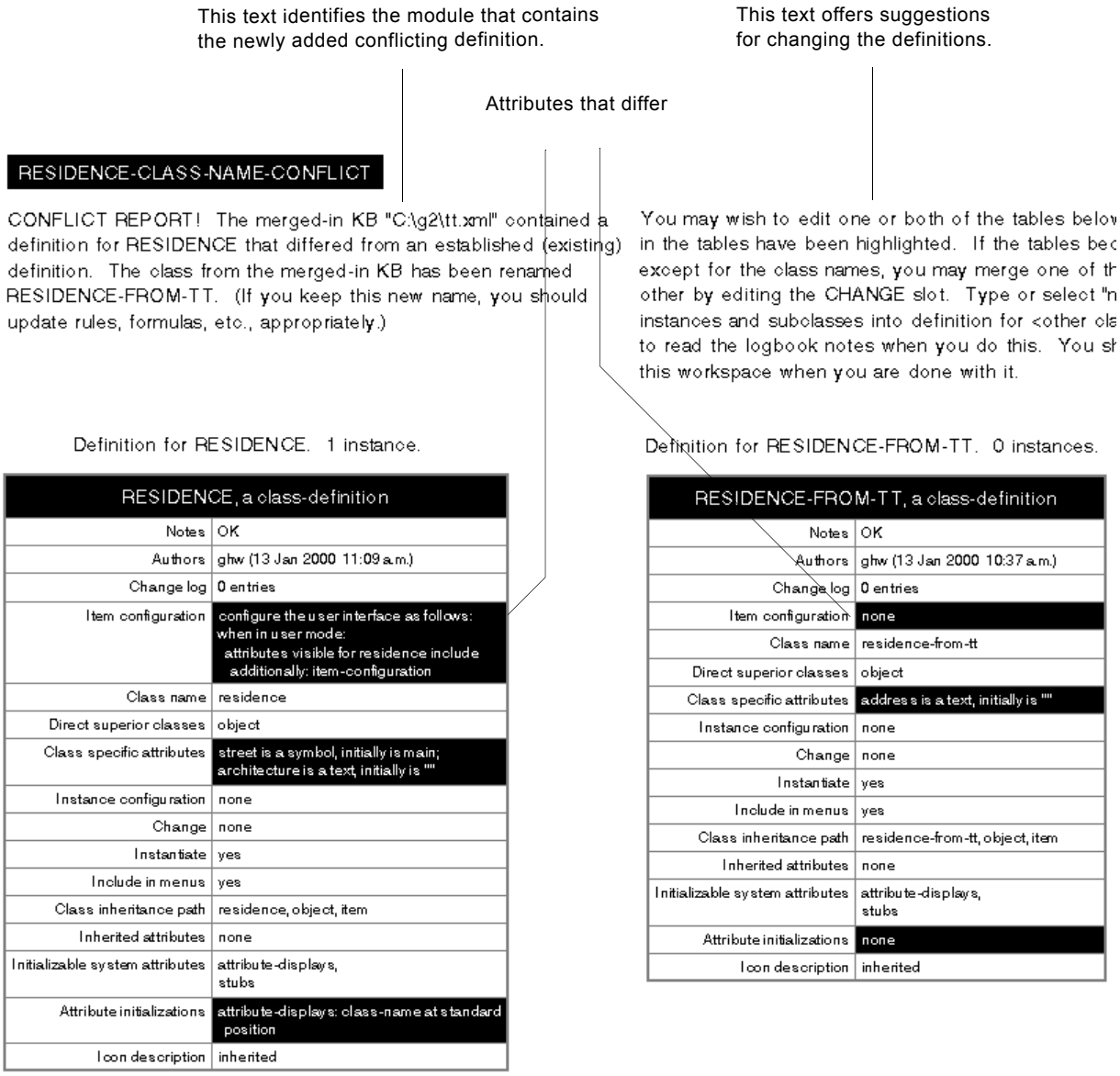
- Changes the name in the class-name attribute of the secondary definition. This also changes the class name for all instances of that class that are being merged. The new secondary name has the form:

  primary-name-from-*module*

  where *primary-name* is the original name of the definition, and *module* is the module in which the secondary definition exists.

- Creates a **conflict workspace**. The workspace displays the tables of the two conflicting definitions, with the primary definition on the left., highlights the corresponding attributes in the two conflicting definitions whose values are not equivalent.

This figure shows the contents of a typical conflict workspace:



RESIDENCE, a class-definition	
Notes	OK
Authors	ghw (13 Jan 2000 11:09 a.m.)
Change log	0 entries
Item configuration	configure the user interface as follows: when in user mode: attributes visible for residence include additionally: item-configuration
Class name	residence
Direct superior classes	object
Class specific attributes	street is a symbol, initially is main; architecture is a text, initially is ""
Instance configuration	none
Change	none
Instantiate	yes
Include in menus	yes
Class inheritance path	residence, object, item
Inherited attributes	none
Initializable system attributes	attribute-displays, stubs
Attribute initializations	attribute-displays: class-name at standard position
Icon description	inherited

RESIDENCE-FROM-TT, a class-definition	
Notes	OK
Authors	ghw (13 Jan 2000 10:37 a.m.)
Change log	0 entries
Item configuration	none
Class name	residence-from-tt
Direct superior classes	object
Class specific attributes	address is a text, initially is ""
Instance configuration	none
Change	none
Instantiate	yes
Include in menus	yes
Class inheritance path	residence-from-tt, object, item
Inherited attributes	none
Initializable system attributes	attribute-displays, stubs
Attribute initializations	none
Icon description	inherited

Responding to Conflict Workspaces

The existence of unresolved conflicts among merged KBs does not prevent G2 from running the resulting KB, but the results may not be what was intended when the KBs were designed. To insure correct results, all conflicts should be resolved, and the previously conflicting definitions merged into one.

Most conflicts are easily resolved, because they result from minor incompatibilities. In such cases, the answer is usually to change the attributes in

the secondary definition and leave the attributes in the primary definition intact, but this approach is not required.

The attribute tables on a conflict workspace are real tables: any change to them changes the corresponding definition. Using a conflict workspace to eliminate conflicts and merge definitions is exactly the same as merging two definitions independently of KB merging. The conflict workspace just provides a convenient interface to the process.

To merge definitions with minor incompatibilities:

➔ Follow the directions under Merging Classes, using the tables on the conflict workspace rather than opening separate copies of the definitions' tables.

Some conflicts are not so easily resolved, because they are unusual or complex in some way. The next section contains examples of various conflicts and shows you what to do about them.

Examples of Manual Conflict Resolution

This section describes various types of conflicts that can arise when you merge inconsistent KBs and shows you how to resolve each of them.

Completely New Version of the Same Class-Definition

Assume that the conflicting definitions are related: one definition is a completely new version of the other, and that the new version must replace the old version.

If the secondary definition contains the *new* version, follow these steps:

1. Edit the attributes in the definition already in the current KB. Edit the definition so that its attributes are equivalent to the secondary definition's attributes.
2. Use the change attribute's merge option on the secondary class to merge all instances and subclasses of the secondary class into the primary class.
3. Unless you have a specific use for it, delete the secondary definition.
4. Delete the conflict workspace.

Name Conflicts between Independent Class-Definitions

Assume the conflicting definitions are *not* related: the two definitions are intended to define distinct classes in your KB. For example, two different developers might have accidentally given two definitions the same name. To resolve this conflict, change the class-name attribute in one or both of the definitions.

Unless the direct-superior-classes attributes in the two definitions have the same foundation class, you cannot accomplish this form of conflict resolution by using

the Merge KB command's automatic conflict resolution feature. For more information, see Unresolvable Conflicts between Class-Definitions.

Separate Development of Groups of Attributes

Assume the conflicting definitions are related. Further, assume that two developers made independent changes to separate copies of a shared definition. However, in this case, each developer added distinct sets of information to the definition, such as distinct class-specific attributes.

To resolve this conflict:

1. For each pair of corresponding attributes in the conflicting definitions, determine which version you intend to keep.
2. Edit the definition already in the current KB so that its attributes contain the values you want to retain.
3. Use the change attribute's merge option on the secondary class to merge all instances and subclasses of the secondary class into the primary class.
4. Unless you have a specific use for it, delete the secondary definition.
5. Delete the conflict workspace.

Separate Development of Specific Attribute Values

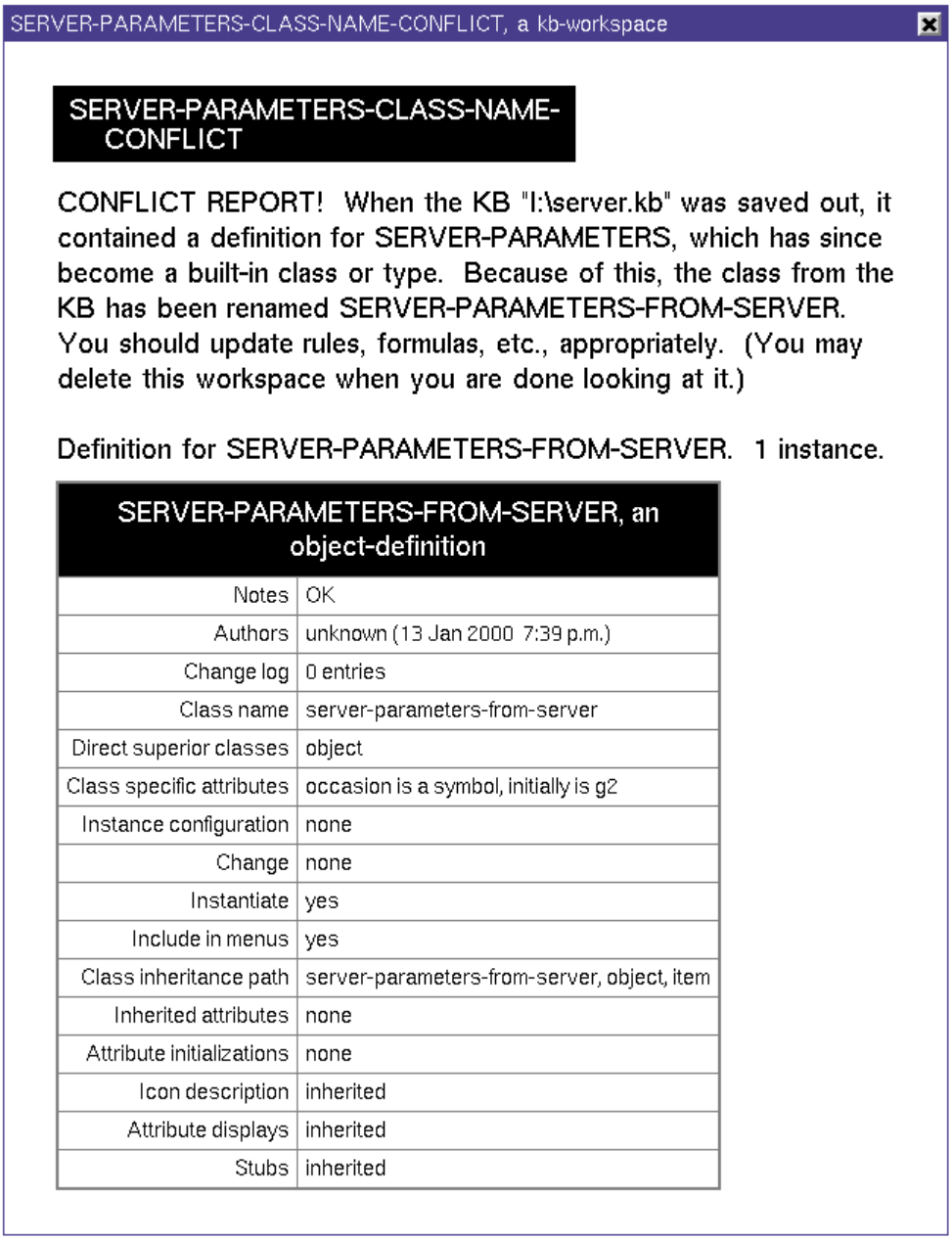
Assume that the conflicting definitions are related and that two developers made independent changes to separate copies of a shared definition. More specifically, assume that each developer simply assigned different default values within the same set of class-specific attributes.

To resolve this conflict, for each pair of corresponding attributes in the conflicting definition, first determine which value in each differing attribute you intend to keep. Next, follow Steps 2 through 5 listed in Separate Development of Groups of Attributes.

Conflict Due to Upgrading to a New G2 Version

Assume that the merged KB contains a definition whose name is the same as a system-defined class name. This is possible only if the merged KB was developed under an older version of G2.

In this case, G2 displays a conflict workspace containing only *one* attribute table, as shown in this figure for an object definition that defines a server-parameters class:



To resolve this conflict, simply edit the name in the definition item's class-name attribute. Instance items based on the edited definition item automatically inherit the changed class name.

Conflict between Original and External Definitions

When writing a KB module, G2 saves all user-defined class-definitions that determine the inheritance of the items that reside in the module, whether or not the needed class-definitions also reside in the module. These class-definitions are called backup definitions.

If you later merge a module containing an external class-definition with the module containing the original, G2 compares the two definitions. If they are identical, G2 merges the external class-definition into the original, then deletes the backup. This is the normal course of events.

If the external definition is not identical to the definition of the same name found in the required module, G2 creates a conflict workspace for the two definitions, as with any conflict. Proceed as described under Manually Resolving Conflicting Class-Definitions.

If no definition of the same name as the external definition exists in any required module, G2 creates a new workspace named backup-definitions-for-*module-name*, where *module-name* is the name of the module associated with the backup definition. G2 places the backup definition on this workspace. The backup definition is thereafter a real definition, identical except in location with the missing original, and can be used as any definition can be.

Differences between Class-Specific Attributes

The possible differences between two class-specific attribute declarations in a definition include:

- In one definition, the attribute is untyped, and in the other definition, the attribute is untyped *and* has a default value. For example, these two declarations conflict:

  vehicle-identifier

  vehicle-identifier initially is V103

- A difference exists in the declared type of the attribute in the two definition items. For example, these two declarations conflict:

  vehicle-identifier is an integer, initially is 0

  vehicle-identifier is a symbol, initially is V103

- In one definition, the attribute is declared to be an instance of a particular class, and in the other definition, the attribute is declared to be an instance of a different class. For example, these two declarations conflict:

  vehicle-identifier is an instance of a custom-message

  vehicle-identifier is an instance of a borderless-free-text

- In one definition, the attribute is declared to be an instance of some class, and in the other definition, the attribute is declared is given by any class of variable or parameter.

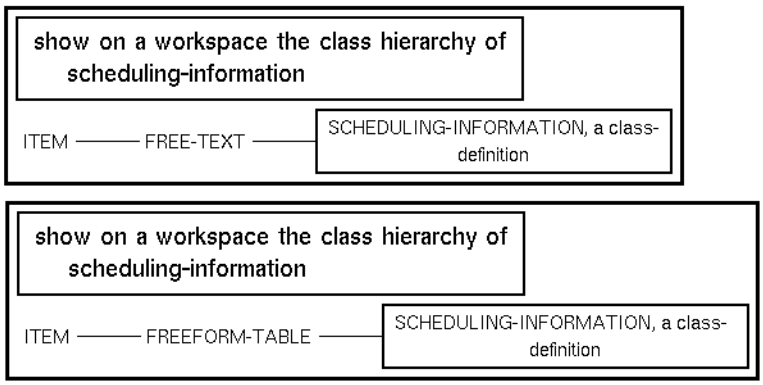
Unresolvable Conflicts between Class-Definitions

Some conflicts between definitions cannot be resolved, either manually or automatically, by editing those items. In these cases, the conflicting definitions must remain distinct. Instead, you must make more significant changes to your class hierarchy.

Suppose two definitions have the same name but specify different superior classes. If those superior classes have different foundation classes, and each of the conflicting classes has at least one instance, the conflict between the two definitions cannot be resolved.

To illustrate, assume that different G2 developers have created two different but related modules. One module contains a class-definition named scheduling-information, whose superior class is the system-defined class free-text. The other module contains a scheduling-information class-definition whose superior class is the system-defined class freeform-table. Finally, assume that each KB has at least one item that is an instance of the scheduling-information class.

These two Inspect workspaces show the class hierarchies for the two scheduling-information definition items:



After merging the KBs with the automatically resolve conflicts option selected, G2 displays the following conflict workspace:

SCHEDULING-INFORMATION-CLASS-NAME-CONFLICT

CONFLICT REPORT! The merged-in KB "C:\g2\bb.xml" contained a definition for SCHEDULING-INFORMATION that differed from an established (existing) definition. The class from the merged-in KB has been renamed SCHEDULING-INFORMATION-FROM-BB. (If you keep this new name, you should update rules, formulas, etc., appropriately.)

You may wish to edit one or both of the tables below. Dif in the tables have been highlighted. If the tables become i except for the class names, you may merge one of them in other by editing the CHANGE slot. Type or select "merge instances and subclasses into definition for <other class>." to read the logbook notes when you do this. You should this workspace when you are done with it.

Definition for SCHEDULING-INFORMATION. 2 instances.

Definition for SCHEDULING-INFORMATION-FROM-BB. 0 instances.

SCHEDULING-INFORMATION, a class-definition	
Notes	OK
Authors	ghw (14 Jan 2000 10:22 a.m.)
Change log	0 entries
Item configuration	none
Class name	scheduling-information
Direct superior classes	free-text
Class specific attributes	none
Instance configuration	none
Change	none
Instantiate	yes
Include in menus	yes
Class inheritance path	scheduling-information, free-text, item
Inherited attributes	none
Initializable system attributes	default-text-box-colors
Attribute initializations	none
Icon description	inherited

SCHEDULING-INFORMATION-FROM-BB, a class-definition	
Notes	OK
Authors	ghw (14 Jan 2000 10:20 a.m.)
Change log	0 entries
Item configuration	none
Class name	scheduling-information-from-bb
Direct superior classes	freeform-table
Class specific attributes	none
Instance configuration	none
Change	none
Instantiate	yes
Include in menus	yes
Class inheritance path	scheduling-information-from-bb, freeform-table, item
Inherited attributes	none
Initializable system attributes	table-size, default-cell-format, default-evaluation-setting
Attribute initializations	none
Icon description	inherited

The conflict workspace shows that only the direct-superior-classes attribute differs in the two versions of scheduling-information. The conflict between these two definitions cannot be resolved, because G2 does not allow you to change the class of an instance whose foundation class is free-text into an instance whose foundation class is not free-text, or one of its system-defined subclasses.

4

Workspaces

Shows how to use workspaces to organize your KB's items.

Introduction

Workspaces are fundamental building blocks for constructing a knowledge base (KB). Each workspace organizes a set of items within a region. You can also link these regions together to form hierarchies of regions, called **workspace hierarchies**.

You use workspaces primarily to collect and to contain other items:

- A workspace forms a two-dimensional region upon which you place items interactively or programmatically. An item has an absolute location within the coordinate system of its parent workspace. The items upon the same workspace also have a spatial relationship to each other.
- Many operations on a workspace also affect the items upon it, for example:
  - Cloning a workspace creates copies of the items upon that workspace.
  - Changing the scale of a workspace changes the scale at which G2 displays all the items upon that workspace.
  - Deleting a workspace deletes all items on that workspace, as well as all workspaces in the workspace subhierarchy. Items that depend on deleted class-definitions for their inheritance are also deleted, regardless of their workspace or module assignments.

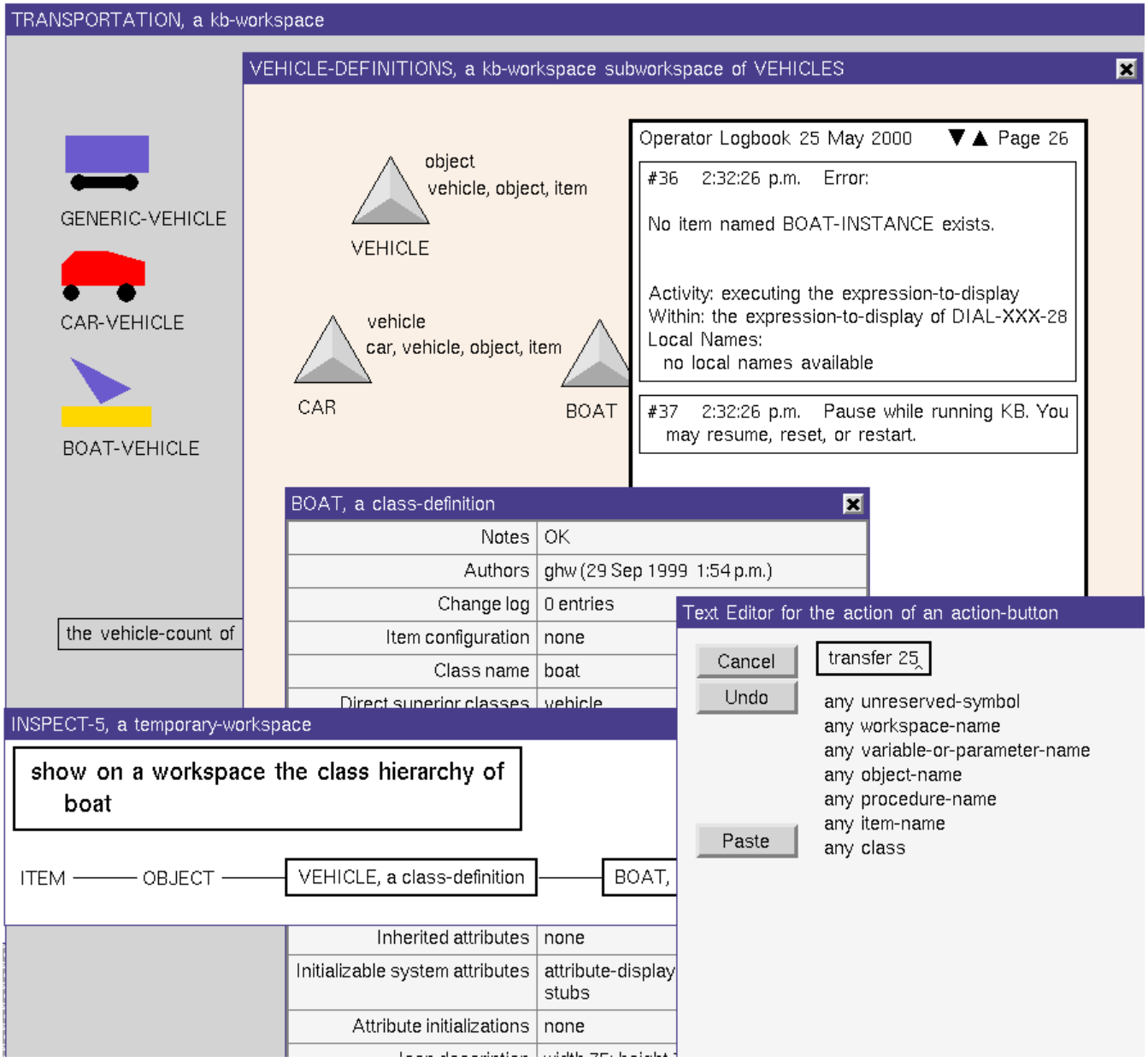
Workspaces also serve other important purposes:

- G2 displays all KB knowledge on workspaces. **Note:** The display of attribute tables are an exception.
- You can print workspaces. **Note:** You cannot print individual items.
- You can associate a hierarchy of workspaces, and the items they contain, with a module.
- You can configure in similar ways items that are located within the same part of the workspace hierarchy.
- You can declare a workspace as proprietary by using special configuration statements to affect the behavior of the items in the hierarchy.
- A G2 process's local G2 window and the Telewindows connected to that process can each display independent sets of workspaces, and each can display any workspace at a different scale and at a different position within its own window.

In addition, you can create, delete, scale, clone, display, hide, and configure any workspace. You can also associate custom borders and custom background images with any workspace.

Kinds of Workspaces

Within the G2 developer's environment, you work with several kinds of workspaces. This figure shows a variety of workspace types.



Common Features of Workspaces

Some common features of workspaces are:

- Each workspace appears as a rectangle.
- Each workspace has a background and border.
- Workspaces can appear on top of each other.
- G2 displays each workspace at its default size or at a factor of its default size.

KB Workspaces

One kind of workspace, called a **KB workspace**, is designed to be a permanent part of your KB. A KB workspace is an item of the system-defined class named kb-workspace.

KB workspaces are the only workspaces that can contain other items. KB workspaces are also the only workspaces that you can save into a KB file.

Your KB can contain any number of KB workspaces. Any item, except another workspace, can reside upon a KB workspace. You can create, delete, show, hide, change the color of, scale, move, clone, activate and deactivate, and print KB workspaces.

G2 offers actions that manipulate KB workspaces. You can refer to KB workspaces in expressions. For more information, see Actions That Apply to KB Workspaces and Expressions That Refer to KB Workspaces.

Note Throughout this chapter and throughout this guide, we typically refer to items of the kb-workspace class as *workspaces*. We differentiate KB workspaces from other kinds of workspaces only when required for clarity.

Other Workspaces

The G2 developer's environment displays other kinds of workspaces, as well. These workspaces appear as you open and interact with G2 editors and facilities, and as you make choices from G2 menus.

These workspaces are not items, and you cannot save them into a KB file. You cannot refer to these workspaces in actions or statements.

The other kinds of workspaces are:

Type of Workspace	Description
Operator Logbook	Displays error messages and informational messages from G2.
Inspect workspace	Displays the results of Inspect commands.
Text Editor workspace	Displays editing sessions.
Class List workspace	Presents lists of classes, items, or other entities for entering in the Text Editor.
Message Board	Displays messages from inform and post actions.
Scrapbook workspace	Contains pieces of text used for insertion in the Text Editor.
Icon Editor workspace	Displays Icon Editor sessions.

The Developer's Environment, describes how to interact with several of these workspaces.

Working with Workspaces

The following operations are common to all workspaces.

To create a workspace interactively:

➔ Choose Main Menu > New Workspace.

When you create a new workspace, it is not associated with any other item. This type of workspace is called a **top-level workspace**.

When you create a new workspace interactively, G2 displays its center at the current center of the G2 process's window.

To create a workspace programmatically:

➔ create a kb-workspace

G2 does not automatically display a workspace that is created programmatically. To display a new workspace programmatically, use the show action.

To display a workspace's menu:

➔ Click on the background of the workspace.

This menu is called the KB Workspace menu.

To move a workspace using the mouse:

➔ With the mouse pointer on the workspace background, depress any mouse button and move the mouse.

To display a workspace on top of all other workspaces:

1 Click on the background of the workspace to display its menu.

2 Choose Lift to Top.

or, for standard-style workspaces:

➔ Click the title bar outside of the hide button.

To display a workspace beneath all other workspaces:

1 Click on the background of the workspace to display its menu.

2 Choose Drop to Bottom.

To minimize the extent of the workspace borders:

1 Click on the background of the workspace to display its menu.

2 Choose Shrink Wrap.

To produce a cascade display of all displayed workspaces:

➔ Choose Main Menu > Miscellany > Neatly Stack Windows.

You can also use several system-defined keystroke commands to affect the position and scale of any workspace. For information on these commands, see Appendix C, Mouse Gestures, Key Bindings, and Shortcut Keys.

Operating on an Area of a Workspace Interactively

You can work with a group of items in the same workspace. The behavior depends on whether you are using standard or classic user interface mode.

Using Standard Selection

When running G2 in standard user interface mode (`-ui standard`), you use standard selection to select a group of items, then work on the group by choosing from the menu for the selection. You can move, clone, transfer, align, distribute, and delete all items in the selection.

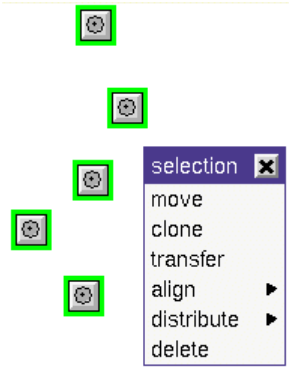
When aligning items, at least two items must be selected. When distributing items, at least three items must be selected. The outermost two items are unchanged, and the remaining inner items are positioned between the outermost items such that the space between any two items is constant.

For more information about working with selections, see Mouse Gestures for Selection and Mouse Gestures for Interacting with Selections.

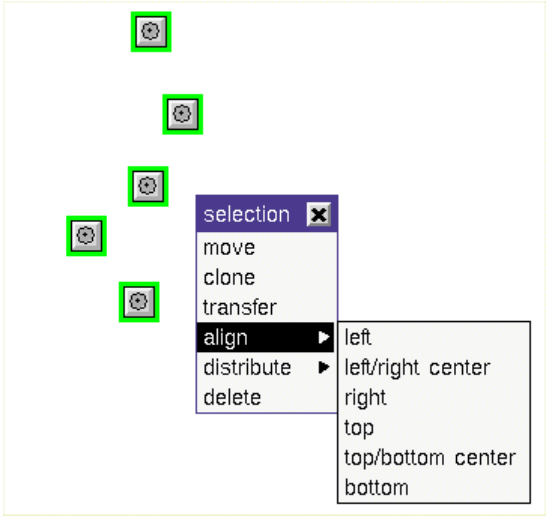
To work with a group of items on a workspace, using standard selection:

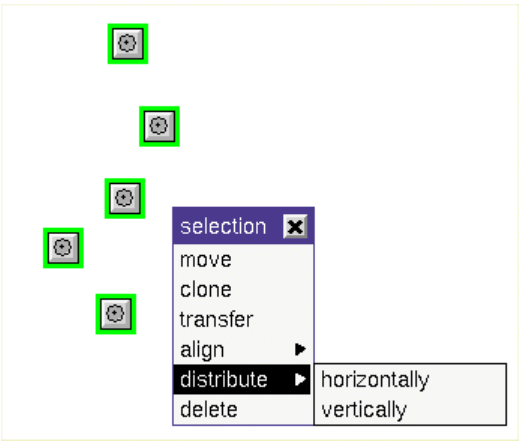
1. Drag in the open area of a workspace to select a group of items within a bounding box.
2. Mouse right on any item to display the popup menu for the selection.
3. Choose the operation for the selection.

This figure shows a selection and the popup menu for the selection:



Here are the align and distribute submenus:





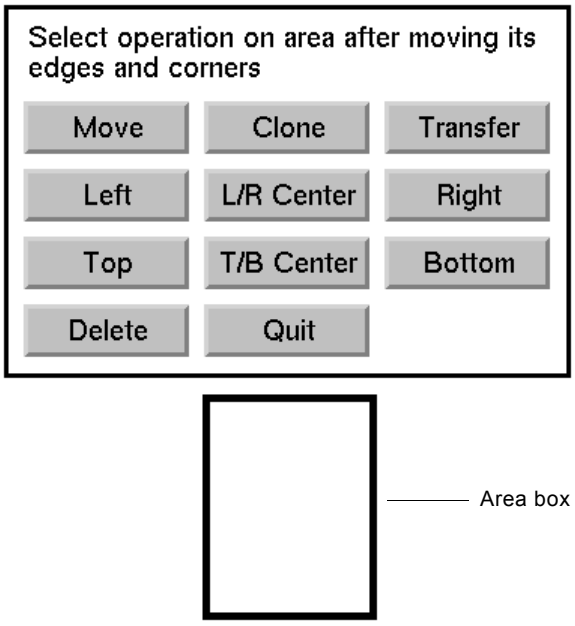
Using Operate on Area in G2 Classic

When running G2 in classic user interface mode (`-ui classic`), you use the Operate on Area menu choice to select items by drawing an **area box** around items on a workspace, then work with the group as you would for a single item.

To work with a group of items on a workspace, using Operate on Area:

1 Choose KB Workspace > Operate on Area.

This dialog box appears:



2 If the dialog box is obscuring the items you want to select, move the dialog box out of the way.

3 Position the area box to surround the items of interest.

a To change the size of the area box, click the mouse on a side and drag the side in or out to shrink or stretch the area box on that side, or click the mouse on a corner to drag the corner out or in to pull two sides at once.

b To move the area box itself, place the mouse pointer anywhere inside the area box (not on the black lines themselves), and drag the area box with the mouse.

An item must be entirely within the inside edge of the area box to be included in the area, although its name box may be partially or entirely outside of the area box.

4 Press the appropriate button:

Button	Description
Move	Attaches everything enclosed in the area box to your cursor. Move to the new location and click to place.
Clone	Clones all of the items in the area box and attaches them to your cursor. Move to a new location and click to place.
Transfer	Attaches everything in the area box to your cursor. Move your cursor to another workspace and click to locate the items there.
Left	Aligns the left sides of the items in the area box with the left side of the leftmost item.
L/R Center	Aligns the left-to-right centers of the items in the area box.
Right	Aligns the right sides of the items in the area box with the right side of the rightmost item.
Top	Aligns the tops of the items in the area box with the top of the topmost item.
T/B Center	Aligns the top-to-bottom centers of the items in the area box.
Bottom	Aligns the bottoms of the items in the box with the bottom of the bottom-most item.
Delete	Deletes all of the items in the area box. G2 prompts you to confirm this operation.
Quit	Stops the Operate On Area operation. The area box and dialog box disappear. You can also quit by pressing Control + a or by starting another activity in another area.

Operating on an Area of a Workspace Programmatically

Several system procedures provide the programmatic equivalent of most of the interactive Operate on Area choices:

- g2-clear-movement-limits
- g2-get-movement-limits
- g2-set-movement-limits

These procedures are described in Movement Limit Operations in the *G2 System Procedures Reference Manual*.

Cloning a Workspace

You can clone a workspace to copy the contents of the cloned workspace to another workspace. G2 copies all items contained on the workspace, including the subworkspaces of those items, the items on those subworkspaces, and so on. Cloning workspaces is a convenient technique for quickly developing groups of items.

To clone a workspace interactively:

➔ Choose the Clone Workspace choice from the KB Workspace menu.

When you create a new workspace interactively by cloning, G2 displays the origin of the new workspace at the center of the window.

To clone a workspace programmatically:

➔ create a kb-workspace by cloning *kb-workspace*

G2 does not automatically display a workspace that is created by cloning programmatically. To display a new workspace programmatically, use the show action.

After cloning a workspace, G2 leaves the resulting cloned items with the same status as if those items had been cloned individually. For example, a cloned rule is left with a status of incomplete, and a cloned class-definition has no class name. For more information about the status of an item, see Identifying the Status Knowledge of Items.

Note If you clone a subworkspace whose top-level workspace is associated with a module, G2 automatically specifies the module-assignment attribute in the new top-level workspace as the name of that module.

Deleting a Workspace

Caution When you delete a workspace, you delete all items upon the workspace itself, the subworkspaces of those items, and so on. You also delete the dependent class-definitions and instances in the class hierarchy of a deleted class-definitions regardless of their workspace or module assignments.

If the workspace contains items that require confirmation for deletion, G2 displays a confirmation dialog before deleting the workspace. If the workspace contains only items that do not require confirmation for deletion, G2 deletes the workspace without confirmation. For example, if the workspace contains only a name box or a table other than a display, G2 deletes the workspace without confirmation.

To delete a workspace interactively:

➔ Choose the Delete Workspace choice from the KB Workspace menu.

To delete a workspace programmatically:

➔ delete *kb-workspace* {without permanence checks}

Disabling and Enabling a Workspace

When you disable a workspace, G2 behaves as if all the items on or below the disabled workspace in the workspace hierarchy do not exist. However, class definitions that reside upon a disabled workspace or upon a subworkspace under its workspace hierarchy remain in effect.

Note A disabled workspace can still be referenced and is included in existence checks in a KB such as the count of each kb-workspace.

Just as for any disabled item, the fact that a workspace is disabled is part of the knowledge stored in a saved KB file. A disabled workspace remains disabled until you enable it. You can disable both top-level workspaces and subworkspaces.

To disable an enabled workspace:

➔ Choose Disable from its menu.

To enable an disabled workspace:

➔ Choose Enable from its menu.

Hiding and Showing a Workspace

Hiding a workspace means to stop displaying it. Showing it means to display the workspace again. You can hide and show a workspace interactively or programmatically.

Hiding a Workspace

To hide a workspace interactively:

➔ Click the hide button on the right side of the workspace title bar

or

➔ Choose Hide Workspace from the KB Workspace menu.

To hide a workspace programmatically:

➔ hide *kb-workspace*

You can hide the workspace of an item, the subworkspace of an item, the workspace of an item on the superior workspace of an item, and the current workspace. For more information, see hide.

Showing a Workspace

The technique for showing a workspace depends on whether the workspace is named and whether it is a subworkspace.

To show a named workspace interactively:

➔ Choose Main Menu > Get Workspace, and select the workspace by name from the resulting submenu.

To show an unnamed workspace interactively:

➔ Use the Inspect facility to find it, by searching for an item on the workspace or by searching for all workspaces that meet a particular criteria.

To show the subworkspace of an item interactively:

➔ Choose the go to subworkspace menu option for the item.

To show the superior workspace of a subworkspace interactively:

➔ Choose Go To Superior from the KB Workspace menu of a subworkspace.

To show a workspace programmatically:

➔ show *kb-workspace*

You can show programmatically any named workspace, the subworkspace of an item, the workspace of an item that is superior to an item, and any workspace that

you can describe using a generic reference. You can also show a workspace at a particular scale and position. For more information, see show.

To ensure that a portion of the workspace is always visible, use the g2-ui-show-workspace system procedure. For details, see User Interface Operations.

Scaling a Workspace

G2 displays a workspace according to its current scale. By default, the current scale of a new workspace is the **normalized scale** for the G2 process, which G2 determines by calculating the ratio of workspace units per pixel of resolution on your computer's monitor.

The absolute size in which a workspace appears when displayed at full scale depends upon the settings specified for the `-magnification` and `-resolution` command-line options when the G2 process was launched. See Appendix A, Launching a G2 Process, for more information.

G2 displays each workspace at some factor greater than or less than its full scale. You can enlarge and shrink the size of a workspace by using keystroke commands or programmatically. G2 scales a workspace up or down to a maximum or minimum size.

To enlarge the size of a workspace interactively:

➔ Place the cursor within a workspace, and press Control + b or Control + 4 repeatedly.

To shrink the size of a workspace interactively:

➔ Place the cursor within a workspace, and press Control + s repeatedly.

See Appendix C, Mouse Gestures, Key Bindings, and Shortcut Keys for a list of all keystroke commands that affect the display of workspaces.

To change the size of a workspace programmatically:

➔ show *kb-workspace* scaled by

For a complete description of this syntax, see show.

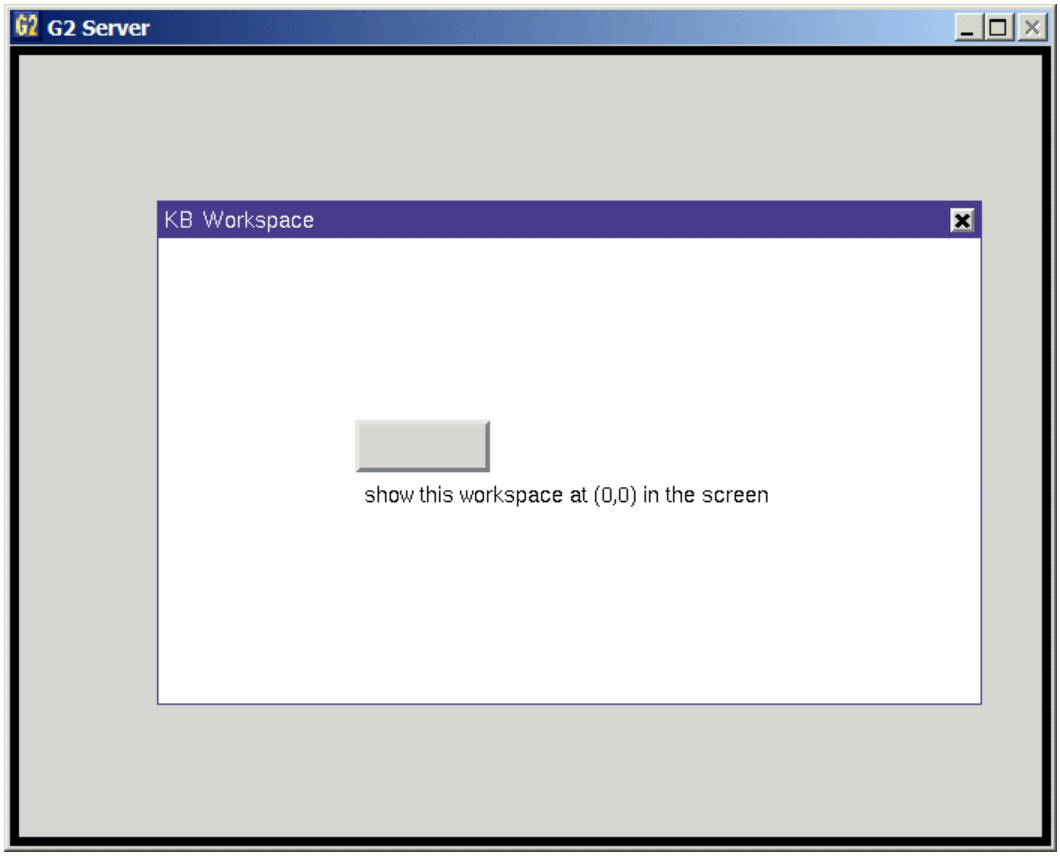
The scale at which G2 displays a workspace is specific to the window in which the user is viewing the current KB. For more information, see Displaying Independent Views of the Current KB.

Positioning a Workspace within its Window

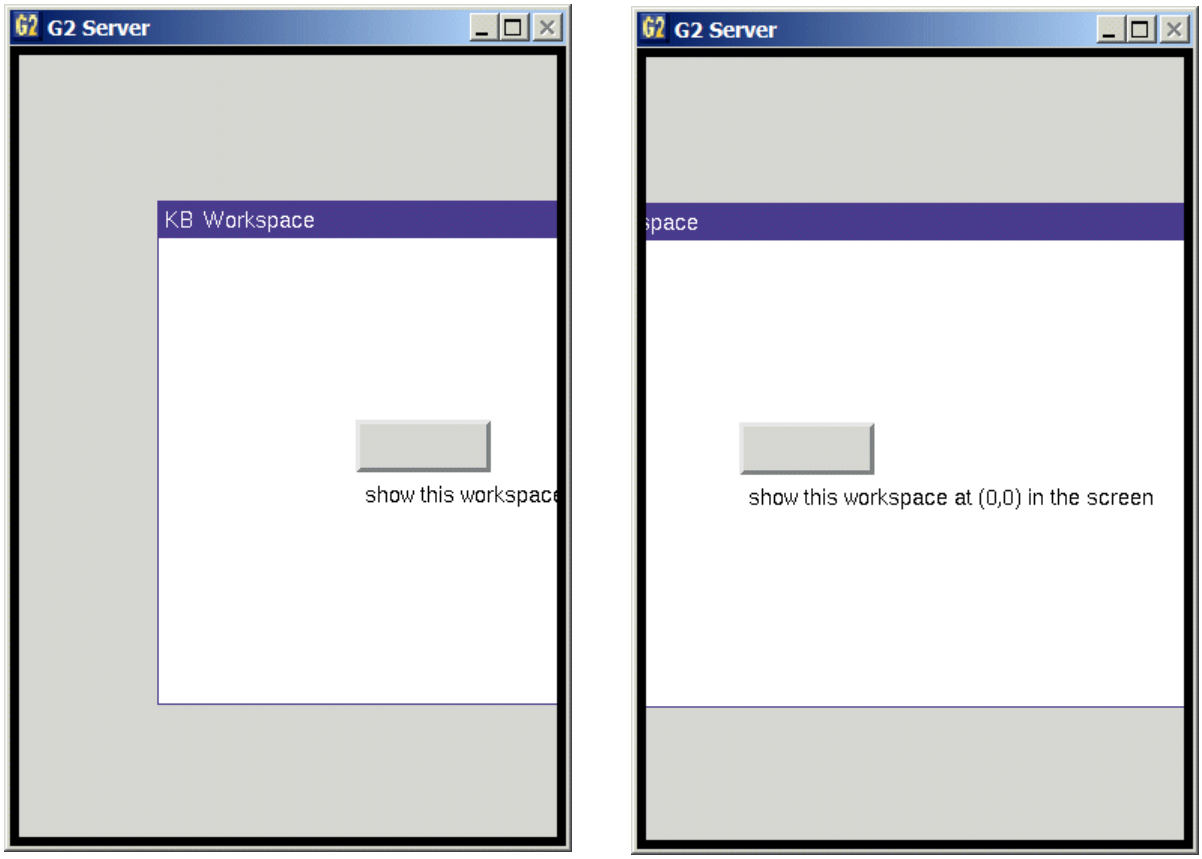
You can move a workspace within its G2 window by specifying a new location for its origin with respect to the current center of the window. G2 considers the current center of the window to be the location (0,0).

Note that the window's current center can change from moment to moment, as you resize the window by using the controls provided by your computer's window manager software.

For example, assume that you have created the workspace shown in the figure below. The workspace contains an action button that moves the workspace to the center of the window:



The figure on the left below shows the G2 window after operating-system resizing. The figure on the right shows the same window after the show action has been executed.



Positioning Items upon a Workspace

Each workspace defines its own two-dimensional x, y coordinate system measured in **workspace units**. Each workspace unit corresponds to some number of pixels of resolution on your computer's display device. By default, G2 displays workspaces at 75 workspace units per inch.

Tip You can set the ratio of workspace units per inch and the ratio of workspace units per pixel on your display device when you launch G2. See the description of G2's `-magnification` and `-resolution` command-line options in Appendix A, Launching a G2 Process.

Using the Workspace Origin

The **workspace origin** is defined as the location (0,0). When you create a new workspace interactively, G2 displays it with its origin at the current center of the G2 window.

You specify the locations of items on a workspace as coordinates with respect to the origin. For example, this action moves an item within its own workspace so that its center is at the location 100 workspace right of the origin and 200 workspace units below the origin:

```
move my-object to (100,-200)
```

You can specify a location up to 16,777,215 workspace units away from the workspace origin.

Displaying the Visible Portion of a Workspace

The **extent** of a workspace is the visible portion of is two-dimensional region. The extent of a workspace is always rectangular. When you display a workspace, you are displaying its extent.

The origin of a new workspace is also the center point of its extent. However, after you add items to the workspace, and after you shrink wrap it one or more times, its origin might no longer be the same location as the center of its extent.

It is also possible for the origin of a workspace to lie *outside* the visible portion of the workspace. Even when the origin is no longer within the visible portion of the workspace's two-dimensional region, you still refer to locations upon the workspace with respect to the origin.

Specifying Margins within the Border of a Workspace

G2 adds a number of extra workspace units, or **margins**, between the outermost items upon a workspace and its borders. As you move or transfer items to workspace regions outside of the workspace margins, G2 automatically adjusts the borders of the workspace outward.

G2 moves the workspace borders outward only when you move items to locations that are outside the current margins.

The workspace-margin attribute of a workspace determines the number of workspace units that G2 automatically maintains between any items that reside upon the workspace and each workspace edge.

Shrink Wrapping the Size of a Workspace

As stated above, G2 automatically adjusts the borders of a workspace outward as you move and transfer items outside its current margins. G2 does not automatically adjust the borders inward as you move items within the workspace margins.

To adjust the borders of a workspace so they just fit the items on the workspace is called **shrink wrapping**.

To shrink wrap a workspace interactively:

➔ Choose the Shrink Wrap choice from the KB Workspace menu.

To shrink wrap a workspace programmatically:

➔ change the size of *kb-workspace* to minimum

An item whose representation is transparent does not appear on the workspace. However, such an item occupies a region within the workspace. When shrink wrapping a workspace, G2 maintains the workspace margin outside any transparent item.

Creating and Using a Workspace Hierarchy

Each top-level workspace, the items it contains, the subworkspaces of those items, and so on, form a pattern called a **workspace hierarchy**. Each workspace hierarchy forms a tree, with the top-level workspace at the root of the tree.

Only KB workspaces and the items they contain can participate in a workspace hierarchy.

Creating a Subworkspace for an Item

Most items can optionally have an associated child workspace, called a **subworkspace**. An item's subworkspace can contain other items. Use the subworkspace of an item to collect other items that have some relationship to that item. For example, you can create a variable that has a subworkspace containing the rules that conclude a new current value for the variable. An item can have only *one* subworkspace.

To create a new subworkspace for an item interactively:

➔ Choose the create subworkspace choice from the item's menu.

This menu choice creates a new workspace and automatically makes it the subworkspace of the selected item. G2 automatically displays the new workspace with its center at the current center of the window.

After the subworkspace of an item exists, the create subworkspace choice no longer appears on the item's menu.

To go to the subworkspace of an item:

➔ Choose go to subworkspace on an item with a subworkspace.

To create a new subworkspace for an item programmatically:

➔ Execute these actions in this order:

create *item*;
create *kb-workspace*;
make *kb-workspace* the subworkspace of *item*

For example:

```
create an item-list L1;
create a kb-workspace W;
make W the subworkspace of L1
```

G2 does not automatically display a new subworkspace that is created programmatically. To display a new subworkspace programmatically, use the show action. This procedure code creates a new item-list, creates a new workspace, and makes the workspace the subworkspace of the item-list:

```
create-list-with-subworkspace()
SUB-WS: class kb-workspace;
IL: class item-list;
begin
    create an item-list IL;
    transfer IL to list-workspace;
    conclude that the names of IL is concerto-item-list;
    make IL permanent;

    create a kb-workspace SUB-WS;
    conclude that the names of SUB-WS is concerto-subworkspace;

    make SUB-WS the subworkspace of IL;
    make SUB-WS permanent;
end
```

Making a Workspace the Subworkspace of an Item

Using the make action, you can make an existing top-level workspace the subworkspace of an item, and you can change the association of a subworkspace from one item to another item. A workspace must be transient before you execute these actions.

To make a workspace the subworkspace of an item:

➔ make *kb-workspace* the subworkspace of *item*

These code examples make workspaces the subworkspaces of items:

```
make top-level-workspace transient;
make top-level-workspace the subworkspace of mineral506;
change the name of top-level-workspace to the symbol mineral506-subworkspace;
make mineral506-subworkspace permanent
```

```
make the subworkspace of item1 transient;
make the subworkspace of item1 the subworkspace of item2;
make the subworkspace of item2 permanent
```

Note The transfer action *does not* change the item association of a subworkspace. However, if the target item already has a subworkspace you can use the transfer or delete actions to remove the item from the target item.

For details, see make and transfer.

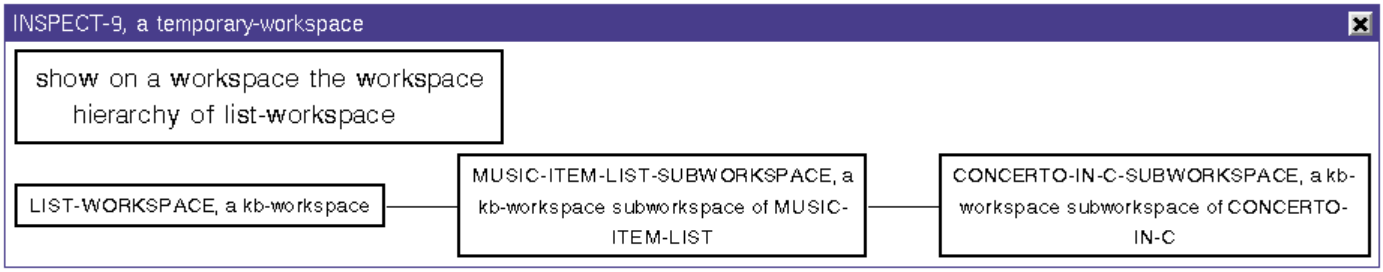
Displaying the Workspace Hierarchy

Each top-level workspace in your KB has a distinct workspace hierarchy. You can use the Inspect facility to view the current workspace hierarchies.

To display the workspace hierarchy:

➔ show on a workspace the workspace hierarchy [of *item*]

This figure shows an example of a workspace hierarchy consisting of one top-level workspace and two subworkspaces:



Determining Whether a Subworkspace Exists

An item has an implicit, system-defined relationship with its subworkspace, which you can determine interactively or programmatically.

To determine whether a subworkspace exists interactively:

➔ Display its menu to see if it includes the go to subworkspace choice.

To determine whether an item has a subworkspace programmatically:

➔ Using the expression the subworkspace of *item* exists, which returns a truth-value.

Referring to Subworkspaces Programmatically

To refer to the subworkspace of an item:

➔ the subworkspace of *item*

To refer to the superior item of a subworkspace:

➔ the superior item of *subworkspace*

Configuring Items Based on the Workspace Hierarchy

You can declare configurations in the item-configuration attributes of a workspace to customize the behavior of items for particular categories of users.

Item configurations declared in one workspace can also pertain to all items below that item in the workspace hierarchy. Thus, the workspace hierarchy can serve as a framework for controlling the behavior of whole regions of KB knowledge. For more information about using item configurations, see Configurations.

Organizing Knowledge in Subworkspaces by Using Connection Posts

You can make the relationship between an item and its subworkspace explicit by using connections and connection posts.

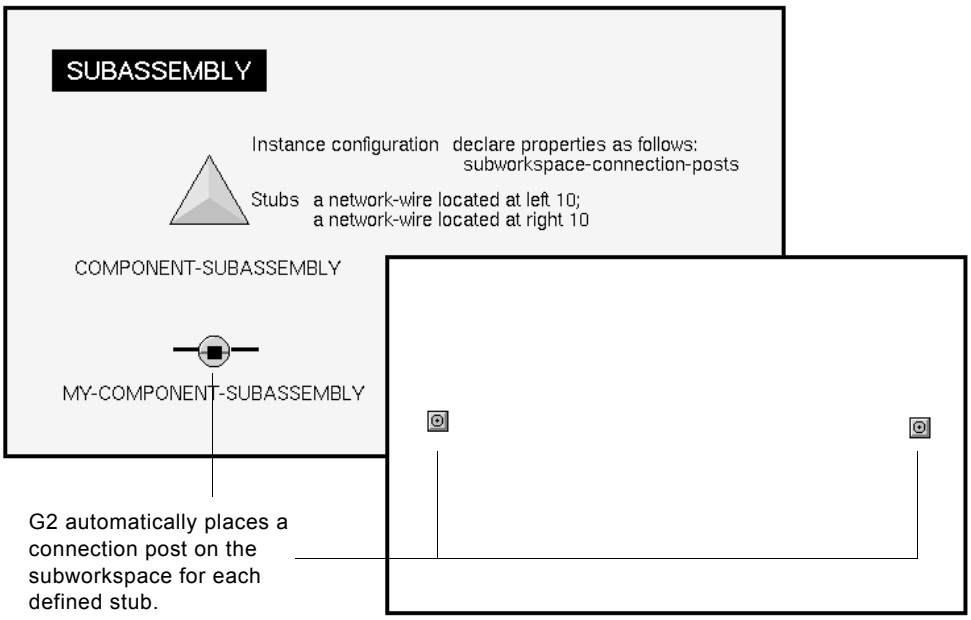
To make the relationship between an item and its subworkspace explicit:

➔ Create a class definition that declares the following instance configuration:

```
declare properties as follows : subworkspace-connection-posts
```

When you create an instance of this user-defined class, the subworkspace of the new instance automatically contains a connection post for each stub defined in the class definition or for each connection that the instance receives from a connection post.

This figure shows an instance of a user-defined class named component-subassembly. This definition declares two stubs for each instance.



The definition also declares this instance configuration:

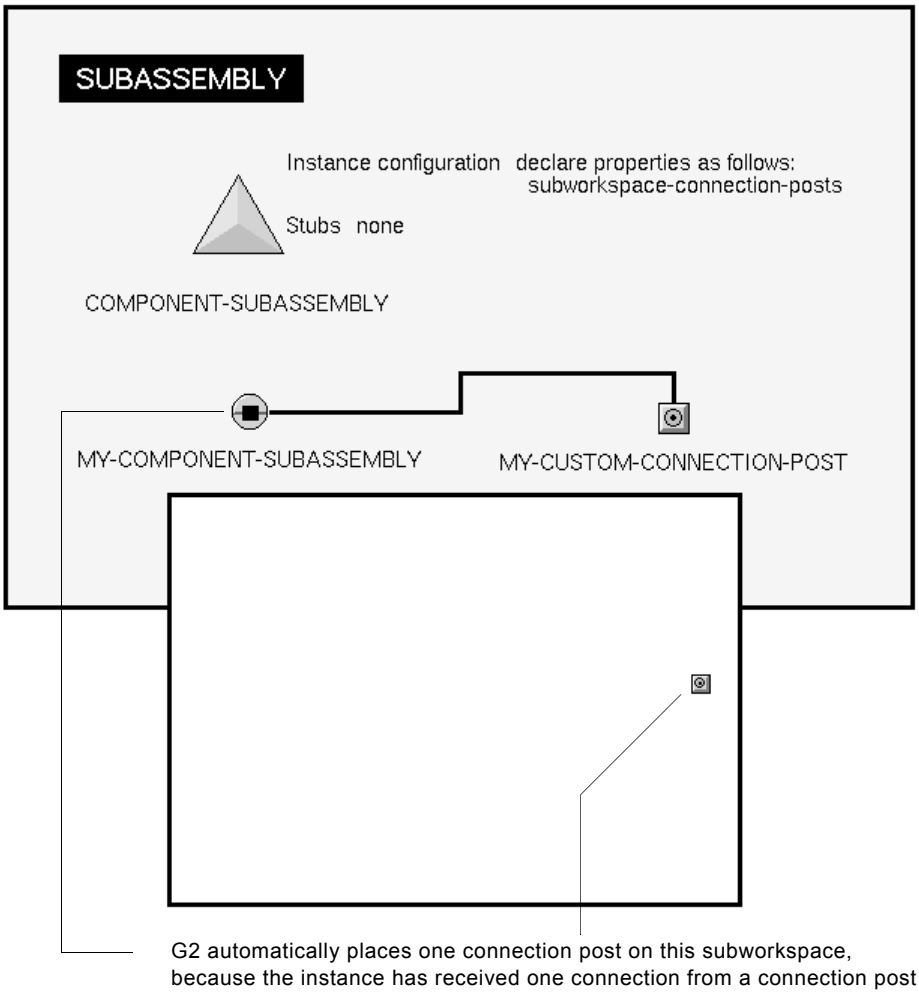
declare properties as follows : subworkspace-connection-posts

As a result, for each instance of this class with a subworkspace, G2 automatically places permanent connection post items on the subworkspace for each declared stub in the definition. G2 also positions the connection posts within the subworkspace relative to the location of the stubs on the instance.

In this example, each connection drawn between a stub on the instance and any connection post is automatically associated with one of the connection posts on the subworkspace of the instance. G2 associates each connection in the superior-connection attribute of the appropriate subworkspace connection post.

If the class definition does not declare stubs, and you interactively create a connection by dragging a stub from a connection post into the instance, G2 automatically creates connection post items on the subworkspace of the instance when you create the subworkspace. G2 locates the connection posts on the subworkspace relative to the location of the connections on the instance.

The following figure illustrates this situation:



The figure shows a new version of the component-subassembly definition that does not declare stubs. After you make a connection between the custom connection post and the instance, and then create a subworkspace for the instance, G2 automatically creates and places a connection post on the subworkspace, and places it relative to the position of the connection on the instance.

Associating Top-Level Workspaces with Modules

By assigning a top-level workspace to a module, you can associate a set of items with a module. Dividing a large KB into modules is the recommended way to organize the knowledge in your KB and to facilitate knowledge reuse.

To learn how to use top-level workspaces to identify the items associated with a module, see Associating Items with a Module.

Activating and Deactivating Workspaces

To *activate* a workspace means to declare the items upon that workspace as available to participate in KB processing. G2 activates enabled workspaces and subworkspaces automatically when you start or restart the current KB, and when you programmatically activate an activatable subworkspace.

The primary effect of activating a workspace is to cause G2 to invoke all initially rules upon them. The invocation of initially rules is described in Activating the Parent Workspace of a Rule.

Activating a workspace also activates all enabled items that reside upon the workspace. The **activation status** of an item determines whether it is active. In general, the activation status of an item propagates from its top-level workspace. For example, if you create an item on an active and enabled workspace, the item and its subworkspace are also active and enabled.

The activation status of an item is distinct from whether it is enabled or disabled, which depends only on whether you have selected the enable and disable choices for the item. By default, a workspace is enabled until you interactively disable it using the disable menu choice. When you disable an item, the workspaces in the hierarchy below the item are no longer active.

Activating Top-Level Workspaces

Each time you start or restart the current KB, G2 automatically does the following:

1. Activates each enabled top-level workspace.
2. Propagates the activation status of each top-level workspace to each item below it in its own workspace hierarchy.

Note After the current KB has started or restarted, when a top-level workspace becomes enabled, all enabled items below it in its own workspace hierarchy also become activated.

All types of definitions (for example, class definitions and relation definitions) remain in effect regardless of their activation status and regardless of whether they are enabled or disabled. This means that you can instantiate definitions that are inactive or disabled.

Executable items, for example, rules and procedures, must be enabled and activated to be eligible to be invoked.

Activating and Deactivating a Subworkspace

Many system-defined items are capable of having a subworkspace, as described in Creating a Subworkspace for an Item. Subworkspaces inherit the activation status from the top-level workspace. If both the workspace and the item for which you create a subworkspace are both enabled and active, the subworkspace of the item is also enabled and active.

An **activatable subworkspace** is the subworkspace of an item whose parent item has been configured using this configuration statement:

declare properties as follows : activatable-subworkspace

You specify this statement in an item-configuration or instance-configuration attribute, as described in Configurations.

An activatable subworkspace does not inherit its activation status from the top-level workspace. Instead, you must activate an activatable subworkspace programmatically, using the activate and deactivate actions.

Note When you deactivate the subworkspace of an item, G2 behaves as though the items upon the subworkspace do not exist. All items upon the subworkspace are no longer active. The subworkspace itself, however, can still be referenced and is included in existence checks such as the count of each kb-workspace.

Activating and deactivating activatable subworkspaces programmatically provides a technique for enabling and disabling portions of a KB. By activating and deactivating appropriate portions of the workspace hierarchy, you can implement modes in your application, activating those branches which are relevant to the current mode while deactivating branches used to model competing modes.

How Activating and Deactivating Affects Items

When you first create an activatable subworkspace, it is active. Subsequently resetting or restarting the KB renders the subworkspace deactivated and it must then be activated programmatically.

When you activate an activatable subworkspace, G2 invokes all initially rules upon that subworkspace, and resets variables and parameters that have initial values to those values. Default attribute values changed since instantiation are not reset.

When you deactivate an activatable subworkspace, G2 ignores the non-definition items that it and its subworkspaces contain, until that subworkspace is again activated. Variables and parameters are reset to their initial values. Deactivating a subworkspace also deactivates all of its subworkspaces automatically.

You activate and deactivate activatable subworkspaces programmatically by using the activate and deactivate actions. For a description of these actions, see activate and deactivate.

Printing a Workspace

G2 can produce a print file that contains the image of a workspace. G2 only supports PostScript print files.

Note When connecting to G2 through Telewindows, you can print directly to a native printer from the client. For details, see the *Telewindows User's Guide*.

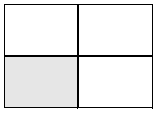
To print a workspace:

➔ Select KB Workspace > Print.

G2 produces a print file subject to the current settings in the installed Printer Setup system table described in Printer Setup.

Printing Multiple Pages

When the print output for a workspace extends across multiple pages, each printed page includes a page index, indicating which part of the whole print job is the current page. For instance, if you are printing page 3 of 4 pages, the page index marker looks like this:



The page index shown here is for illustrative purposes only and is several times larger than what actually appears in the lower-left hand corner of the print output. The page index appears beyond the print area and does not affect the print image.

Generating Encapsulated PostScript Files

To generate an encapsulated PostScript file for printing a workspace:

➔ Configure the printing-file-format attribute of the installed Printer Setup system table to be encapsulated postscript.

Since the encapsulated PostScript convention requires that an image take up only one page, choosing this file format causes G2 to scale the image of the workspace to fit onto a single sheet of paper.

Tip An EPS print file contains both a graphics image and information about the height and width of that image. For this reason, you can import the image in an EPS print file into another document.

However, if you print a workspace whose image must span more than one physical page, based on the current settings in the page-layout attribute, then G2 writes that file as a standard, not encapsulated, PostScript print file. The image contained in such a print file cannot, by definition, conform to the requirements for encapsulated PostScript.

Generating JPEG Files

To generate an JPEG picture file for printing a workspace:

➔ Configure the printing-file-format attribute of the installed Printer Setup system table to be jpeg.

Choosing this file format causes G2 to ignore all page settings and generate a JPEG picture with the same width and height of the workspace.

Printing a Workspace on a Color PostScript Printer

To produce a PostScript print file that prints on a color PostScript printer:

➔ Configure the color conversion detail to full-color in the printing-details attribute of the installed Printer Setup system table.

The next time you print a workspace, G2 creates a PostScript print file that contains the appropriate color information.

Note The image-palette attribute in the Color Parameters system table does *not* affect whether printed output appears in color, black-and-white, or gray-scale.

Printing Workspaces without Borders

The page-economy-mode attribute in the Printer Setup system table allows you to print workspaces without borders. When this attribute is set to yes, G2 prints the workspace without borders, unless there is a frame style defined for the workspace. Also, G2 does not print blank pages and suppresses the multipage indicator. Use this option to save paper when printing workspaces.

For details, see Printer Setup.

Using Double Buffering

Workspaces support "double buffering," which means G2 and Telewindows first draw intermediate display updates to an offscreen bitmap, then copy the final bitmap contents to the screen. This technique can reduce flickering when updating workspaces.

To support this feature, workspaces provide the prefer-buffered-drawing attribute, with values yes or no. If the value is yes, then G2 and Telewindows try to use the "double buffering" approach to rendering images, whenever possible.

Setting the Color of Workspaces

Workspaces have two color attributes, foreground-color and background-color, where:

- The **foreground color** determines the color of items on this workspace that do not specify a local color. The default foreground color for a workspace is the color black.
- The **background color** is the color in which the background of the workspace appears. The default background color for a workspace is the color white.

The foreground color of a workspace also determines the value of the metacolor foreground for an item upon the workspace. Typically, the color setting for attribute displays of items, and for the text and border of items with a text box, such as rules, is foreground.

The background color of a workspace also determines the value of the metacolor transparent for any item upon the workspace. Typically, the background color setting for items with a text box, such as rules, is set to transparent.

To set the workspace color interactively:

➔ Select KB Workspace > Color > background-color | foreground-color > *color*.

To set the workspace color programmatically:

➔ change the *color-attribute-name* of *kb-workspace* to
{*color-name* | *symbolic-expression*}

For example:

```
change the background-color of my-workspace to salmon
```

You can also provide a symbol of the form RGB*rrggbb* as a valid color name, where *rr*, *gg*, *bb*, are the 8-bit hex values for red, green, and blue. For details, see Other Literal Terms.

Creating Custom Workspace Borders

You can create custom borders for workspaces by using a frame-style-definition. The definition specifies the color of the border and its thickness in workspace units. A workspace with a frame-style definition does not have a title bar.

Note The default borders of a kb-workspace, as well as borders created by using a frame-style-definition are *not* included in the item-width and item-height of a kb-workspace.

To create a frame-style definition:

➔ Choose KB Workspace > New Definition > frame-style-definition.

Associate a frame-style definition with a workspace by entering its name in the frame-style attribute for the workspace.

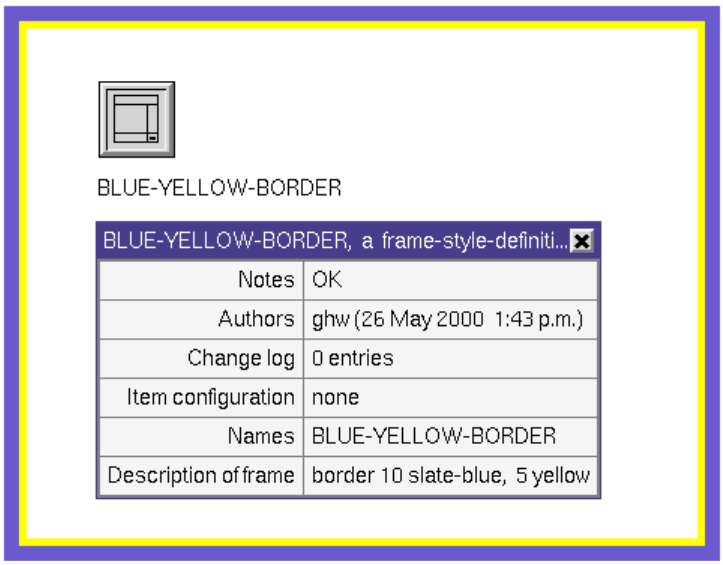
A frame-style definition has one class-specific attribute, description-of-frame, in which you enter one or more clauses that define the display characteristics of the workspace border. Use a semicolon to separate clauses in a description-of-frame attribute.

For example, to declare a border with two stripes, enter a statement like this:

```
border 10 gold , 5 forest-green
```

The statement must specify an integer value or expression, which represents the thickness in workspace units of one border section. Enter a system-defined color name, or select a color name from the color menu. Use a comma to separate any two border stripe descriptions. You can specify more than one border clause.

The next figure shows a workspace whose frame-style definition has a border with two differently colored sections:



The first color specified in the first border clause refers to the *outermost* stripe in the border.

Note When you change the scale of a workspace, G2 does *not* scale the borders defined by a frame-style definition; G2 only resizes them. As you scale the workspace, G2 redraws the border to fit around the workspace, but the thickness in workspace units of the border does not change.

When a workspace is not selected, the frame-style turns gray.

Using a Graphic as a Background Image

You can specify that a workspace display a graphics image as its background. To reference the image, enter the name of an image-definition in the background-images attribute for the workspace. Image definitions support `.jpeg`, `.gif`, and `.xmb` file types. For more information about image definitions, see External Images.

The image definition must refer to a file that contains the bitmap graphics data. The image definition bitmap itself can be up to 65,536 by 65,536 pixels in size.

For example, to include the image referenced in the image definition named world-map, enter the following statement in the background-images attribute:

```
world-map at (10,10)
```

In this statement, the x and y coordinates direct G2 to place the center of the image 10 workspace units above and 10 workspace units to the right of the workspace origin.

Various GIFs that can be used as workspace background images are available in the G2 `demos` directory, as described under GIF Files.

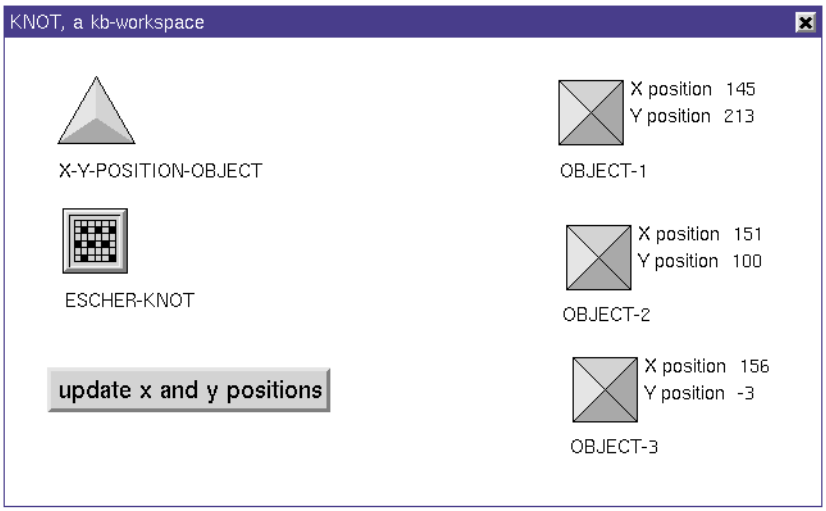
Specifying the Center of the Background Image

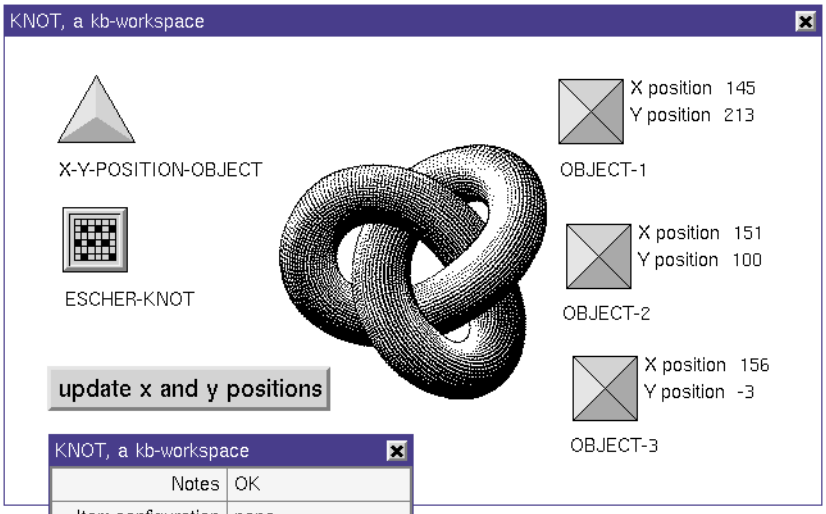
The x, y coordinates in the background-images attribute identify where G2 places the center of the image. G2 positions the center of the image with respect to the center of the workspace's extent; it does *not* position the image with respect to the workspace origin.

If you do not specify x, y coordinates in the background-images attribute:

- G2 places the center of the background image at the center of the workspace's displayed extent.
- G2 automatically updates the background-images attribute to include the x, y coordinates within the workspace's extent where the image's center was placed.

For example, the figure on the next page shows two versions of a workspace that contains six items: *before* and *after* including a background image.





KNOT, a kb-workspace

Notes	OK
Item configuration	none
Names	KNOT
Workspace margin	30
Background color	white
Foreground color	black
Background images	escher-knot at (6,88)
Frame style	none
Module assignment	unspecified

The three items on the right half of each workspace display their respective item-x-positions and item-y-positions in attribute displays. The attribute displays indicate that the workspace origin is not within its extent.

After entering the name of an image definition in the workspace background-images attribute, with no x, y coordinates included, G2 changes this workspace by automatically:

- Placing the center of the image at the center of the workspace's displayed extent.
- Expanding the workspace's extent, as necessary, to allow the entire image to display, while allowing for the margins.
- Updating the background-images attribute to include the x, y coordinates for the location of the image's center.

Using Tiled Workspace Backgrounds

You can use tiled images as the background of a workspace by configuring the background-images attribute of a workspace, using this syntax:

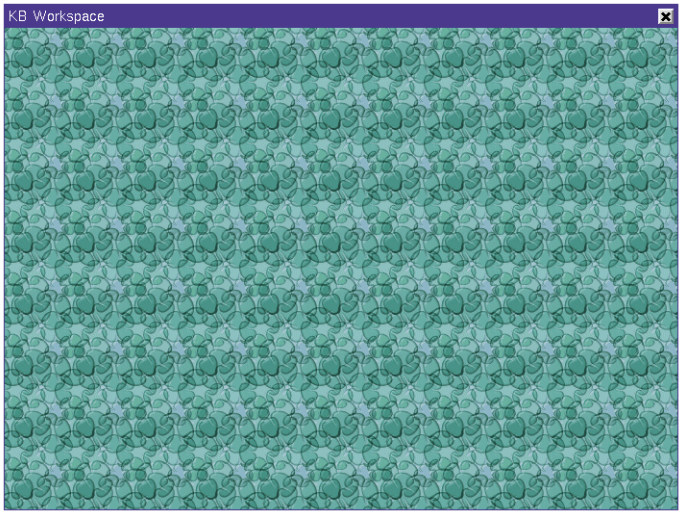
image-name tiled [at (x,y)]

where:

image-name is the name of an image-definition object.

By default, the image is tiled at the center of the workspace, at (0, 0). You can also specify the x, y coordinates at which to tile the image.

Here is an example of a tiled workspace background:



Displaying More Than One Background Image

You can display more than one image in the background of a workspace. To do this, enter a statement in the background-images attribute, such as:

```
world-map at (10,10), map-legend at (100,100)
```

Note If you specify more than one image definition in the background-images attribute, and if the extents of the images overlap, G2 displays image definitions at the end of the list on top of those at the beginning of the list.

Saving the Background Image in the KB

You can save the graphics data that comprise the background image when you save your KB file. To do so, specify yes as the value of the save-image-data-with-kb attribute of the image definition referenced in the background-images attribute. Doing so prevents you from inadvertently separating the image data from your KB when you move the KB to another system, but doing so also increases the size of your KB file when next saved.

Other Considerations for Using Background Images

As you work with background images for your workspaces, keep these considerations in mind:

- G2 always displays the entire image stored in the image file. When you add or change the background image of a workspace, G2 automatically enlarges the workspace so that the bitmap graphics image fits within it.
- If the bitmap graphics image does not fill the workspace extent, the remainder of the workspace extent appears in the background color.
- When shrink-wrapping a workspace, G2 does not hide or crop any portion of the background image.
- After you add a new reference to an image definition in the background-images attribute, and if that image is in color, then the first time G2 displays the image, its colors might not appear in the colors you expect. If so, set the image-palette attribute in the Drawing Parameters system table to extended-colors, rather than standard-colors.
- If your KB contains a workspace that uses a background image that is not saved in your KB file, that KB is inherently incomplete. Thus, when you load the KB on another computer, the bitmap graphics data file(s) to which the background image refers must accompany the KB on the new computer.

Be aware that G2 reads the bitmap graphics file referenced in an image definition for a background image in only three situations:

- When G2 loads the KB, if the image's graphics data are not already saved as part of the KB itself.
- When you finish editing the name of an image definition in the background-images attribute of the workspace.
- When the KB invokes the g2-refresh-image-definition system procedure.

The Kb-Workspace Class

A KB workspace is an item of the system-defined kb-workspace class.

A workspace has its own unique representation which depends on the window style defined for your interaction with G2. Its appearance is not iconic.

The following table summarizes the class-specific attributes of the kb-workspace class:

Attribute	Description
workspace-margin	Distance in workspace units between the outermost items on the workspace and the innermost stripe of the workspace border.
Allowable values:	Any integer, zero (0) or greater
Default value:	30 workspace units
Notes:	See Specifying Margins within the Border of a Workspace.
background-color	The background color of the workspace
Allowable values:	Any available color symbol.
Default value:	white
foreground-color	The foreground color of the workspace
Allowable values:	Any available color symbol.
Default value:	black

Attribute	Description
background-images	Names of one or more image definitions, each of which specifies a graphics image that appears as the workspace background.
Allowable values:	none Name of any image definition item
Default value:	none
Notes:	See Using a Graphic as a Background Image.
frame-style	Name of a frame-style definition, which determines a reusable, custom border for this workspace.
Allowable values:	none Name of any frame-style definition
Default value:	none
Notes:	See Creating Custom Workspace Borders.
title-bar-text	The text to display in the title bar, which can be a string, with quotes, or as an expression to display the workspace name, class, or table header.
Allowable values:	default | *string* | the class | the table header | the name [if any, otherwise, *string* | the class | the table header]
Default value:	default
Notes:	See Editing Title Bar Text.
view-preferences	Controls the display behavior when programmatically showing workspaces in the server and client, using the show action.
Allowable values:	none | fixed size | unselectable
Default value:	none
Notes:	See Using View-Preferences.

Attribute	Description
prefer-buffered-drawing	Determines whether the workspace uses double-buffering to reduce flickering.
Allowable values:	yes no
Default value:	no
Notes:	See Using Double Buffering.
module-assignment	Name of the parent module of this top-level workspace.
Allowable values:	unspecified Name of any module in the current KB
Default value:	unspecified
Notes:	See Associating Top-Level Workspaces with Modules.

Using View-Preferences

The view-preferences attribute has these options:

- none, the default, which allows the workspace to be selected in the server and the window to be resized in the client.
- unselectable, which prevents the workspace from ever being selected, either programmatically or interactively, or from affecting the current selection in the server.

Note By making a workspace unselectable, you are preventing menu bar operations on the workspace in the Telewindows client. Making the workspace unselectable does not affect the ability to interactively select the workspace by clicking its title bar.

- fixed size, which prevents the window containing the workspace from changing size when programmatically showing it in the client. Instead, if the workspace size becomes larger than the window, the window displays scroll bars. Note that the user can still resize the window interactively.

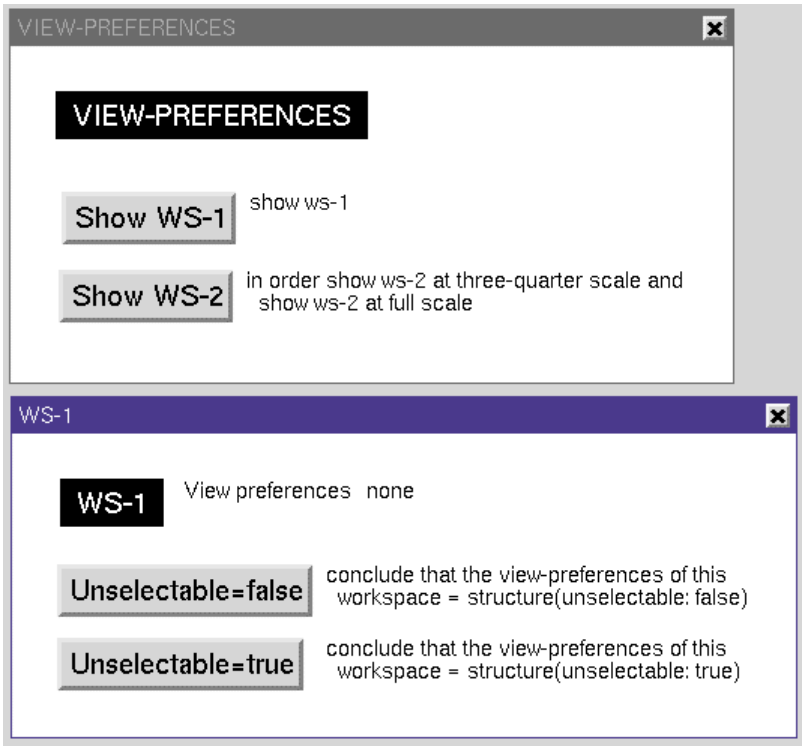
You can specify none, or any combination of unselectable and/or fixed size. When concluding a value for this attribute programmatically, you conclude the value as a structure with this syntax:

structure (unselectable: *truth-value*, fixed-size: *truth-value*)

The view preferences are applied to the view created when a workspace is shown.

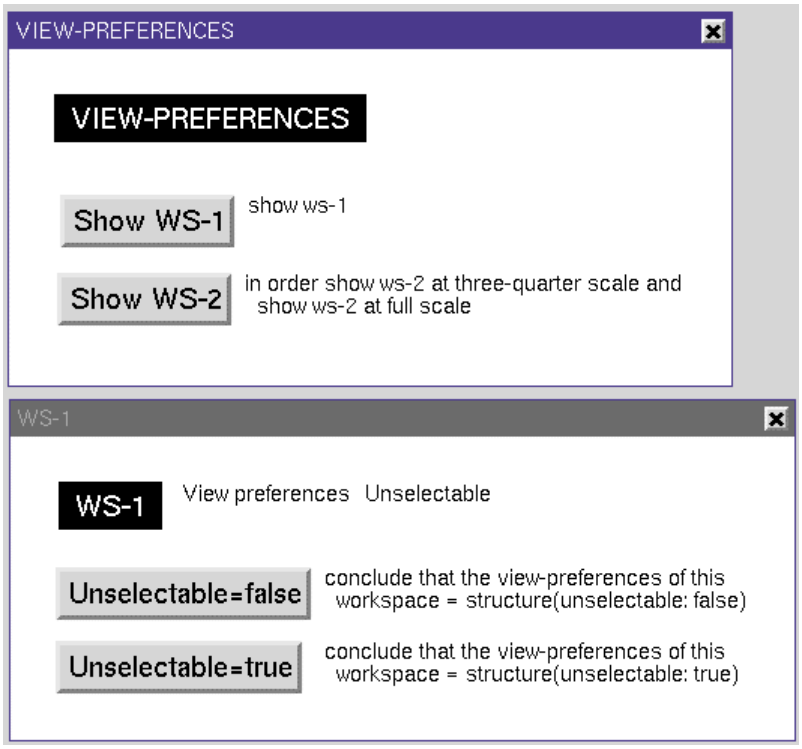
Example: Setting View-Preferences to Unselectable

In this example, ws-1 sets the view-preferences to none, the default, by setting unselectable to false. Clicking the Show WS-1 button shows ws-1, which also selects it because unselectable is false. This figure shows the workspaces in the G2 server.



WS-1 is selected.

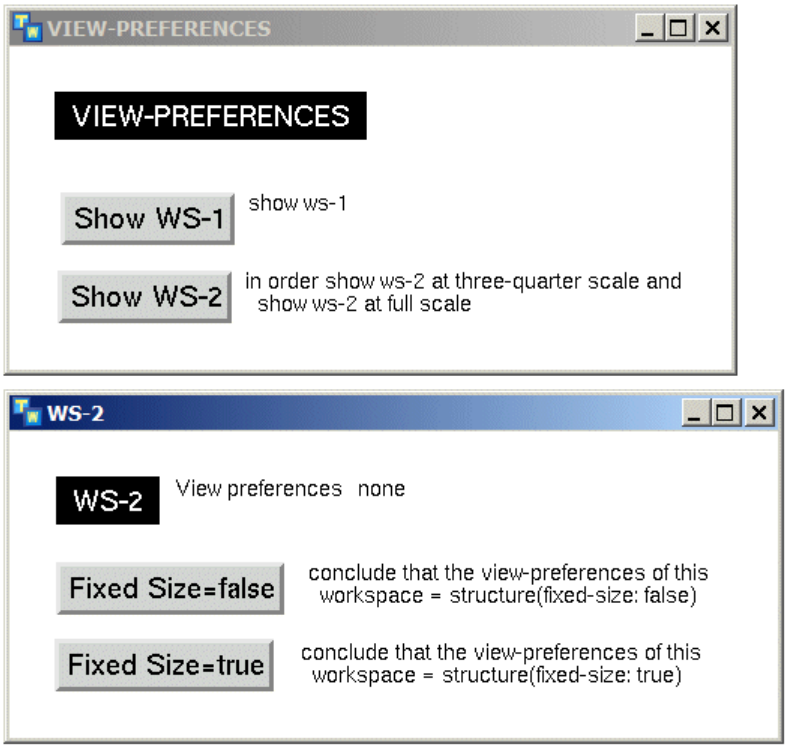
Now, ws-1 sets the view-preferences to unselectable by concluding that unselectable is true. Clicking the Show WS-1 button shows ws-1, but it does not select the workspace because unselectable is true. This figure shows the workspaces in the server.



WS-1 is not selected.

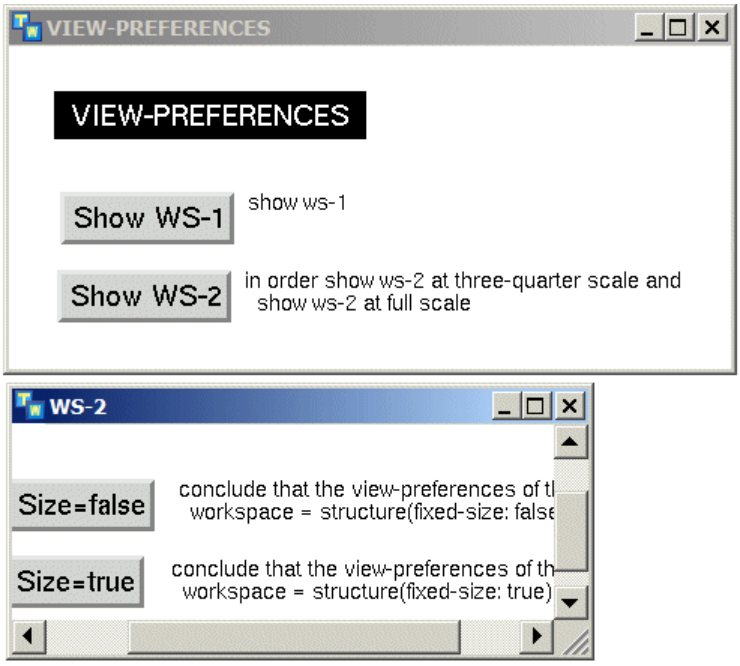
Example: Setting View-Preferences to Fixed Size

This figure shows the workspaces in the Telewindows client. In this example, ws-2 sets the view-preferences to none, the default. Clicking the Show WS-2 button shows ws-2 at three-quarter scale, then at full scale again. The window containing the workspace resizes to fit the workspace each time the workspace is scaled, because fixed-size is false.



The window containing WS-2 resizes each time the workspace is scaled.

Now, ws-1 sets the view-preferences to fixed size by concluding that fixed-size is true. Clicking the Show WS-2 button shows ws-2 at three-quarter scale, then at full scale again. However, this time, the window containing the workspace does not resize to fit the workspace, because fixed-size is true. This figure shows the workspaces in the Telewindows client.



The window containing WS-2 remains a fixed size when the workspace is scaled.

Actions That Apply to KB Workspaces

You can use the show and hide actions to control workspaces programmatically. For more information about these actions, see hide, and show.

Expressions That Refer to KB Workspaces

You can use the following expressions for KB workspaces.

To refer to whether a workspace has been activated:

➔ *kb-workspace* has [not] been activated
-> *truth-value*

The expression produces a truth-value that indicates whether the specified workspace is activated. For example:

```
for any help-button B
    if the subworkspace W of B exists and W has been activated
        then start evaluate-status-of(B)
```

To refer to the workspace of an item:

➔ the workspace [*local-name*] of *item*
-> *kb-workspace*

For example, this expression brings the parent workspace of the specified item to the top of the display hierarchy:

```
hide the workspace of pump-1
```

If the specified item does not have a parent workspace, evaluating this expression causes G2 to signal an error. To prevent this, use this expression with the exists expression, as follows:

```
for any item X
    if the workspace of X exists and the name of X is CUSTOM then
        conclude that the status of X is OK
```

This generic if rule checks each item that is upon a workspace, and for each such item whose name is custom, sets its status attribute to the symbol ok.

To refer to the subworkspace of an item:

➔ the subworkspace [*local-name*] of *item*
-> *kb-workspace*

This expression produces the workspace that is the subworkspace of the specified item. For example:

```
show the subworkspace of pump-1
```

If the specified item does not have a subworkspace, evaluating this expression causes G2 to signal an error. To prevent this, use this expression with the exists expression, as follows:

```
for any custom-object O
    if the subworkspace of O exists and the name of O is custom
        then conclude that the status of O is OK
```

This generic if rule identifies each custom-object that has a subworkspace and, for each that does and whose name also is custom, sets its status attribute to the symbol ok.

To refer to an item upon a particular workspace:

➔ the *class-name* [*local-name*] upon *kb-workspace*
-> *item*

With the the quantifier, this generic reference expression produces the one and only item of the specified class that resides upon the specified workspace. With the any quantifier, this expression produces the set of items of the specified class that reside upon the specified workspace. For example:

```
move the help-button upon this workspace by (100,100)
```

5

Modularized KBs

Describes how to partition your KB into modules.

Introduction

You can develop a large knowledge base (KB) from smaller, more manageable pieces called **modules**. Each module contains a set of related items that together comprise a KB. You define a module in the Module Information system table. We recommend that you design and implement your application to use modules.

One module can directly require another. For example, a module that contains instances of user-defined classes would directly require modules that contain the definitions for those classes. When you load a KB file that contains a module, G2 automatically loads all required modules.

Developers on an application team can develop modules more or less independently of one another. Also, you can design your modules so that you can use them to build more than one application.

By organizing items into modules, you can:

- Store items associated with each module into a separate KB.
- Add or change knowledge in each module independently of the other modules.
- Reload the complete KB by loading all required modules.
- Merge modules from one KB into a different KB.

You can work with modules both interactively and programmatically.

Modules affect many aspects of a G2 application, and information about them appears in various places in the G2 documentation. This chapter describes the essential techniques for using modules and module hierarchies. Additional information appears as follows:

- The *Getting Started with G2 Tutorials* introduce modules and provide exercises that show you how to use them.
- The *G2 Developer's Guide* describes techniques and guidelines for using modules and module hierarchies effectively in complex situations.
- The *G2 Foundation Resources User's Guide* describes additional capabilities for module management that are available in GFR.

Understanding Modules

A module identifies a set of items that represents a component of a larger KB.

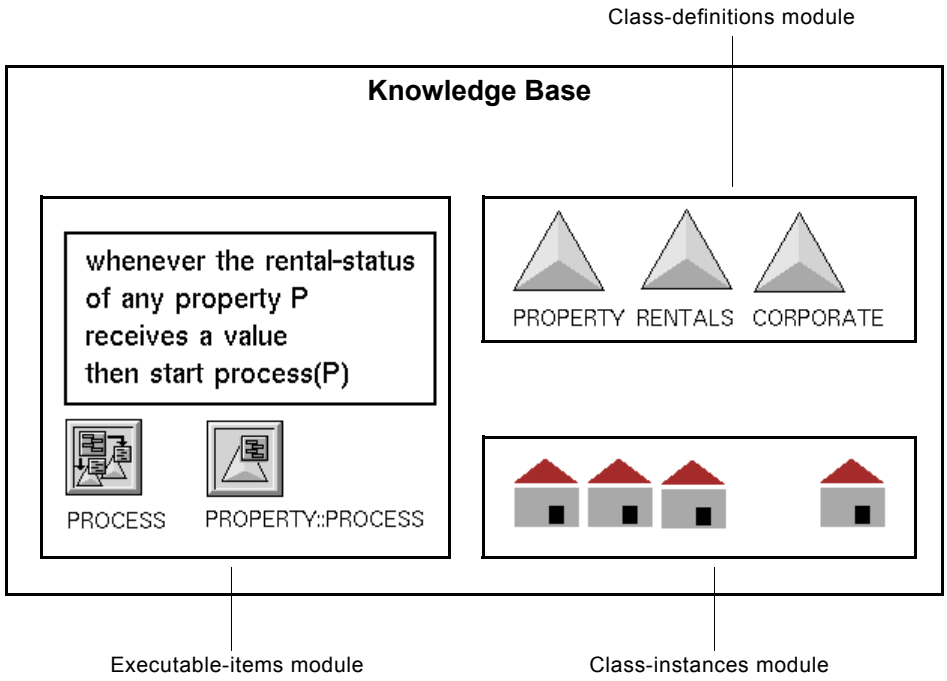
You can work with modules in a flexible manner, as follows:

1. Start by building an application without modules.
2. Create modules and associate the items in the KB with those modules
3. Save the KB into separate KB files, with one module per file.

For details about how to do this, see Creating, Populating, and Saving Modules.

You might begin to build an application by populating an empty KB, organizing the knowledge that pertains to certain classes of items into different modules.

For example, you could define a module for class definitions, define another module for instances of the classes, and define a third module for executable items that manipulate class instances, as this figure shows:



You can create a module and associate it with a set of items, regardless of whether the KB is reset, running, or paused.

Once you have modularized your KB and saved these modules into separate KB files, you can load individual modules, and merge other modules into the current KB. For information about loading and merging KBs, see Working with Modularized KBs.

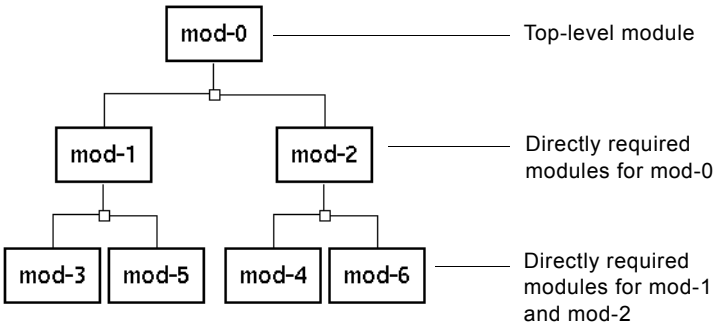
For large applications, you typically create a **module hierarchy** by defining modules that directly require other modules. This is explained next.

The Module Hierarchy

To create a module hierarchy, you must have a **top-level module**, which is the root of the module hierarchy. To form the hierarchy, you define modules in the hierarchy to **directly require** one or more other modules.

Defining a module to directly require another module means that G2 automatically loads the directly required module before loading the module that depends on it. In this manner, you can load an entire application by loading a single top-level module.

The following diagram illustrates the relationships among the modules in a module hierarchy:



A module that directly requires another module can function independently from its directly required modules. However, you *must* define a module to directly require another module when:

- A class definition contained in one module has the definition of one or more superior classes contained in another module.
- An item contained in one module is an instance of a class definition contained in another module.

In the figure above, for example, mod-0 might contain items that are instances of classes defined in definitions assigned to mod-1. In this case, you must define mod-0 to directly require mod-1.

A KB that includes a module hierarchy is called a **modularized KB**. When saving a modularized KB, you save the modules into separate files. If a KB is not consistently modularized according to the criteria outlined in Rules for Consistent Modularization, it is considered **unmodularized**. G2 saves unmodularized KB modules into a single KB file.

For more information about how to create and save a module hierarchy, see Creating a Module Hierarchy.

Modules and System Tables

Each module that you create or load has its own set of system tables. You define each module in the Module Information system table, including its name and its directly required modules.

When you load a KB, G2 installs a set of system tables for each module contained in the KB. After loading is complete, the set of system tables associated with the top-level module defines much of the functionality for all the modules in the current KB.

If you create a new module, G2 automatically creates a new set of system tables and associates them with the new module. If you delete a module from the current KB, G2 automatically deletes its associated system tables. When you save a module, G2 also saves its associated system tables in the KB file.

There is one system table, the Server Parameters system table, which is not associated with any module. Only one Server Parameters system table exists in a G2 process. It is created by G2's initialization process when you first launch G2 and remains in residence throughout the G2 process, even when you clear and load KBs. You use this system table to specify preferences that pertain to your G2 process independent of the resident KB. For more details, see Server Parameters.

Modules and Items

You associate items in a KB with a module by assigning that module to one or more top-level workspaces in the KB. This causes that workspace, all items upon that workspace, and all items below them in the workspace hierarchy to be associated with that module.

For information on assigning items to a module, see Associating Items with a Module.

Creating, Populating, and Saving Modules

The basic tasks for working with modules are:

- Naming the top-level module, which creates a new empty module.
- Associating items with the module.
- Saving the module in a KB file.

For information about creating and saving a hierarchy of modules, see Creating a Module Hierarchy.

Naming Conventions for Modules

When naming the top level or other modules, we recommend following the standard naming conventions described in Platform File Systems and KB File Names.

Naming the Top-Level Module

The first step in creating a module hierarchy is to name the top-level module. This is true whether you are creating modules in an empty KB, or in a KB that already contains items.

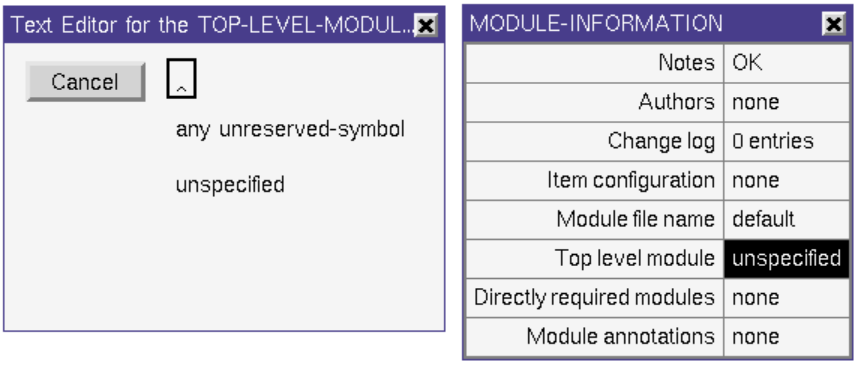
Naming the top-level module creates a module of that name in the current KB. If you want the current KB to have a single module only, all you have to do to create the module is to name it and assign all top-level workspaces to it.

You can name the top-level module interactively or programmatically.

To name the top-level module interactively:

1 Open the Module Information system table by choosing Main Menu > System Tables > Module Information.

2 Enter the name of the top-level module in the top-level-module attribute:



You can enter any unreserved symbol as the name of the top-level module.

To name the top-level module programmatically:

➔ conclude that the top-level-module of module-information
= the symbol *module-name*

where *module-name* is the symbolic name of the module.

If you have named a top-level module in the current KB, each time you create a new top-level workspace, G2 automatically assigns the workspace to the top-level module.

Note If you delete a module from the current KB, you can optionally delete all the workspaces associated with that module. For more information, see Deleting a Module.

Associating Items with a Module

After creating a module, you typically associate the module with a set of items in the current KB. You do this by associating the module with one or more top-level KB workspaces. G2 associates the module with these workspaces and with all items below the workspaces in the workspace hierarchies.

To associate a module with a top-level workspace interactively:

1 Open the table for a top-level workspace.

2 Enter the name of a module in the module-assignment attribute.

You can enter only one module name in this attribute.

To associate a module with a top-level workspace programmatically:

➔ conclude that the module-assignment of *kb-workspace*
is *module-name*

where *module-name* is the name of the module to which you wish to assign *kb-workspace*.

Tip You can assign more than one top-level KB workspace to the same module. However, we recommend that you assign only one top-level workspace to a module. By observing this convention, your modularized KBs have a predictable structure, and are more convenient to work with.

If the current KB has a top-level module, and you add a new top-level workspace to the current KB, G2 automatically sets its module-assignment attribute to the name of the top-level module.

Saving a Module in a Separate KB File

When your KB contains a single top-level module, you can save the module in its own KB file. For information about saving a modularized KB, see Saving the Module Hierarchy.

You can save modules interactively or programmatically.

Note G2 does not accept the wildcard characters *, ?, {, and } in filenames, file extensions, or version numbers. They are allowed in pathnames. The following filenames would not be accepted in the editor and would generate an error when given to a system procedure: `mod*.kb`, `mod.k*b`, `*.kb`, and `*.*`. This pathname is accepted on UNIX: `/home/user/*/mod.kb`.

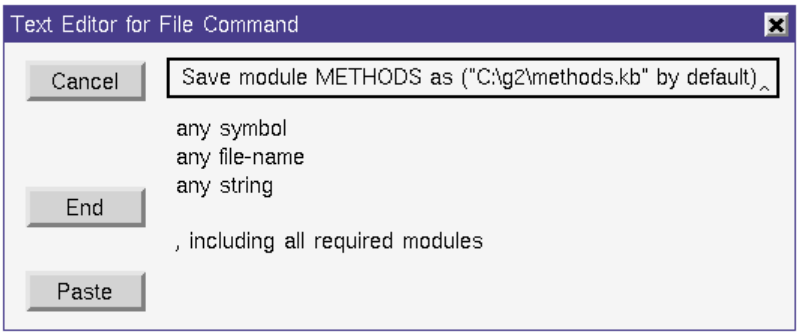
To save a module in a file interactively:

1 Select Main Menu > Save KB.

G2 displays a special workspace for saving KBs, which shows:

- The name of the top-level module to save.
- The default directory path in which it will save the KB file.
- The default KB filename, which is based on the name of the top-level module.

The following figure shows the save KB workspace that G2 displays when saving the methods module:



2 At this point you can:

- Enter a new default directory path and filename of the KB file into which you want to save the module.
- Enter just a new filename.
- Accept the default filename.

Tip We recommend that you name the KB file using the same name as the module it contains. This is especially critical when saving modularized KB, as explained in Specifying the Filename of a Saved Module.

To save a module programmatically:

➔ Use the g2-save-module system procedure, as described in KB and Module Operations in the *G2 System Procedures Reference Manual*.

You can programmatically save a module, even while the current KB is running.

Creating a Module Hierarchy

Typically, for large applications, you will want to create a **module hierarchy** to organize your KB.

In general, a module hierarchy consists of one top-level module and multiple directly required modules below the top-level module, where each submodule can also directly require one or more modules.

For information on when one module *must* directly require another module, see The Module Hierarchy.

The general steps for creating a module hierarchy are:

1. Create a top-level module.
2. Create one or more additional modules.
3. Declare the directly required modules for each module in the KB.
4. Check for consistent modularization.
5. Save the modularized KB into separate KB files.

The following sections outline these steps in detail.

Creating a Top-Level Module

The first step for creating a module hierarchy is to name the top-level module. This is described in Naming the Top-Level Module.

Naming the top-level module creates a single top-level module for the KB. The next step is to create additional modules in the current KB.

Creating a New Module

To create a module hierarchy, you must create additional modules in the current KB. Once you have created these modules, you can declared them to be directly required by the top-level module, as well as by other modules in the hierarchy.

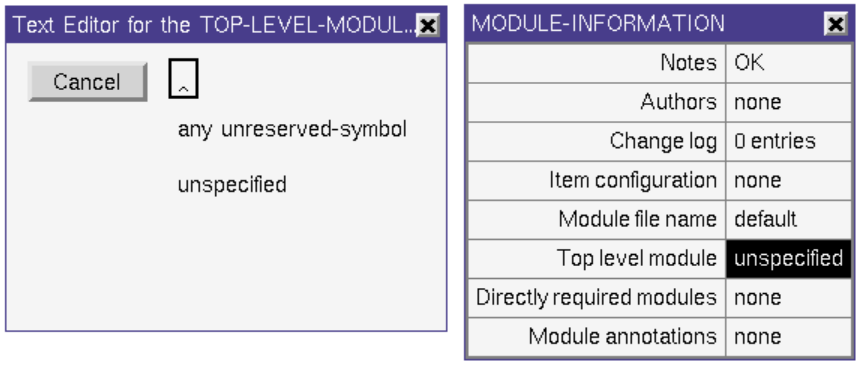
Creating a New Module Interactively

You can create a new module interactively or programmatically.

To create a new module interactively:

1 Select Main Menu > Miscellany > Create New Module.

G2 displays the Module Information system table for the new module:



2 Enter the name of the new module in the top-level-module attribute of the new Module Information system table.

Creating a New Module Programmatically

G2 includes a system procedure for creating a module programmatically.

To create a new module programmatically:

➔ g2-create-module
(*module-name*: symbol)

where *module-name* is the name of the module in your current KB that you wish to create programmatically.

The *module-name* cannot:

- Duplicate the name of an existing module.
- Be a reserved word in G2.
- Be the symbol unspecified.

On successful execution, g2-create-module creates a set of system tables for the new module. The top-level-module attribute of the Module Information system table is *module-name*. All other attributes have default values.

The next procedure creates a new module based on the symbolic name passed as its argument, and assigns the kb-workspace to the new module:

```
create-module(module-name: symbol, ws: class kb-workspace)
begin
    call g2-create-module(module-name);
    conclude that the module-assignment of ws = module-name
end
```

System Tables Associated with a New Module

When you create a new non-top-level module interactively or programmatically, G2 creates a set of twenty associated system tables for the new module. However, these system tables are not installed in the current KB; with the exception of some module-specific attributes, only the system tables of the top-level module are in effect for the current KB.

To display the system tables for all loaded modules:

➔ Choose Main Menu > Inspect and enter this command:

show on a workspace every system-table

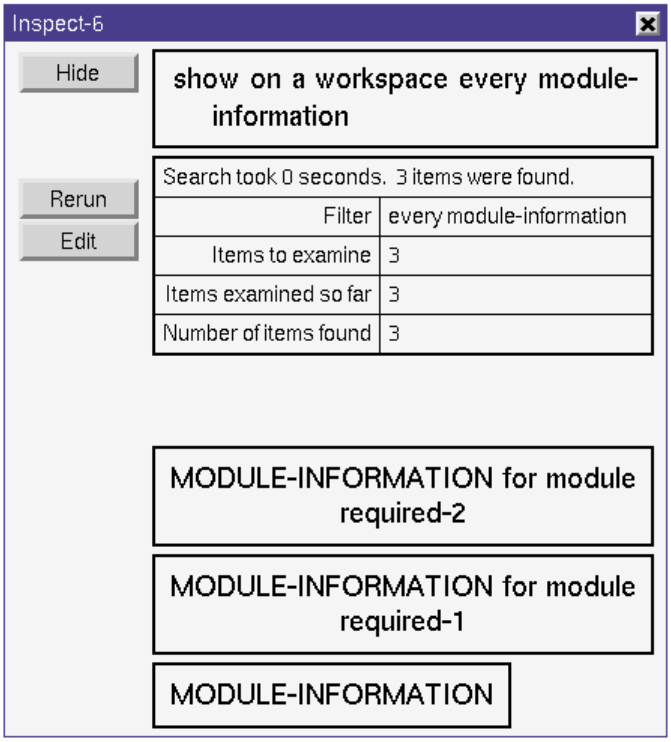
To display a particular system table subclass for all loaded modules:

➔ Choose Main Menu > Inspect and enter this command:

show on a workspace every *system-table-subclass*

For example, the next illustration shows an Inspect workspace for show on a workspace every module-information. The KB has three defined modules: a top-level module named top-level and two additional modules named required-1 and required-2. The Module Information representation for the top-level module is unique in that it does not display the module name.

Only representations of non-top-level modules are identified by the name of the module:



Declaring Directly Required Modules

For each module that requires other modules, you must assign the name of the required module to the directly-required-modules attribute in its associated Module Information system table. A module can directly require one or more other modules.

For example, suppose your current KB contains a top-level module named top, and suppose you create a new module named classes, which you want to be below the module top in the hierarchy.

To declare a directly required module of a module interactively:

1 Open the Module Information system table for the module that requires another module.

2 Enter the name of the directly required module or modules in the directly-required-modules attribute.

To declare one or more directly required modules programmatically:

➔ Use the conclude action to change the directly-required-modules attribute of a specific Module Information system table.

For example:

```
change-required-modules-for-module(module-name: symbol,
    required-modules: sequence)
MI: class module-information;
begin
    if there exists a module-information MI such that
        (the top-level-module of MI = module-name)
            then conclude that the directly-required-modules of MI =
                required-modules
end
```

Use this technique to create a module hierarchy with module branches, as shown in the figure in The Module Hierarchy.

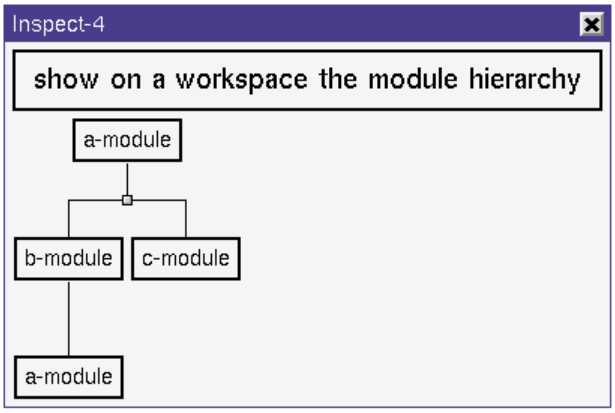
Rules for Consistent Modularization

If the current KB contains one or more modules, those modules must conform to G2's rules for **consistent modularization**, as follows:

- Every module must be named in its own Module Information system table.
- The module hierarchy must include one and only one top-level module. All modules other than the top-level module must be either directly or indirectly required by the top-level module.
- Every top-level workspace must be assigned to an existing module.
- An item and its attribute table must reside in the same module.
- The attribute table of a transient item cannot reside on a permanent workspace that does not also contain the transient item.
- An instance of a class must appear in either the same module as its class definition, or in a module that is above it in the module hierarchy. An instance of a class cannot appear in a module that is below its definition in the hierarchy.
- The dependencies among modules must not be cyclic. A **cyclic dependency** occurs when a module higher in the hierarchy directly requires another module lower in the hierarchy, but the higher module is also directly required by a module lower in the hierarchy.

The diagram below shows an illegal cyclic dependency formed by the chain of references:

a-module Æ b-module Æ a-module



MODULE-INFORMATION

Notes	OK, and note that the module hierarchy has the following cycle: A-MODULE-- B-MODULE--A-MODULE
Authors	ghw (26 May 2000 4:01 p.m.)
Change log	0 entries
Item configuration	none
Module file name	default
Top level module	a-module
Directly required modules	b-module, c-module
Module annotations	none

You can eliminate this cycle by moving the items in a-module that are referenced in b-module to another module.

When G2 detects that the modules in the current KB violate any of the rules for consistent modularization, G2 signals an error and adds information to the notes attributes of the nonconforming modules, nonconforming top-level workspaces, and so on, as shown in the previous figure.

Tip As you develop the current KB's module hierarchy, you should regularly check the notes attributes of your items.

G2 validates the consistency of modules in the current KB only. G2 does not compare the knowledge in the currently loaded modules with the knowledge stored in KB files that are not loaded. This means that if you load a KB file and another developer happens to save a change to the same KB file, G2 cannot detect whether that change affects its consistency with other modules until they are next loaded.

G2 does not evaluate the consistency and completeness of *all* references within the KB that cross module boundaries. For instance, G2 does not validate the existence of procedures and functions that are referenced but not contained in the same module. These and other execution linkage references play no role in how G2 validates the consistency of a module hierarchy.

Checking for Consistent Modularization

G2 checks that the current KB is consistently modularized when you attempt to save the KB or any module in the KB. As you work with your current KB, you can also check for consistent modularization, using Inspect. You can also check for consistent modularization programmatically, using a system procedure.

To check for consistent modularization interactively:

➔ Choose Main Menu > Inspect and enter this command:

```
check for consistent modularization
```

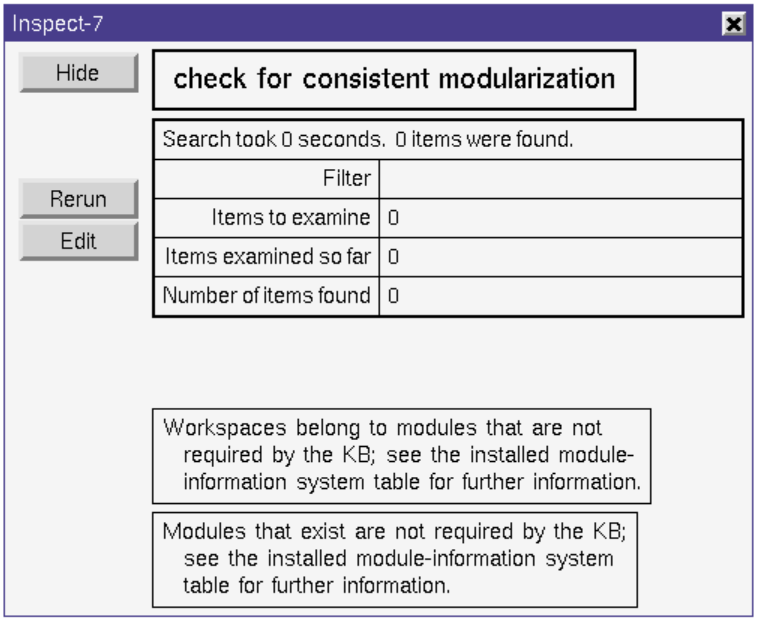
To check for consistent modularization programmatically:

➔
```
g2-check-for-consistent-modularization
    ( )
    -> return-value: sequence
```

For information on the return value, see KB and Module Operations in the *G2 System Procedures Reference Manual*.

When you issue the check for consistent modularization command, Inspect displays an Inspect workspace and places on it any messages that describe why the current KB is not consistently modularized.

The following figure shows an Inspect workspace with two messages, produced after executing the check for consistent modularization command:



In this figure, the messages from Inspect report that the current KB contains at least one top-level KB workspace whose module-assignment attribute has the value unspecified, and that at least one module exists in the KB that is not required by any other module.

To respond to these messages:

- You can use Inspect to find the workspaces whose module-assignment attribute has the value unspecified. Then, either enter a module assignment for that workspace or delete it.
- Use Inspect to view the module hierarchy. Then, for each module that is not presently required, determine whether it belongs in this KB's module hierarchy. If so, include the module name in the directly-required-modules attribute of the Module Information system table associated with the module that requires it.

Saving the Module Hierarchy

When saving a modularized KB, G2 saves each module in its own KB file. You can choose to save individual modules or all modules in the hierarchy, either interactively or programmatically.

See also Saving Your KB Knowledge for information on choosing a KB format, saving a KB, and unsavable-module change protection.

To save an entire module hierarchy into separate KB files:

1 Choose Main Menu > Pause.

2 Choose Main Menu > Save KB.

3 In the save KB workspace, include the including all required modules statement in the save module ... as command.

For a description of the default module name and filename that G2 provides, as well as requirements for specifying the KB filename, see Specifying the Filename of a Saved Module.

G2 displays a list of all the modules it will save with a confirmation message.

4 Click OK to save the modules.

To save an individual module in a module hierarchy:

1 Choose Main Menu > Save KB.

2 Edit the name of the module in the save module ... as command that appears to specify the name of the module to save.

3 Edit the associated filename to correspond to the module name you are saving.

To save a module hierarchy programmatically:

➔ Use the g2-save-module and g2-save-module-without-other-processing system procedures, as described in KB and Module Operations in the *G2 System Procedures Reference Manual*.

If any directly or indirectly required module was not loaded from an existing KB file, then when you specify the ,including all required modules phrase in the save module ... as command, G2 creates new KB files with names based on the combination of:

- The module name in the top-level-module attribute of each module's associated Module Information system table.
- The entries in the module map file, if it exists.

Note The module map file, if it exists, also determines how G2 saves modules into corresponding modularized KB files. See Using a Module Map File to Load and Save a KB for more information.

Specifying the Filename of a Saved Module

When saving a module that is directly required by other modules, you should save it to a filename that is the same as the module name. This allows G2 to find the directly required module's associated KB file by using G2's default search techniques.

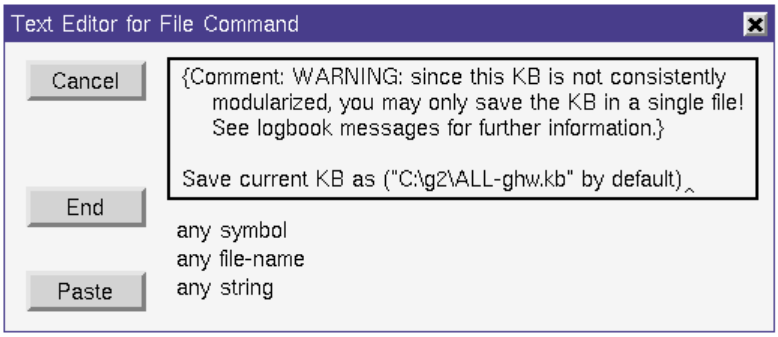
If you choose to store the module in a KB file with a different name, then in order for G2 to locate that module's KB file when loading or merging it later as a directly required module, you must also create a module map file. The module map file associates a module name with either a directory path or a KB file path. See Using a Module Map File to Load and Save a KB.

When saving a module, G2 offers defaults as follows:

- If you created the top-level module using an empty KB, G2 offers the name of the top-level module as the default KB filename used to save the specified module.
- If the current KB was loaded from an existing KB file, G2 offers the existing filename as the default KB filename used to save the specified module. This allows you change the module name of the top-level module without affecting the default KB filename in which the top-level module will next be saved.

Saving an Inconsistently Modularized KB

If the current KB is inconsistently modularized, you cannot save any of its modules into separate KB files. Instead, when you attempt to save the current KB, G2 displays a warning message, such as the following, and only permits you to save the entire KB into a single KB file.



By default, G2 stores this KB file in a file whose name begins with the `ALL-` prefix and ends with the name of the current KB's top-level module.

Deleting a Module

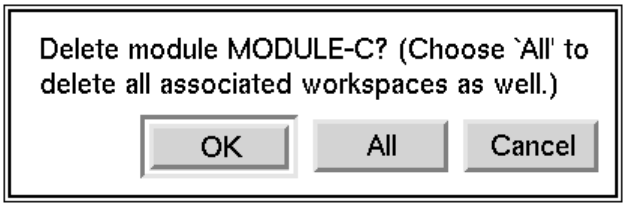
Deleting a module means removing the module, all of its associated system tables, and, optionally, all its associated items from the current KB. Recall that items are associated with a module based on the module assignment of the workspace on which the items reside. Deleting a module does *not* mean deleting the KB file in which a particular module is stored.

To delete a module from the current KB interactively:

1 Choose Main Menu > Miscellany > Delete Module.

2 From the choose a module to delete menu, select a module to delete.

G2 displays this dialog:



3 Do one of the following:

- Click the OK button to delete only the selected module and its associated set of system tables.
- Click the All button to delete the selected module, its associated set of system tables, and all KB workspaces assigned to that module.

Note If you attempt to delete a module that is required by the KB, G2 displays an appropriate warning on the confirmation dialog.

To delete a module from the current KB programmatically:

➔ Execute the g2-delete-module system procedure, as described in KB and Module Operations in the *G2 System Procedures Reference Manual*.

Note If you delete the top-level module from the current KB, G2 replaces all installed system tables with new system tables and initializes their attributes to system-defined default values.

Determining Programmatically Whether a Module is Loaded

You can use an if rule or an if statement in a procedure to determine whether a particular module exists in the current KB. This procedure accepts a module name as its argument to check whether it is the top-level-module:

```
check-module(module-name: symbol)
begin
    if the top-level-module of module-information = module-name
        then post "Module [module-name] is installed as the top-level-module
            of this KB."
        else post "Module [module-name] is not the top-level-module
            of this KB. The top-level-module is
                [the top-level-module of module-information]."
end
```

Obtaining Information about Modules

You can use the Inspect facility to show the module hierarchy of the current KB. You can perform operations on modules from the module hierarchy display. You can also display all Module Information system tables in the current KB.

At the item level, you can display the module assignment of a workspace or the items that reside upon it. You can also programmatically obtain the containing-module of an item.

Displaying the Module Hierarchy

You can display the module hierarchy of the current KB or of a particular module in the KB.

The module hierarchy shown in the Inspect workspace represents only the network of references to module names found in the directly-required-modules attributes of the Module Information system tables.

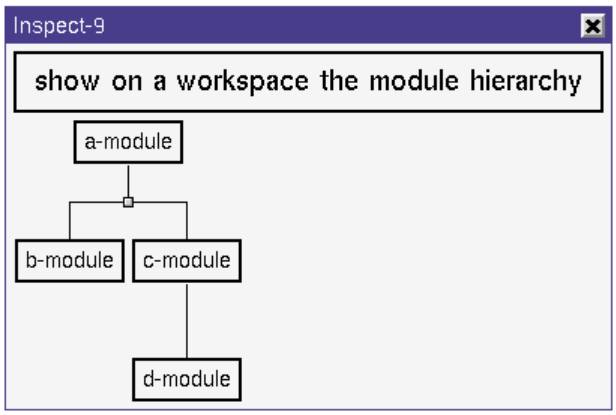
Note The module hierarchy does *not* indicate whether top-level workspaces are assigned to a particular module.

To display the module hierarchy of the current KB:

➔ Choose Main Menu > Inspect and enter this command:

```
show on a workspace the module hierarchy
```

Entering this command causes G2 to display an Inspect workspace containing a diagram of the complete module hierarchy, such as:



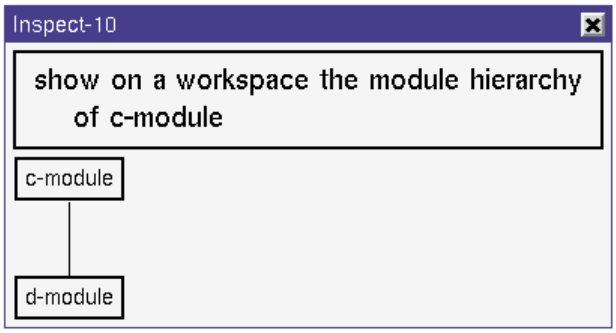
In this module hierarchy, a-module is the top-level module.

To display the module hierarchy for a particular module:

➔ Choose Main Menu > Inspect and enter a command such as:

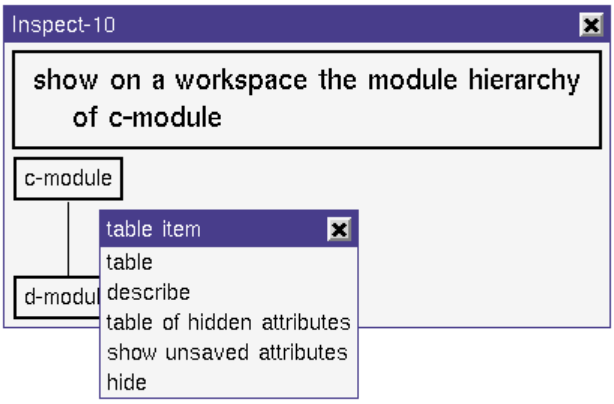
```
show on a workspace the module hierarchy of module-c
```

Entering this command causes G2 to display an Inspect workspace containing a partial module hierarchy, such as:



To perform operations on a module in the hierarchy:

1 Click on the representation of the module in the hierarchy to display this menu:



2 Select table to display the Module Information system table for the module.

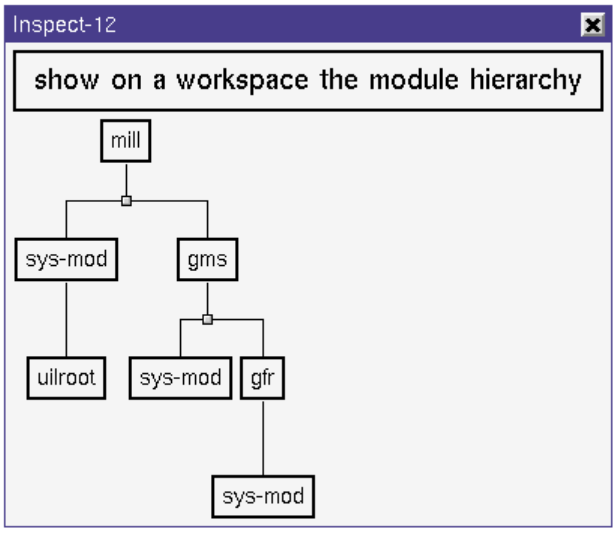
3 Select describe to describe the module by using the Describe facility.

4 Select table of hidden attributes to display a table of virtual attributes.

5 Select hide to hide the short representation of the module in the hierarchy.

If a module hierarchy contains modules that are directly required by more than one module, G2 displays the subhierarchy for the module only once. In the other locations in the hierarchy that require the module, G2 displays only the directly required module, not its submodules.

For example, if you show the workspace hierarchy for some KBs, you see that the sys-mod module is directly required by more than one module. However, G2 displays sys-mod's directly required modules only once, to the far left.



Displaying Module Information System Tables

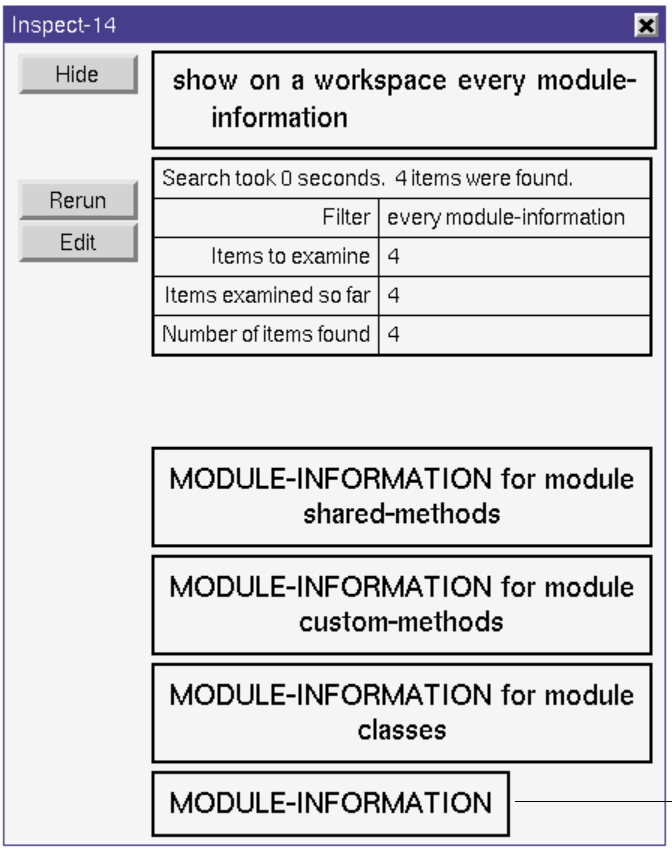
You can use Inspect to display short representations of the Module Information system tables of all loaded modules.

To display the Module Information system tables for all modules:

➔ Choose Main Menu > Inspect and enter this command:

show on a workspace every module-information

G2 displays a workspace such as the following:



In this Inspect workspace, the Module Information whose short representation does not identify a module is associated with the top-level module.

Displaying the Module Assignment of Items

If an item resides upon a workspace that is assigned to a module, you can display that module assignment.

To display the module assignment of an item:

1 Click any item to display its menu.

2 Select describe to describe the item, using the Describe facility.

If the item resides upon a workspace that is assigned to a module, its description includes a line such as:

This is assigned to module my-module.

3 Select Delete Workspace or click on the hide-workspace button to close the item.

Choosing the workspace Describe menu option reveals its module assignment, which is also available in the module-assignment attribute of the workspace attribute table and on the table of hidden attributes for the item. You can access the hidden attributes by choosing the table of hidden attributes menu choice from the item menu.

Obtaining the Containing Module for Items Programmatically

To obtain the containing module of an item programmatically:

➔ the containing-module of *item*
-> *symbol*

Returns a symbol value of the module name.

Working with Modularized KBs

A modularized KB contains one or more modules. When you save a modularized KB file, G2 saves one module per file.

As explained below, if the current KB contains modules that are inconsistently modularized, and you save the KB, G2 requires that the entire current KB and all its modules be saved into one KB file. In this case, the KB file contains knowledge about more than one module.

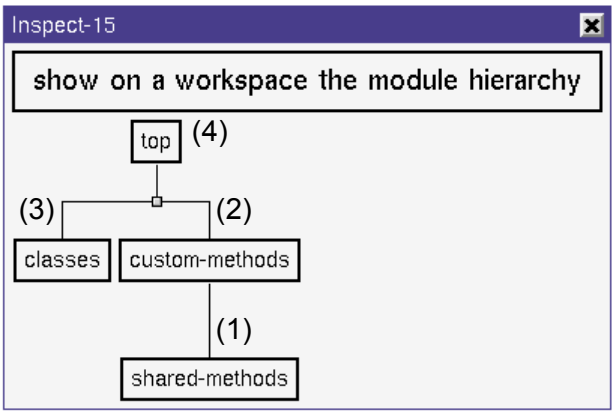
Loading a Modularized KB

If you direct G2, interactively or programmatically, to load a KB file, G2 first attempts to load all KB files that the specified KB directly or indirectly requires. Indirectly required KB files are files that contain modules that are directly required by submodules in the hierarchy.

When loading a KB, G2 does the following, in this order:

1. Traces down the module hierarchy of the top-level module in the specified KB until it finds a KB whose module does not directly require another module.
2. Loads that KB.
3. Loads, in turn, the KB that directly requires the KB already loaded.
4. Continues marching up the hierarchy until it loads all directly required modules of the top-level module.

The following figure indicates the order in which G2 loads the modularized KBs that are directly and indirectly required by the top-level module named top:



No matter how many modules directly require a particular module, G2 loads that module only once.

Loading Modularized KBs and Detecting Conflicts

After you direct G2 to load a modularized KB, as G2 loads the modularized KBs that form a particular module hierarchy, G2 actually performs one load operation and one or more merge operations. First, G2 loads the KB whose module requires no other modules, then G2 merges one or more KBs into the current KB.

Because G2 actually performs merge operations to load the second through last KBs into the current KB, it is possible for G2 to detect conflicts among the definitions found in the various KBs. Therefore, when loading a KB that directly require modules in other KBs, it is recommended that the automatically resolve conflicts box in the Load KB dialog is selected. This option is selected by default because resolving conflicts by hand is difficult and time consuming.

Tip For information on whether to select the automatically resolve conflicts check box and how to respond to a conflict workspace, see Detecting Conflicting Class-Definitions.

Loading a Particular Version of a KB File

Your application development team might work with different versions of the same KB file, with each version stored in different directories. If a KB directly requires a module located in a particular version of another KB, you can use either of two techniques to ensure that a particular KB file is loaded:

- Define a module search path, to specify the order in which G2 searches a list of directories for KB files to load. See Using a Module Search Path to Load KB Files.
- Create a module map file, to associate a directly required module's name with a KB file of the same name in a particular directory, or even with a particular KB filename. See Using a Module Map File to Load and Save a KB.

Automatic Loading of Directly Required Modules

When a module stored in one KB directly requires a module stored in another KB, and you load the first KB, G2 automatically loads the directly required module's KB first, loads the requiring KB next, and so on, until the specified KB is loaded.

By default, G2 looks for a KB file with the same name as the directly required module. For example, if you load a KB that contains a module named top, and top directly requires another module named classes, then G2 attempts to find the module by searching for and loading a KB file named `classes.kb` in the same directory where the KB file containing the module top resides.

Tip You can optionally use a module map file or module search path, to direct G2 where to find directly required modules. For more information about using a module map file, see Using a Module Map File to Load and Save a KB. For more information about using a module search path, see Using a Module Search Path to Load KB Files.

Merging a Modularized KB into the Current KB

Merging any KB file means to read a stored KB and to add its knowledge to the current KB. You can merge any KB into the current KB.

To merge a KB interactively:

➔ Chose Main Menu > Merge KB.

G2 automatically selects the Merge in this KB option in the save KB workspace.

To merge a KB programmatically:

➔ Invoke the g2-merge-kb or g2-merge-kb-ex system procedure, as described in KB and Module Operations in the *G2 System Procedures Reference Manual*.

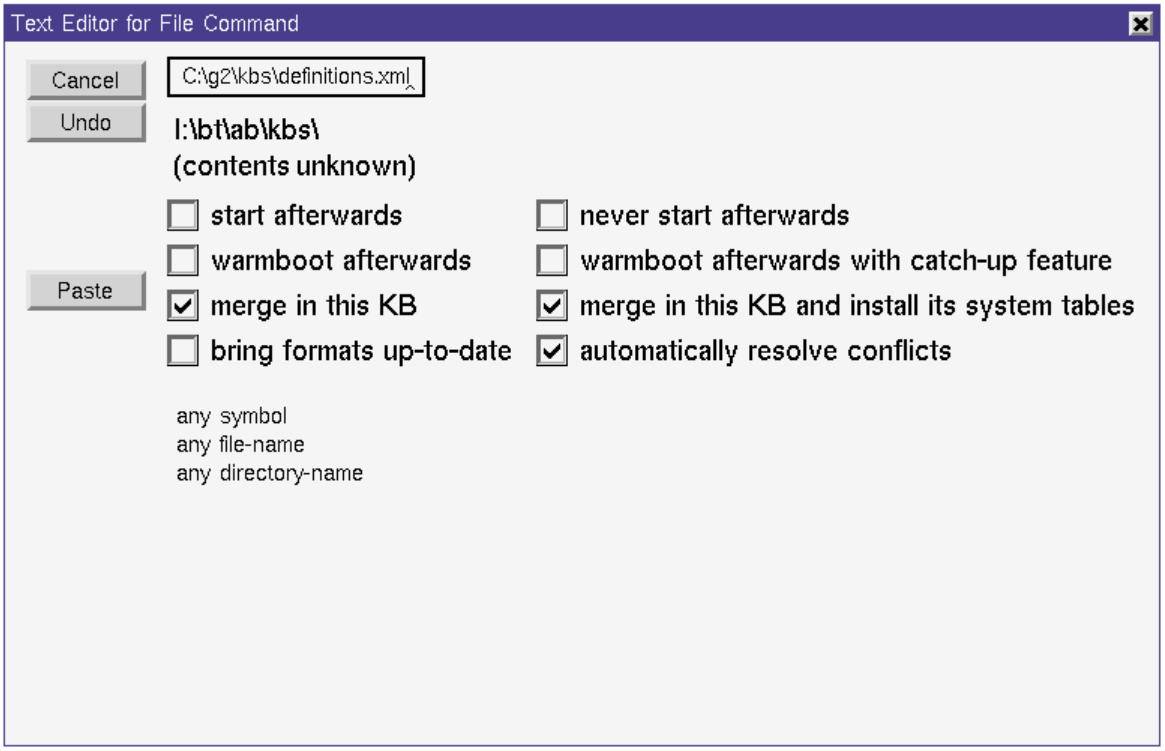
Merging Directly Required Modules

When you merge a modularized KB, G2 automatically merges other KBs that the specified KB directly or indirectly requires. This is also true when you load a KB that requires other modules. G2 selects the other KBs to merge as described in Loading a Modularized KB.

Installing System Tables of a Merged Modularized KB

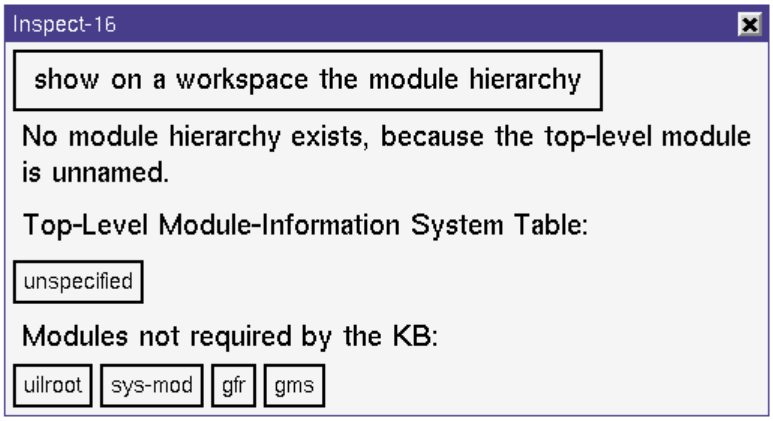
When you merge a KB, you can either install or not install its system tables into the current KB. Installing the system tables of a merged KB causes the module described in the Module Information system table of the merged KB to become the top-level module in the resulting current KB.

To install the system tables of a merged KB, check the merge in this KB and install its system tables check box, as shown in this figure:



If you merge a modularized KB *without* installing its system tables, the resulting KB contains the merged modules, but they may not participate in the current KB's module hierarchy.

For example, if the current KB contains no top-level module, after merging a KB without installing its system tables, the Inspect facility shows that there is no module hierarchy, because there is no top-level module:



Because the resulting KB has no top-level module, the merged modules are not directly required, and the resulting KB is inconsistently modularized.

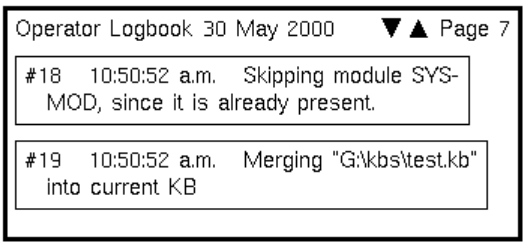
If the current KB contains modules, the result of merging a KB and installing its system tables depends upon whether the current KB already has a top-level module, as follows:

- If the current KB contains modules but no top-level module, the current KB is not consistently modularized. When you merge a KB and install its system tables, the top-level module in the merged KB becomes the top-level module. The current KB remains inconsistently modularized.
- If the current KB contains modules including a top-level module, and you merge a KB and install its system tables:
  - G2 uninstalls the installed set of system tables. The uninstalled system tables remain in the current KB and remain associated with the same module.
  - G2 installs the system tables associated with the top-level module in the merged KB. This means that the top-level module in the merged KB becomes the top-level module in the current KB.

Ignoring Modules with Duplicate Names

If a merged KB directly requires a module with the same module name as a module already in the current KB, G2 does not attempt to load that module again. G2 ignores the second and subsequent attempts to load a module with the same name, even if those modularized KBs reside in different directories.

When G2 ignores merging a module in this fashion, G2 places a message on the Operator Logbook, such as the example that follows:



In this example, if your current KB is modularized and contains the module uilroot, and you merge another KB whose top-level module classes also directly or indirectly requires a module named uilroot, G2 does not attempt to merge another KB, which may be located in the same directory, and which also contains a module named uilroot.

Merging a Particular Version of a KB

As explained in Loading a Particular Version of a KB File, you can also merge a particular version of a KB by defining a module search path or creating a module map file.

For more information, see Using a Module Search Path to Load KB Files and Using a Module Map File to Load and Save a KB.

Using a Module Search Path to Load KB Files

By default, when loading a KB whose module directly requires other modules, G2 searches in the same directory as the loaded KB for the files containing the other modules.

You can also load KB files that are located in other directories. When structuring your KB directories in this way, you might want to specify a **module search path**. A module search path is a list of directories that G2 searches to find the KB file containing a directly required module.

Tip Specifying a module search path can be especially helpful if you load or merge KB files for which there are multiple copies stored in different directories. In this situation, you specify a module search path to direct G2 to load or merge a particular version of the KB file, if it exists in one directory rather than in another directory.

G2 consults the list of directories in the module search path in these situations:

- If the current KB is modularized and G2 cannot find a directly required modularized KB file in the directory from which the current KB was loaded.
- If the current KB is empty and G2 cannot find a directly required KB in the directory that contains the KB file that you directed G2 to load.

Tip On Windows platforms, you can start the G2 server with a batch file to load a default module search path. For details, see the `readme-g2.html` file.

Specifying a Module Search Path

There are three ways you can direct a G2 process to use a module search path to locate KB files.

- Include the `-module-search-path` option in the command line that launches the G2 process. Specify a list of directory paths as the argument to this option. The `-module-search-path` command-line option is described in module-search-path.
- Before starting G2, use the appropriate operating system command to define and set the `G2_MODULE_SEARCH_PATH` environment variable. Specify a list of directory paths as its value.
- At any time after G2 is launched, specify a module search path in the module-search-path attribute of the Server Parameters system table.

Module Search Path Syntax

A module search path specified on the command line or with an environment variable is a quoted text value containing one or more directory paths with blank-space characters separating directory paths. A module search path specified with the module-search-path attribute on the Server Parameters system table is one or more quoted directory paths separated with commas.

When specifying the module search path on the command line or with an environment variable, special syntax is provided to support directory and file names that contain blank spaces. Blank spaces in file paths are fairly common on Windows platforms and are supported by the NTFS and FAT32 file system built on top of DOS. Unix also allows spaces, but its parsing methods have discouraged their use. You do not need to use the special syntax when specifying path names with spaces for the module-search-path attribute on the Server Parameters system table.

To specify a file path that contains blank spaces:

➔ Enclose the path in single forward quotes (apostrophes); do not use the backquote character.

You can use the single-quote delimiters for both the Windows and Unix styles of file paths. Although it is not necessary, you can enclose file paths without spaces in single quotes. Paths with embedded apostrophes are supported, but a path that has both an embedded space and an embedded apostrophe is not supported.

Some examples of supported search paths specified for the module-search-path attribute of the Server Parameters system table are:

```
"/home/user/kbs/", "/gensym/kbs"

"C:\Program Files\Gensym\g2-2011\g2\kbs\demo\"

"\\server1\Program Files\Gensym\g2-2011\g2\kbs\demo\"

"\kbs\current-release\", "D:\product\marketing kbs\"

"/home/user kbs/", "/gensym/kbs"
```

Examples of supported module search paths specified on the command line or with the environment variable are:

```
"/home/user/kbs/ /gensym/kbs"

"'C:\Program Files\demo\kbs'"

"'\\server1\Program Files\Gensym\g2-2011\g2\kbs\demo\'"

"\kbs\current-release\ 'D:\product\marketing kbs\'"

"'/home/user kbs/' '/gensym/kbs'"
```

This path is not supported:

```
"'\Program Files\doc's-kbs\'"
```

Here are example command scripts you can use on a UNIX platform to start a G2 process that searches for KBs in two directories other than the current directory:

```
# Start G2 and specify two directories in its
#  module search path
g2 -module-search-path "/dev/g2-mods/ /usr/kmm/g2-mods"
```

or

```
# Start G2 and specify two directories in its
# module search path
setenv G2_MODULE_SEARCH_PATH "/g2-mods/ /usr/kmm/g2-mods"
g2
```

How G2 Searches for KB Modules

When you load a new top-level KB, G2 searches the directories listed in the module search path, as follows:

1. G2 determines whether the directly-required-modules attribute in that KB's Module Information system table refers to other KBs.
   a. If so, G2 searches for the directly required KB file in the directory that contains the top-level KB file.
   b. If G2 does not find the directly required KB file in the top-level KB file's directory, G2 searches, in order, each directory specified in the module search path.
2. When G2 finds a KB file of the correct name, G2 determines whether it, in turn, directly requires other KB files, then follows Steps 1a and 1b to locate that KB file.
3. If G2 cannot find either the top-level KB file or its directly required KB file(s), using Steps 1 through 2, G2 searches the directory that was current when you launched G2. If G2 does not find the KB file in this directory, G2 reports an error.

Using a Module Map File to Load and Save a KB

When you work with KB files whose filenames are not the same as the names of the modules they contain, you can create a **module map file** to associate a module name with a particular KB file. If this file exists, G2 consults it to find the KB file that contains a particular module.

Locating the Module Map File

If you create a module map file, it must be named `g2.mm`. When a G2 process starts, it searches for the module map file, as follows:

1. If you include the `-module-map` option in the command line to start G2, G2 searches for the file at the fully qualified file pathname that is specified as an argument to the option.
2. If you do not include the `-module-map` option in the command line to start G2, G2 checks whether a `G2_MODULE_MAP` environment variable is defined. If such an environment variable exists, G2 attempts to open `g2.mm` under the directory path assigned to that variable.
3. If G2 locates no `g2.mm` file using Steps 1 and 2, G2 attempts to open `g2.mm` located in the directory that was current when you launched G2.

Tip The `-module-map` command-line option is described in module-map.

Adding Entries to the Module Map File

Use any text editor to create a module map file. It should contain only ASCII text.

Each line in the module map file associates the name of a module contained in a KB with either a fully qualified directory path or a fully qualified file path. Use one or more blank spaces to separate the module name from its associated directory path or file path.

When specifying a directory path, include a trailing directory delimiter character:

For this platform...	Use this delimiter...
Windows	\
UNIX	/

For example, a module map file that describes two KBs that reside on a Windows platform could contain:

```
vehicle-root C:\Program Files\Gensym\g2-2011\kbs\modules\vh.kb
vehicle-classes C:\Program Files\Gensym\g2-2011\kbs\shared\
```

The first line specifies that G2 loads, merges, and saves the module named vehicle-root using the file `C:\Program Files\Gensym\g2-2011\kbs\modules\vh.kb`. The second line specifies that G2 must load, merge, or save the module named vehicle-classes using a KB file of the same name (`vehicle-classes.kb`) under the directory `C:\Program Files\Gensym\g2-2011\kbs\shared\`.

6

System Tables

Describes the use of system tables to set global preferences.

Introduction

System tables define certain global defaults applicable to an entire KB, similar to the Preferences you can set in many applications. The attributes in a system table affect the settings of related system parameters.

This chapter describes all system tables in alphabetical order.

Using System Tables

Every KB has one set of system tables in effect at a given time. G2 refers to these as the installed system tables.

In a new KB, the installed system tables contain the default values that G2 provides for each system table attribute. If you change one or more system table attributes and then save your KB, the modified system tables are saved as a permanent part of the KB's knowledge. G2 installs these system tables when you next load the KB.

Each module in a KB has an associated set of system tables which includes an instance of every system table except the Servers Parameters system table. There is only one Servers Parameters system table per KB which is created by G2's initialization process and is associated with the top-level module. The set of system tables for the top-level module is the installed set.

Two reasons why modules include their own set of system tables are:

- Every module is capable of being the top-level module, and therefore requires a complete set of system tables.
- The critical information G2 needs to know about a module is contained in the Module Information system table.

For a description of the Module Information system table, see Module Information.

You can replace one set of installed system tables with another by merging in a KB. If you merge a KB module into an existing KB, one of the options on the KB Merge menu is:

merge in this KB and install its system tables

Choosing this option causes G2 to install the system tables of the merged KB, overriding those of the existing KB. The overridden system tables are still present in the KB, but their attributes are no longer in effect. You can search for system tables in your KB by using the Inspect facility. For more information on merging KB files with system tables, see Merging a KB File.

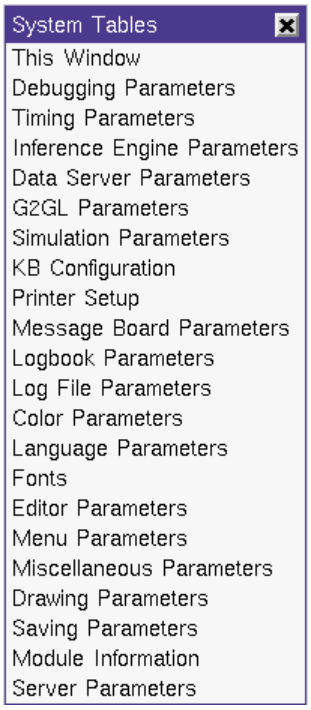
You can also start G2 with one or more optional command-line options that let you specify a module map file or a module search path. Using either of these command-line options can affect the values of the Module Information system table.

For information about using the module map file, or module search path command-line options, see module-map and module-search-path. For a description of creating a module map file, see Using a Module Map File to Load and Save a KB.

Changing System Tables Values Interactively

To access the installed set of system tables:

➔ Choose Main Menu > System Tables. This menu appears:



When you select a system table, G2 displays its attributes and the values for those attributes. Unlike most items, system tables are not associated with a workspace and display directly on the Gensym background area.

Changing System Table Values Programmatically

You can use the conclude action to change the value of most system table attributes. For example, to change the default font size for the Text Editor, enter a statement such as:

```
conclude that the font-for-editing of fonts is extra-large
```

Information about the types and read and write access of system-defined attributes is available in the *G2 Class Reference Manual*.

Color Parameters

The Color Parameters system table lets you control which colors appear in the background- and foreground-color menus of G2, and the order in which those colors appear.

To display the color menu of a workspace:

➔ Choose KB Workspace > Color > *foreground-or-background-color*.

where *foreground-or-background-color* is either background-color or foreground-color. Both choices have the same color selections. The color menus appear in other locations within G2, such as the Icon Editor.

Controlling the Menu Order of Colors

The color-menu-ordering attribute controls the menu ordering. By default, the system-defined set of colors is ordered this way on the color menus:

This value...	Arranges the colors...
hue	By hue. For example, greens are grouped together, reds are grouped together, blues are grouped together, and so on.
intensity	From light to dark.
alphabetic	Alphabetically, according to the color names.

Specifying the Colors on the First Color Menu

The color-on-1st-level-color-menu attribute specifies what colors appear on the first level menu as follows:

This value...	Provides...
color-name, color-name...	Any color from the G2 color set. Enter the names of the color in the order of your choice to construct your own color set. G2 accepts some variations in spelling. For example, you can enter grey instead of gray. You can also provide a symbol of the form RGB*rrggbb* as a valid color name, where *rr*, *gg*, *bb*, are the 8-bit hex values for red, green, and blue. For details, see Other Literal Terms.
standard-set	A subset of the G2 color set consisting of: aquamarine, black, blue, brown, dark gray, gray, green, light gray, orange, purple, red, white, yellow
all	The full G2 color set as they appear: transparent, foreground, black, dim gray, dark gray, gray, light gray, white, pink, Indian red, salmon, brown, orange, red, tan, gold, coral, sienna, wheat, medium goldenrod, khaki, goldenrod, yellow, green yellow, pale green, forest green, lime green, green, aquamarine, medium aquamarine, light blue, turquoise, cadet blue, cyan, sky blue, slate blue, medium blue, blue, medium orchid, dark slate blue, thistle, plum, purple, violet, magenta, maroon, and violet red. The colors are arranged by hue, since that is the default for the color-menu-ordering attribute

Defining the Colors on the Second Color Menu

The color-on-2nd-level-color-menu attribute defines what colors appear on the second level menu. Specify this attribute as you would for the first level color menu.

Specifying the Number of Columns for the First Color Menu

The number-of-columns-for-1st-level-color-menu attribute determines the number of columns, up to 7, that you want the menu to contain.

Specifying the Number of Columns for the Second Color Menu

The number-of-columns-for-2nd-level-color-menu attribute determines the number of columns, up to 7, that you want the menu to contain.

Indicating Whether to Dismiss the Color Menu

The dismiss-color-menu-after-choosing? attribute indicates whether the color menu remains displayed after you pick a color.

Class-Specific Attributes of Color Parameters

The class-specific attributes of the Color Parameters system table are:

Attribute	Description
color-menu-ordering	Controls the order in which colors are displayed in menus.
Allowable values:	{hue | intensity | alphabetic}
Default value:	hue
colors-on-1st-level-color-menu	Controls which colors appear in the first level color menus.
Allowable values:	{*color-name* [, ...] | standard-set | all | none}
Default value:	standard-set
colors-on-2nd-level-color-menu	Lets you create a subset of colors from the first level colors menu. It has the same syntax as the color-on-1st-level-color-menu attribute.
Allowable values:	{*color-name* [, ...] | standard-set | all | none}
Default value:	all

Attribute	Description
number-of-columns-for-1st-level-color-menu	Controls the number of columns to display the 1st level color menu.
Allowable values:	1 – 7
Default value:	1
number-of-columns-for-2nd-level-color-menu	Controls the number of columns to display the 2nd level color menu.
Allowable values:	1 – 7
Default value:	3
dismiss-color-menu-after-choosing?	Specifies how you want to dismiss menus. If yes, G2 dismisses the menus immediately. If no, all of the menus are left up, and you must dismiss them manually.
Allowable values:	{yes | no}
Default value:	yes

Data Server Parameters

The Data Server Parameters system table lets you control data service.

Specifying a Data Server Alias

The data-server-aliases attribute lets you substitute an alternate name for one or more data servers. For example, you could use this attribute to create aliases for each of the available data servers. The syntax for defining a data server alias is:

symbolic-expression implies service through
{inference engine |
g2 [simulator | meter | data server] |
gfi data server |
gsi data server}

An example is:

```
robot-controller implies service through gsi data server, process-computers
    implies service through gsi data server
```

GFI and the G2 Simulator are superseded capabilities. For more information, see Appendix F, Superseded Practices.

More than one alias can imply service through the same data server. Thus, both robot-controller and process-computers imply service through the G2 Gateway data server.

Once an alias exists, you can use it in the data-server attribute of a variable to indicate where G2 obtains a new current value for a variable, though G2 does not include a Text Editor prompt for any aliases.

Specifying Data Service Scheduling Priority

The priority-of-data-service attribute specifies the default priority at which G2 schedules data server requests.

For more information about scheduling, see Task Scheduling.

Turning on G2 Meters

The g2-meter-data-service-on? attribute determines whether variables that have G2-meter as their data server receive values.

Although meters use only a small fraction of the available processing time, you may want to deactivate them when the KB does not need them. To do this, set this attribute to no. To activate any enabled meters, set this attribute to yes.

If a KB has enabled G2 meter variables and this attribute is set to no, G2 signals an error. Disable any G2 meter variables to prevent that error.

For a complete description of using G2 meters, see G2-Meters.

Class-Specific Attributes of Data Server Parameters

The class-specific attributes of the Data Server Parameters system table are:

Attribute	Description
data-server-aliases	Specifies which data server is implied by each data server alias.
Allowable values:	*dataserver-alias-symbol* implies data service through {inference engine | g2 simulator | g2 meter | g2 data server | gfi data server | gsi data server} GFI and the G2 Simulator are superseded capabilities. For further information, see Appendix F, Superseded Practices.
Default value:	none
priority-of-data-service	Determines the priority at which tasks are scheduled to service data servers, by specifying an integer between 1 and 10. The highest priority is 1.
Allowable values:	1 – 10
Default value:	4
g2-meter-data-service-on?	Controls whether G2 meter variables receive values.
Allowable values:	yes no
Default value:	no

Debugging Parameters

The Debugging Parameters system table controls the kind of error feedback G2 provides while a KB is running. The following sections summarize the debugging behavior that is specified by the attributes on the Debugging Parameters system table. For information on how to use these attributes for debugging your KB, see Debugging and Tracing.

G2 saves the values of these attributes with the KB: warning-message-level, tracing-message-level, breakpoint-level, source-stepping-level, show-procedure-invocation-hierarchy-at-pause-from-breakpoint, disassembler-enabled?, and tracing-file.

Controlling Error and Warning Message Displays

The warning-message-level attribute controls the error and warning messages that G2 displays in the operator logbook while a KB is running.

This value...	Informs G2 to...
0	Not to display any warning or error messages.
1	Display error messages, and display warning messages when G2 encounters problems in a KB, such as a rule that it cannot interpret.
2	Display error messages, and display all level 1 warning messages, plus warning messages about missing information such as a non-existent variable reference from a data server.
3	Display all error messages, and display level 2 warning messages, plus messages about conditions that are interesting but do not necessarily indicate that something is wrong. For example, a level 3 message can inform you that a particular reference in a rule does not denote an existing item, but this may not be a problem. Rather, it may be an expected side effect of the rule.

Specifying Debugging Trace Messages

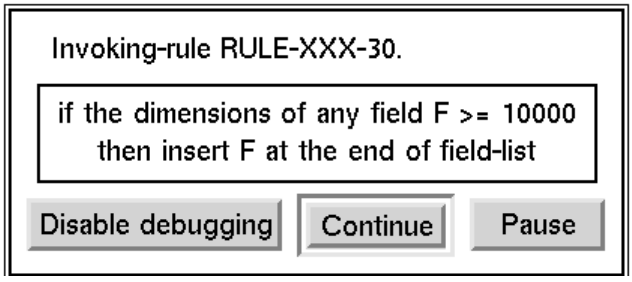
The tracing-message-level attribute controls the **trace messages** that G2 displays in the operator logbook. Trace messages tell you what steps G2 is taking to evaluate procedures, methods, rules, formulas, and display expressions.

This value...	Informs G2 to...
0	Not display any trace messages.
1	Display messages each time G2 begins or finishes evaluating a procedure, rule, etc.
2	Display messages for major steps; for example, when evaluating the antecedent of rules and when evaluating actions in procedures, methods, rules, formulas, and expressions.
3	Display messages at every step.

Specifying Breakpoints for Debugging

The breakpoint-level attribute tells G2 to halt a running KB at particular times, called **breakpoints**, to display a trace message, and to wait for you to acknowledge the message.

The messages that G2 displays at breakpoints are identical to those of the corresponding level of trace messages. At a breakpoint, however, in addition to displaying the trace message, G2 halts and displays a dialog like this:



G2 cannot continue until you click one of the buttons:

This button...	Has this effect...
Disable debugging	Changes the value of the tracing-and-breakpoints-enabled? attribute to no, and continues running without tracing or breakpoints.
Pause	Pauses G2 so that you can view the trace messages and examine other changes in the state of your kb. You can also view the current procedure invocation hierarchy.
Continue	Resumes G2. This is the default. Select it by pressing the Return key.

The commands Load KB, Merge KB, Clear KB, Reset G2, Restart G2, and Delete Module are not available while G2 is paused at a breakpoint.

Because G2 stops at breakpoints, it does not run in real time when breakpoints are set. Thus, you should never set breakpoints when G2 is controlling an application in real time. Breakpoints are convenient, however, when you are running a simulation that does not run in real time, because you can halt as long as you like at a breakpoint without altering the simulation behavior.

When you use breakpoint debugging, it is best to change to simulated time. This has the effect of suspending the real-time clock during debugging. For more information about using simulated time, see Timing Parameters.

G2 also supports dynamic breakpoints in the Telewindows client. For more information, see the *Telewindows User's Guide*.

The G2 Simulator, which can provide simulation that does not run in real time, is a superseded capability. For more information, see Appendix F, Superseded Practices.

Specifying Single-Stepping through Source Code

The source-stepping-level attribute controls whether single-stepping through procedure source code is enabled. When G2 is single-stepping through source code, before the next line of source code is executed, it performs a similar action as when a halt action is executed. In the Telewindows client, the standard Windows debugger appears, and in the server, a dialog appears that shows the source code around the line of source code G2 is about to execute, the line numbers, the contents of the stack, and the local variable bindings.

This value...	Informs G2 to...
0	Not allow single-stepping though source code.
1	Allow single-stepping though source code.

For more information, see Stepping Through Procedure Source Code.

Enabling Tracing and Breakpoints for Debugging

Specifying yes for tracing-and-breakpoints-enabled? enables tracing and breakpoint functionality. This attribute provides a convenient way of turning tracing and breakpoints on and off without editing the attributes that specify what items should be traced and at what level they should be traced.

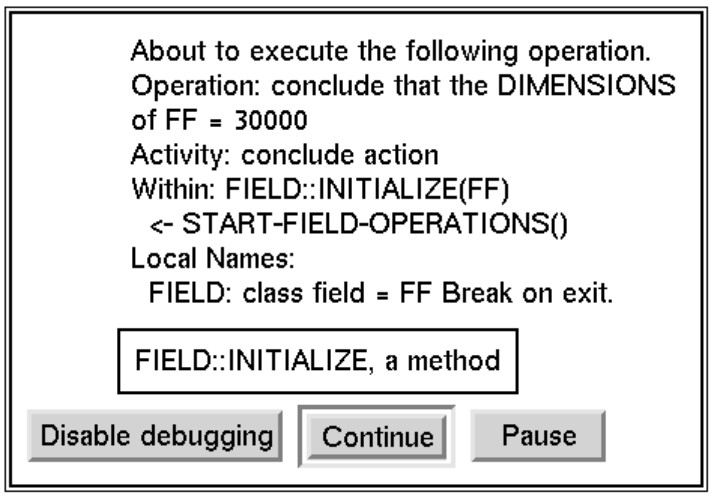
The specification of what and how items should be traced and paused is determined for the entire KB by the values of the tracing-message-level, breakpoint-level, and source-stepping-level attributes of the Debugging Parameters system table; for individual items it is determined by the tracing-and-breakpoints attribute of the items.

The tracing-and-breakpoints-enabled? attribute is not savable, so each time you load a KB it has its default value of no. By default, G2 does not stop for any breakpoints and does not display any trace messages.

See Displaying Trace Messages and Specifying Breakpoints and Tracing on how to use tracing and breakpoints for debugging your kb.

Displaying the Procedure Invocation Hierarchy while Paused

When G2 halts at a breakpoint, it displays a dialog like this:

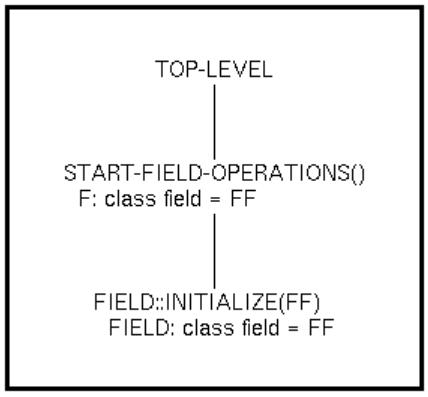


The show-procedure-invocation-hierarchy-at-pause-from-breakpoint attribute controls whether Pause on the breakpoint dialog runs the Inspect command show on a workspace the procedure invocation hierarchy. The Inspect command will be run when this attribute is yes, the default value.

To display the procedure-invocation hierarchy:

➔ Click the Pause button on the breakpoint dialog.

Here is an example of a procedure-invocation hierarchy display:



See Specifying Breakpoints for Debugging for more information about debugging with breakpoints.

Enabling the Display of Disassembled Code

The disassembler-enabled? attribute controls whether disassembled code is ever displayed.

When the disassemble-enabled? attribute is yes, three changes occur to the G2 environment that facilitate debugging:

- The describe menu choice on a procedure, method, or rule shows the corresponding byte code representation.
- G2 error messages indicate the byte code instruction that was running when the error was generated.
- The Inspect command show on a workspace the procedure invocation hierarchy indicates the byte code instruction that is running for every procedure invocation.

Also see Showing Disassembled Code.

Saving Tracing Data to a File

To write tracing messages to a file, specify a file name as a text-value for the tracing-file attribute, and set the enable-explanation-controls attribute in the Miscellaneous Parameters system table to yes. See Saving Tracing Data to a File for a detailed description.

Specifying the Display Interval for Explanation Data

The dynamic-display-delay-in-milliseconds attribute allows you to specify the number of milliseconds that dynamically displayed explanation data remains on display. You can enter an integer between 0 and 6000. The default value is 200.

Class-Specific Attributes of Debugging Parameters

The class-specific attributes of the Debugging Parameters system table are:

Attribute	Description
warning-message-level	Controls the error and warning messages that G2 displays while a KB is running.
Allowable values:	{0 (no warning messages) | 1 (kb errors only) | 2 (kb errors and deficiencies) | 3 (kb errors, deficiencies, and other conditions) }
Default value:	2 (KB errors and deficiencies)
tracing-message-level	Specifies the trace messages that G2 displays in the operator logbook.
Allowable values:	{0 (no trace messages) | 1 (trace messages on entry and exit) | 2 (trace messages at major steps) | 3 (trace messages at every step) }
Default value:	0 (no trace messages)
breakpoint-level	Tells G2 to halt a running KB at particular times, called breakpoints, display a trace message, and wait for you to acknowledge the message.
Allowable values:	{0 (no breakpoints) | 1 (breakpoints on entry and exit) | 2 (breakpoints at major steps) | 3 (breakpoints at every step) }
Default value:	0 (no breakpoints)

Attribute	Description
source-stepping-level	Tells G2 to allow single stepping though procedure source code in the standard Windows debugger in the Telewindows client.
Allowable values:	{0 (no source stepping) | 1 (source stepping)
Default value:	0 (no source stepping)
tracing-and-breakpoints-enabled?	Controls whether G2 can display messages or set breakpoints, regardless of the value of any other system table attribute or the attribute of any item.
Allowable values:	{yes | no}
Default value:	no
show-procedure-invocation-hierarchy-at-pause-from-breakpoint	Controls whether Pause on the breakpoint dialog runs this Inspect command: show on a workspace the procedure invocation hierarchy.
Allowable values:	yes, no
Default value:	yes
disassembler-enabled?	Controls whether disassembled code is ever displayed.
Allowable values:	yes, no
Default value:	no
generate-source-annotation-info	Controls whether source-code annotation information is generated when you compile your procedures. The information makes it possible for G2 to show you which procedure statement is responsible for an error.
Allowable values:	yes, no

Attribute	Description
Default value:	yes
tracing-file	Names a file to which tracing data is written.
Allowable values:	none or a pathname
Default value:	none
dynamic-display-delay-in-milliseconds	Specifies the number of milliseconds that dynamically displayed explanation data remains on display.
Allowable values:	0 - 60000
Default value	200

Drawing Parameters

The Drawing Parameters system table accommodates several options that affect graphical representation and drawing scheduling within G2. The term *drawing* refers to the way in which G2 renders all items that display within a KB, such as tables, workspaces, icons, and messages, to name just a few.

Specifying Scheduled Drawing

The allow-scheduled-drawing? attribute specifies whether drawing is a scheduled task. Drawing occurs in one of two modes, **scheduled** drawing mode (the attribute is yes), or **immediate** drawing mode (the attribute is no). Scheduled drawing is the default mode when you start G2. Immediate drawing is a superseded capability. For more information, see Appendix F, Superseded Practices.

When scheduled drawing is in effect:

- G2 consolidates drawing commands and eliminates unnecessary redraw and refresh operations.
- You can invoke KB drawing on demand, using the g2-work-on-drawing system procedure from within any procedure from which you want drawing to occur on demand.
- The scheduler can allocate large drawing tasks over a longer interval to avoid delaying computational processing.

In effect, scheduled drawing lets G2 continue other processing while completing its drawing tasks. This attribute works in conjunction with paint-mode?. If scheduled drawing is in effect, paint-mode? must be set to yes.

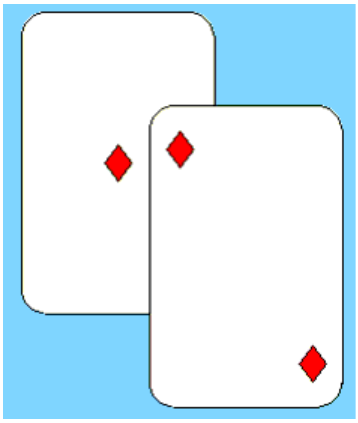
For a complete description of system procedures, see the *G2 System Procedures Reference Manual*. For more information about how the scheduler handles drawing tasks and priorities, see Task Scheduling.

Specifying the Paint Drawing Mode

The paint-mode? attribute specifies whether the **Paint** drawing mode (the attribute is yes), or the **XOR** drawing mode (the attribute is no) is in effect. Paint mode is the default when you start G2.

XOR mode is a superseded capability. For more information, see Appendix F, Superseded Practices.

In Paint mode, icons maintain their individual color patterns without distortion if you place them directly on top of one another. Paint mode supports the use of full and defined color regardless of where you place an item, as illustrated next:



Note Item menu choices let you choose the layering order of items interactively. For a description of these menu choices, see Lifting to the Top and Dropping to the Bottom. System procedures let you change the layering order of items programmatically.

Controlling the Set of Rendering Colors

The image-palette attribute controls the set of colors G2 uses to render workspace background images, allowing G2 to allocate more colors than usual.

Displaying Colors on Your System

Most color monitors on which G2 runs can display millions of colors. Because of certain limitations in display hardware or in current window systems, however, most monitors display only a limited number of colors at one time. Also, other running applications affect which colors are available to display at any time.

Each application running on your system competes on a first-come-first-served basis for the use of available colors. Once an application uses a color, fewer colors are available. If all of the applications running on a display require the same color palette, no conflict exists. For instance, if you have a system dedicated to running G2 and start a second G2 process that requires the *same* color palette, both G2 processes have the colors they require.

If you start an application on your system prior to running G2 and that application uses many colors, however, it can effectively prevent G2 from accessing its full color palette.

Note On PC platforms, the window with the focus (the foreground window under Windows) is guaranteed to have a full color palette. When G2 is in the foreground, it will use any colors it requires from other applications. Whenever you do not display G2 in the focus window, KB colors can be compromised.

Selecting a Color Palette

G2 can display color images from a graphics image file (GIF) file. When a color image is drawn in a KB, G2 chooses a color from its current color palette that most closely matches a given pixel of the image, compromising the color if necessary. As noted in the example above, the G2 color palette may be reduced to fewer colors because of another application. The image-palette attribute provides a means of controlling the set of colors from which G2 chooses.

Note On Windows platforms, if the monitor is set to use more than 256 colors, the color specified in the image-palette attribute is not used, and the GIF file is displayed with its true colors.

The image-palette attribute lets you select from 5 fixed palettes and 2 custom (image-specific) palettes as follows:

Available Palettes	Description
black and white 	Images are drawn in black and white, rendering a pixel as either black if its intensity is below a certain threshold, or white if it is above a certain threshold. The diagram to the left shows a portion of an image drawn in black and white.
standard grays 	Images are drawn using only the pure gray colors from the standard G2 palette. Using standard grays provides a total of 7 shades of gray: black, dim-gray, dark-gray, gray, light-gray, extra-light-gray, and white. In contrast to the black and white diagram, this figure shows part of the same image drawn with standard gray.
standard colors	Images are drawn using any of the 63 standard G2 colors. This is the default. Note that since the standard G2 palette is not uniformly spread over all colors, G2 may be unable to render some continuous-tone images satisfactorily.
extended grays	Images are drawn using pure shades of gray from an extended palette of 64 uniformly-spaced grays.
extended colors	Images are drawn using an extended palette of approximately 64 additional colors, spread uniformly in color space. Use this if you are using a full-color image as a workspace background and the colors are not displaying properly. Using extended colors is generally the best compromise when you want to display many disparate images in a single KB.

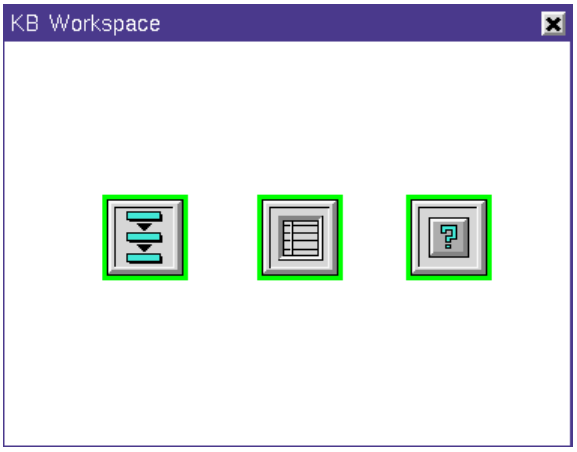
Available Palettes	Description
custom grays from *image-definition*	This value lets you define the G2 palette containing the shades of gray that actually appear in the image-definition you specify. If the image definition does not exist, or its notes attribute is not OK, drawing reverts to standard grays.
custom colors from *image-definition*	This value lets you define the G2 color palette containing the colors that actually appear in the image-definition you specify. If the image definition does not exist, or its notes attribute is not OK, drawing reverts to standard colors.

The black and white and standard choices do not allocate any more colors than G2 is already using. The extended grays and extended colors choices allocate more colors just for images.

The custom grays and custom colors selection provide you with the most control over the G2 color palette (subject to the display limitations) by letting you specify an image file whose colors G2 will use. You could, for instance, create an image file containing a single row of pixels, each pixel being one of the colors you want the G2 palette to contain. By specifying this image file as a custom colors value in the image-palette attribute, the G2 color palette would then contain exactly the colors you wanted it to use. The image itself would not need to be displayed on a workspace for its colors to be in use.

Editing the Color Used for Selection

By default, when you select an item on a workspace and when you add an item to the selection, the selected items appear with a green outline to indicate that they are selected. For example:



You can change the colors that G2 uses for selection by editing the primary-selection-color and secondary-selection-color attributes of the Drawing Parameters system table.

Displaying a Visible Grid on Workspaces

The alignment-grid attribute controls a visible grid and a snap grid on KB workspaces.

The snap grid is disabled by default. To enable it, use the following grammar in the alignment-grid attribute:

grid [, line color: *color*] [, line pattern: *symbol*] [, snap to *grid*]

where *grid* can be either an integer, giving the spacing in workspace units for both X and Y, or a pair on integers (*integer*, *integer*) giving spacings for X and Y. For example:

100, line color: gray, line pattern: long dash, snap to 10

When given as a structure, the syntax is:

structure
(spacing: sequence(*integer*, *integer*),
line-color: *symbol*,
line-pattern: *symbol*,
snap-to: sequence(*integer*, *integer*))

The default value is:

```
structure
(spacing: sequence (50,50),
line-color: the symbol foreground,
line-pattern: the symbol coarse-dot)
```

The visible grid is invisible by default. To view the grid, do one of the following:

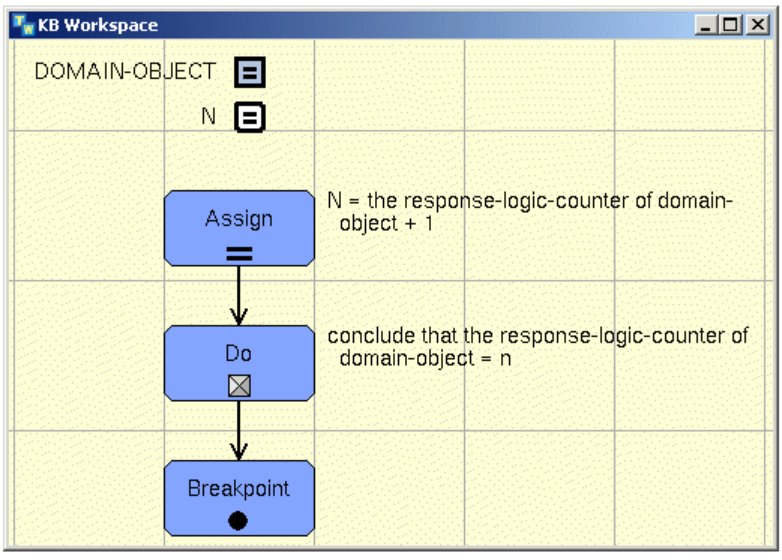
- Choose View > Toggle Visible Grid.
- Enter Ctrl + G with the mouse over a KB workspace.
- Execute the toggle-visible-grid system command, using the g2-system-command system procedure. For details, see g2-system-command in User Interface Operations in the *G2 System Procedures Reference Manual*.
- Set the view-preferences of a KB workspace to visible-grid, or conclude the visible-grid attribute in the view-preferences structure of a KB workspace, for example:

```
conclude that the view-preferences of this workspace =
structure(visible-grid: true)
```

If the snap grid is enabled, and both a constrain moving ... item configuration and the snap grid apply to a particular item, then the item configuration takes precedence and the snap grid is ignored.

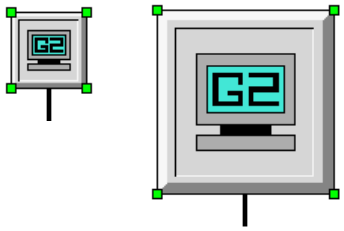
For example, here is a workspace whose alignment-grid is set to:

100, line color: gray, line pattern: short dash, snap to 10

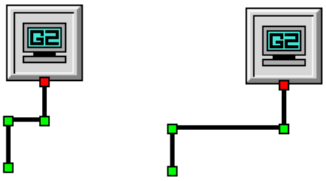


Interactively Resizing Objects and Changing Connection Vertices

The show-selection-handles attribute allows you to interactively resize objects, using selection handles on the object. For example:



You can also interactively change the connection vertices of a connection, using handles on the connection. For example:



When show-selection-handles is true, the default, selection handles appear and the change size menu choice does not appear on items. When show-selection-handles is false, selection handles do not appear and the change size menu choice appears on items.

Class-Specific Attributes of Drawing Parameters

The class-specific attributes of the Drawing Parameters system table are:

Attribute	Description
allow-scheduled-drawing?	Specifies whether drawing is scheduled (yes) or immediate (no).
Allowable values:	yes no
Default value:	yes
Notes:	Immediate drawing (attribute is no) is a superseded capability. For more information, see Appendix F, Superseded Practices.
paint-mode?	Determines the default drawing mode: Paint mode (yes) or XOR mode (no).
Allowable values:	yes no
Default value:	yes
Notes:	XOR mode (attribute is no) is a superseded capability. For more information, see Appendix F, Superseded Practices.
image-palette	Determines the palette of colors to use for drawing background images.
Allowable values:	{black and white | standard colors | standard grays | extended colors | extended grays | custom colors from *image-definition* | custom grays from *image-definition*}
Default value:	standard colors

Attribute	Description
primary-selection-color	The color used as the outline to indicate selection.
Allowable values:	Any color
Default value:	green
secondary-selection-color	The color used as the outline for all subsequent selected items when adding to a selection.
Allowable values:	Any color
Default value:	green
alignment-grid	Controls a visible grid and a snap grid on KB workspaces.
Allowable values:	structure
Default value:	structure (spacing: sequence (50,50), line-color: the symbol foreground, line-pattern: the symbol coarse-dot)
Notes:	See Displaying a Visible Grid on Workspaces.
show-selection-handles	Allows you to interactively resize objects, using selection handles on the object.
Allowable values:	truth-value
Default value:	true
Notes:	See Interactively Resizing Objects and Changing Connection Vertices.

Editor Parameters

The Editor Parameters system table lets you customize some aspects of editing in a KB.

Specifying the Maximum Number of Names to Show

The maximum-number-of-names-in-menus attribute controls the maximum number of names that G2 displays in prompt menus in the Text Editor.

For example, if this attribute has a value of 7 and you are editing a rule and can enter an item name, if *more* than seven names exist in the KB, G2 does not display any names in the text editor. If you type the letter s and there are seven or fewer names that begin with s, G2 displays those names.

Defining the Minimum Text Editor Width

The minimum-width-for-edit-box attribute defines the minimum width at which the Text Editor is displayed.

This attribute is applicable for both editors (scrolling and non-scrolling). The default value for this attribute is 0. By default, the scrolling editor is approximately 500 workspace units wide. Setting this attribute to a value greater than 500 affects the scrolling editor. Setting it to a lesser value has no effect.

Specifying Whether to Enable Author Recording

The author-recording-enabled? attribute specifies whether G2 maintains user information about changes made to items that include the authors attribute. The authors attribute appears in a select number of G2 items. For a full description of the authors attribute, see Using the Authors Attribute.

Edit Operations Menus and Buttons

You can specify whether G2 automatically pops up an edit operations menu or edit operations buttons when you are entering text in the Text Editor. These facilities are controlled by the pop-up-edit-operations-menu and buttons-for-edit-operations attributes.

Controlling the Display of Calling Signatures

By default G2 displays the calling signature of a procedure or function when you enter the procedure or function name followed by the left parenthesis in the Text Editor. You can enable and disable this facility through the show-procedures-signatures attribute.

Displaying the Native Text Editor

The prefer-native-text-editor attribute determines whether to use the native Windows text editor in Telewindows or whether to use the classic G2 Text editor. By default, Telewindows uses the native text editor.

For information on how to use the native text editor, see Editing Text in Using Telewindows in the *Telewindows User's Guide*.

Class-Specific Attributes of Editor Parameters

The class-specific attributes of the Editor Parameters system table are:

Attribute	Description
maximum-number-of-names-in-menus	Controls the maximum number of names that display at one time in the Text Editor menus.
Allowable values:	any positive integer
Default value:	7
object-name-menus-in-upper-case?	Controls whether G2 displays object names in uppercase in Text Editor prompts. When these names are displayed in uppercase, G2 inserts them in text in upper case as well, when you select them.
Allowable values:	{yes | no}
Default value:	no
number-of-spaces-to-insert-on-a-tab	Controls the number of spaces that G2 inserts when you press the Tab key.
Allowable values:	*integer*
Default value:	4

Attribute	Description
maximum-number-of-undos-to-remember	Specifies how many text editing operations G2 remembers and allows you to Undo. G2 allows you to undo any of the available edit operations.
Allowable values:	*integer*
Default value:	100
maximum-number-of-scraps-to-keep	Sets the maximum number of text pieces, or scraps, kept by the G2 text editor as you cut scraps from the text editor window, or copy them into the scrapbook. Use an integer to set the number of scraps kept. If you exceed this limit, the oldest scrap is thrown away.
Allowable values:	*integer*
Default value:	50
minimum-width-for-edit-box	Determines the minimum width of the Text Editor box.
Allowable values:	*integer*
Default value:	0
author-recording-enabled?	Determines whether G2 maintains user information about changes made to items that include the authors attribute.
Allowable values:	{yes | no}
Default value:	yes
pop-up-edit-operations menu	Controls whether the edit operations menu comes up in the Text Editor when text is selected. This menu contains these menu choices: cut, copy, paste, delete, insert, move, and cut and insert.
Allowable values:	{yes | no}
Default value:	yes

Attribute	Description
buttons-for-edit-operations	Controls whether edit operations buttons appear when entering text in the Text Editor.
Allowable values:	{yes | no}
Default value:	yes
show-procedure-signatures?	Determines whether G2 should automatically display the calling signature of a procedure or function when you enter the procedure or function name followed by the left parenthesis in the Text Editor.
Allowable values:	{yes | no}
Default value:	yes
smart-space-insertion	Controls whether or not to insert spaces when pasting copied text in the classic G2 text editor. The default value is yes, which inserts a space before and after the pasted text. To avoid inserting spaces, set the attribute to no.
Allowable values:	{yes | no}
Default value:	yes
prefer-native-editor	Determines the type of text editor to use in Telewindows. The default value is yes, which uses the Windows text editor. To use the classic G2 text editor, set prefer-native-editor to no. You can also configure this attribute when you launch the text editor, using the g2-ui-launch-editor system procedure.
Allowable values:	{yes | no}
Default value:	yes

Fonts

The Fonts system table controls which font G2 uses for attribute tables, statements, free text, editing, and descriptions. G2 supports three font sizes: small, large, and extra-large.

Note The font-for-attribute-tables and font-for-editing attributes of the Fonts System Table only affect the text-editor and attribute-table fonts in the g2-5.x window style. These attributes have no effect on the standard and standard-large window styles. See G2 Window Styles for a discussion of the three window styles.

Class-Specific Attributes of Fonts

The class-specific attributes of the Fonts system table are:

Attribute	Description
font-for-attribute-tables	Controls the size of the font used for the text on attribute tables in the g2-5.x window style.
Allowable values:	{small | large | extra-large}
Default value:	large
font-for-attribute-displays	Controls the size of the font that G2 uses for attribute displays for all window styles.
Allowable values:	{small | large | extra-large}
Default value:	small

Attribute	Description
font-for-statements	Controls the size of the font used for the text in statements for all window styles; thus, this attribute controls how rules, generic formulas, generic simulation formulas, units of measure, and procedures appear. The G2 Simulator, which can use generic simulation formulas, is a superseded capability. For more information, see Appendix F, Superseded Practices.
Allowable values:	{small | large | extra-large}
Default value:	large
font-for-free-text	Controls the size of the font used for free text and borderless free text for all window styles.
Allowable values:	{small | large | extra-large}
Default value:	large
font-for-editing	Controls the size of the font used in Text Editor workspaces in the g2-5.x window style.
Allowable values:	{small | large | extra-large}
Default value:	large
font-for-descriptions	Controls the size of the font that G2 uses in describing an item for all window styles.
Allowable values:	{small | large | extra-large}
Default value:	small

G2 Graphical Language (G2GL) Parameters

For information about G2GL, see G2 Graphical Language (G2GL).

The class-specific attributes of the G2GL Parameters system table are:

Attribute	Description
time-between-time-slice-for-execution-of-thread	An extra amount time that the G2 scheduler should wait between time slices for a given G2GL process to run, in seconds. The default is none, which means no additional waiting time is required.
Allowable values:	*integer*
Default value:	none
break-on-all-execution-faults	Whether to show an individual execution display with appropriately placed breakpoints whenever any kind of fault occurs in a G2GL process, including system-defined faults.
Allowable values:	yes | no
Default value:	no
suppress-unspecified-partner-link-variable-type-faults	Whether to suppress compilation errors and/or execution faults when the type of a partner link variable is not specified. The default value is yes, which means you can create G2GL processes that communicate, without having to specify a partner link type. G2GL does not require partner link types, whereas BPEL does. For more information, see BPEL Compliance.
Allowable values:	yes | no
Default value:	yes
name-of-window-for-g2gl-execution-displays	The name of a g2-window on which to display individual execution displays. The default is none, which uses every logged-in window.
Allowable values:	symbol

Attribute	Description
Default value:	none
default-scale-for-execution-displays	The default scale for individual execution displays.
Allowable values:	float
Default value:	1.0
compile-texts-for-execution-displays	Whether individual execution displays should show text. The default value is no, which omits text from individual execution displays.
Allowable values:	yes | no
Default value:	no
time-between-mini-tracing-steps	The time between mini tracing steps in individual execution displays, in seconds. A mini tracing step includes all but the last step as the thread token moves from one activity to another.
Allowable values:	float
Default value:	0.02
time-between-maxi-tracing-steps	The time between maxi tracing steps in individual execution displays. A maxi tracing step is the last or only step, depending on the mini tracing steps size.
Allowable values:	float
Default value:	0.5
mini-tracing-step-size	The size of each step when tracing is enabled, in workspace units. By setting the mini tracing step to a larger number, all tracing steps become maxi steps.
Allowable values:	integer

Attribute	Description
Default value:	10
g2gl-activity-elbow-room	The minimum distance between a thread token and an activity icon, in workspace units.
Allowable values:	integer
Default value:	2
default-thread-token-class	The class used for the thread token icon in individual execution displays when debugging.
Allowable values:	symbol
Default value:	g2gl-standard-thread-token
default-thread-token-color	The color used for the thread token icon.
Allowable values:	symbol
Default value:	coral

Inference Engine Parameters

The Inference Engine Parameters system table controls computational aspects of the inference engine.

Limiting the Depth of Recursion

The recursion-limit attribute limits the depth of recursion for user-defined functions. This limit does not affect procedures at all. If user-defined functions extend beyond the recursion limit, G2 fails to evaluate the function and displays a level 1 warning message, indicating that the user-defined function is in an infinite recursion or that the recursion limit is too low.

Defining the Timeout for Getting a Variable Value

The timeout-for-variables attribute defines how much time G2 allows before concluding that it has failed to receive a value for a variable. The default is 30 seconds. Specify none if you do not want variables to time out.

When a variable fails to receive a value, G2 invokes all of the whenever rules containing fails-to-receive-a-value statements for that variable. For example, if the variable temperature-sensor-1 fails to receive a value within the time interval that the timeout-for-variables attribute specifies, G2 invokes the following whenever rule for that variable:

```
whenever temperature-sensor-1 fails to receive a value
    post "Temperature-sensor-1 is not responding."
```

Specifying the Timeout for Rule Completion

The timeout-for-inference-completion attribute specifies the amount of time a rule has to complete. If the rule does not complete in this time, it is considered failed. When G2 is evaluating an expression, if a rule cannot be completed immediately (because a variable does not have a current value and G2 cannot immediately get one through backward chaining or data service), the rule goes to sleep. G2 temporarily stops evaluating the expressions in the rule. The variable that needs a value will wake up the expression when the value is available. If the timeout for the rule occurs before the rule awakens, the rule tries one last time to execute, and then completes whether or not it succeeds.

This attribute lets you control when inferencing occurs. If rules did not have timeouts, they could reawaken long after the conditions that cause G2 to invoke them end.

You can override the timeout-for-inference-completion attribute for a particular rule with that rule's timeout-for-rule-completion attribute. For example, if you want one rule to have up to 1 minute to complete, but all other rules to have 30 seconds to complete, set the timeout-for-inference-completion system table attribute to 30 seconds and the special rule's timeout-for-rule-completion attribute to 1 minute.

The default is 30 seconds. Specify none if you do not want rules to time out.

Specifying the Retry Interval for a Variable Value

The retry-interval-after-timeout attribute specifies the number of retries for a variable value. In this context, the term *retry* refers to when G2 checks to see if a variable has received a value after it initially fails to receive one. The value of this attribute determines how long G2 waits before a retry. Note that these guidelines for variable retry also apply to a GSI variable whose data server is gsi-data-server.

When the retry-interval-after-timeout attribute has a time interval value, G2 requests a value for the variable immediately when the variable exceeds the time interval that the timeout-for-variables attribute specifies, and continues to do so every retry interval. For example, if the retry-interval-after-timeout is 2 minutes, and a variable fails to receive a value within the timeout-for-variables interval, G2 sends out an additional request every 2 minutes until the variable receives a value.

Setting this attribute to do-not-retry prevents G2 from retrying a variable.

Specifying the Fuzzy Truth Threshold

The truth-threshold attribute specifies the threshold for fuzzy truth expressions, as described in Producing Fuzzy Truth Values from Relational Operations.

This attribute can have a value from 0 to 1. Fuzzy truth expressions that evaluate to less than the threshold are false, while those that are equal to or greater than the threshold are true. If, for example, the truth-threshold attribute is set to 0.5 and the antecedent to a rule has a truth value of 0.6, the antecedent is true and the rule fires. The default truth threshold is 0.8.

Class-Specific Attributes of the Inference Engine Parameters

The class-specific attributes of the Inference Engine Parameters system table are:

Attribute	Description
recursion-limit	Limits the depth of recursion for user-defined functions.
Allowable values:	*integer*
Default value:	50
timeout-for-variables	Determines how much time G2 allows before concluding that it is has failed to receive a value for a variable.
Allowable values:	{*time-interval* | none}
Default value:	30 seconds

Attribute	Description
timeout-for-inference-completion	Specifies the amount of time a rule has to complete.
Allowable values:	*time-interval*
Default value:	30 seconds
retry-interval-after-timeout	Tells G2 how often to retry for a value after a variable fails to receive one.
Allowable values:	{*time-interval* | do not retry}
Default value:	1 minute
truth-threshold	Determines the threshold for fuzzy truth expressions.
Allowable values:	{*number* true | true | false}
Default value:	.800 true

KB Configuration

The KB Configuration system table acts as the root of the configuration hierarchy that operates within the workspace hierarchy.

Specifying Item Configurations for the KB

The item-configuration attribute determines the default configurations for the entire KB. The defaults are as follows:

```
configure the user interface as follows:
unless in administrator mode:

    attributes visible for item exclude additionally: item-configuration;
    attributes visible for kb-restrictions exclude:
        main-menu-user-restrictions, keyboard-command-restrictions,
        initial-g2-user-mode-for-this-kb;
    menu choices for item exclude additionally: describe-configuration
```

You can specify any appropriate configurations that your KB may require. For more information about available configurations, see Configurations.

Restricting Main Menu Options

The main-menu-user-restrictions attribute lets you specify which Main Menu choices, if any, you wish to restrict. For example, you may want your KB to exclude the change mode menu option in all user modes. Enter this statement to accomplish this:

```
unless in administrator mode: main menu choices
    exclude absolutely: change mode
```

Providing or Restricting Global Keyboard Commands

Use the keyboard-command-restrictions attribute to restrict global or workspace commands (such as center-origin). An example is:

```
when in proprietary mode:
    global keyboard commands exclude: center-origin
```

Note You cannot restrict commands that begin with the Control key, such as Control + y to display the login dialog.

Setting the Initial User Mode for a KB

You can specify the initial user mode for a KB by entering the mode as the value of the initial-g2-user-mode-for-this-kb attribute.

Noting Your Optional Modules

The authorized-optional-modules attribute lists the license type, license option, and any optional modules from your authorization file (`g2.ok`) that are currently available for your machine. For instance, if you purchased an Online license with a Developer's option, the value of this attribute will be:

```
online
```

You cannot change this attribute.

Simulating Optional Modules

The simulated-optional-modules attribute lets you simulate an optional module less powerful than the one for which you are licensed.

Typically, you will use this facility to test KB behavior under the license with which you intend to deploy the application. Simulating an optional license module remains in effect until you change this attribute value. A simulation mode is not saved with a KB.

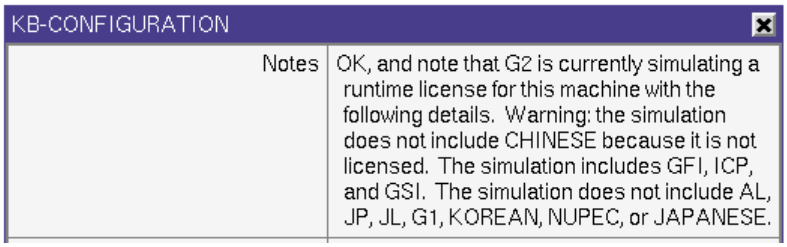
You can simulate any license option less powerful than your own licensing options or optional modules that you have purchased. For instance, if you are developing a KB using a G2 Online license, with the Development option, you can simulate all of the other less-powerful options: restricted, runtime, or embedded. Simulating a less powerful license option does not prevent you from accessing the KB Configuration system table so that you can revert to your license option.

The possible values for this attribute include all of the optional modules that you can purchase or include with G2, and the various types of licenses available that are less powerful than your own license option.

Note For a description of license types, license options, and optional modules available for G2, see your Gensym representative for a copy of the latest price list.

While you can enter more than one selection from the text editor (such as japanese and runtime), some combinations of choices are invalid. For example, even though the text-editor permits such an entry, it does not make sense to enter two values like restricted-use and embedded, because you can simulate only one optional module at a time.

When you enter an optional module to simulate, the notes attribute of the KB Configuration system table indicates exactly what license is being simulated, shown next, for example:



Note Some Gensym internal-use only option names appear in the notes attribute during license simulation.

Class-Specific Attributes of KB Configuration

The class-specific attributes of the KB Configuration system table are:

Attribute	Description
item-configuration	The KB-level configuration statements.
Allowable values:	For a complete description of using configuration statements, refer to Configurations.
Default value:	SeeSpecifying Item Configurations for the KB.
main-menu-user-restrictions	Lets you restrict all menu choices on the Main Menu. The default is none.
Allowable values:	For a complete description of using configuration statements, refer to Configurations.
Default value:	none
keyboard-command-restrictions	Lets you exclude or include global keyboard commands while in a user mode.
Allowable values:	For a complete description of using configuration statements, refer to Configurations.
Default value:	none
initial-g2-user-mode-for-this-kb	Specifies a default user mode for the KB. The default is none, which means that the user is in administrator mode (the only system-defined user mode).
Allowable values:	For a complete description of user modes, refer to Configurations.
Default value:	none

Attribute	Description
authorized-optional-modules	The modules for which your G2 process is authorized.
Allowable values:	none, icp, g1, offline, online, runtime, restricted-use, embedded, japanese, korean, gfi, gsi GFI is a superseded capability. For further information, see Appendix F, Superseded Practices.
Default value:	A list of the current license modules.
simulated-optional-modules	The optional module(s) to simulate.
Allowable values:	do not simulate, none, icp, offline, online, runtime, restricted-use, embedded, japanese, korean, gfi, gsi GFI is a superseded capability. For further information, see Appendix F, Superseded Practices.
Default value:	do not simulate

Language Parameters

The Language Parameters system table lets you set the current language for a KB. The current language may be different than the default language, as described under Setting the Current Language.

Specifying the Current Language

The current-language attribute specifies the default language for a KB. This language can be overridden by users accessing G2 through Telewindows and either specifying another language as a command-line option, or changing the default language for the G2 window.

For a description of using command-line options, see Appendix A, Launching a G2 Process. For an explanation of specifying a language for a G2 window, see Supporting a Window-Specific Language.

Using a Text-Conversion-Style

You can specify the name of a text-conversion-style item in the attribute of the same name.

For a complete description of text-conversion-style items, see Working with Text Conversion Styles.

Class-Specific Attributes of Language Parameters

The class-specific attribute of the Language Parameters system table is:

Attribute	Description
current-language	Specifies which language to use by activating a set of predefined menu translations within a KB.
Allowable values:	{*symbol* | english | russian | japanese | korean}
Default value:	english
text-conversion-style	Specifies the text-conversion-style item to use for the KB.
Allowable values:	*text-conversion-style*: symbol
Default value:	none

Logbook Parameters

The Logbook Parameters system table controls the size, number, and behavior of the operator logbook pages.

The first seven attributes of this system table after item-configuration and the spacing-between-entries attribute are expressed in workspace units, which is one pixel when the workspace is scaled to full size, and proportionally larger or smaller when the workspace is scaled up or down.

Defining the Logbook Page Size

The width-for-pages and height-for-pages attributes defines the size of the logbook pages. The default values are 345 and 400, respectively. Entering a lower value in either of these attributes reduces the page size, while entering a higher number increases it.

Specifying the Margin for Logbook Messages

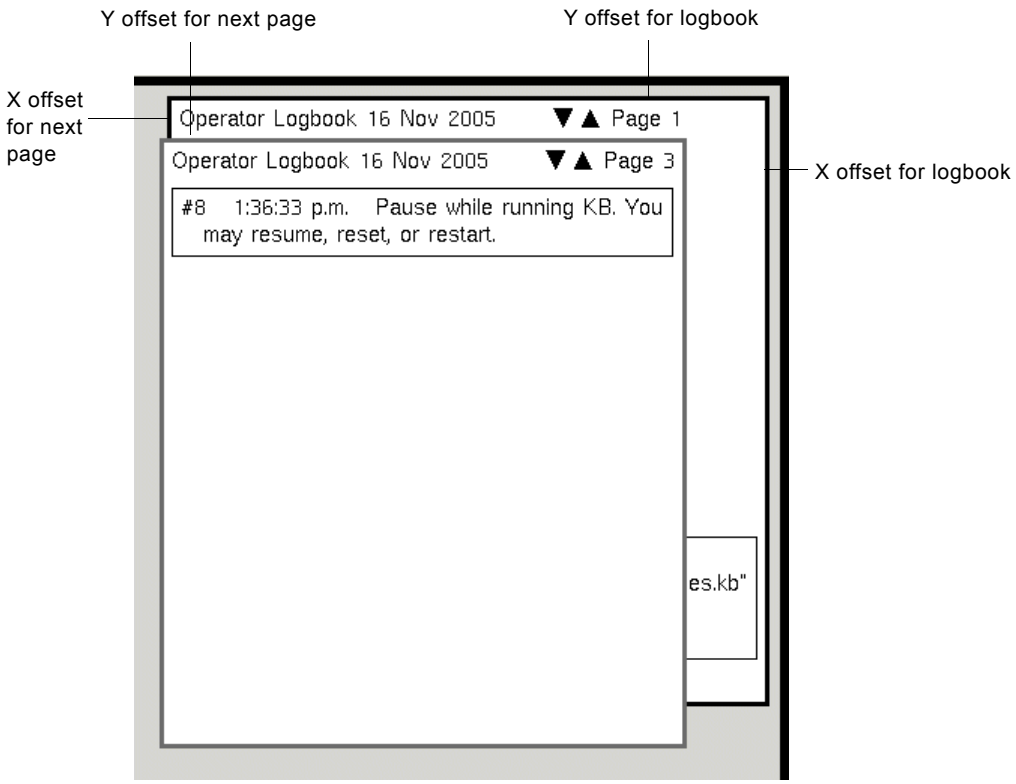
The margin-for-pages attribute specifies the amount of space left at the edge of messages upon each logbook page.

Defining Where to Position Logbook Pages

The x-offset-for-next-page and y-offset-for-next-page attributes defines where G2 positions each page of the logbook in relation to the previous page.

Specifying Where to Position the Logbook

The x-offset-for-logbook and y-offset-for-logbook attributes specifies where G2 positions each logbook in relation to the G2 window. This diagram illustrates several of the logbook options:



Controlling How Many Logbook Pages to Show

The maximum-number-of-pages-to-show attribute controls the number of logbook pages that G2 keeps visible on the screen. G2 always shows at least a small part of each of these pages on the screen, even if you try manually to move them off the screen.

By selecting the up and down arrows at the top of any logbook page, you can turn the logbook pages that are visible, as well as those that are not visible but in memory. When you reach the maximum number of pages, G2 hides the oldest pages, which are those with the smallest numbers.

Controlling the Number of Logbook Pages

The maximum-number-of-pages-to-keep-in-memory attribute controls the number of logbook pages that G2 keeps in memory, including the pages that are currently visible at any time.

By selecting the up and down arrows at the top of any logbook page, you can flip through the logbook pages that are visible and those that are not visible but in memory. When you reach the maximum number of pages, G2 discards the oldest pages, which are those with the smallest numbers.

The number-of-pages-to-shed-at-limit attribute determines how many logbook pages to discard when the maximum-number-of-pages-to-show limit has been reached.

Note The value of maximum-number-of-pages-to-keep-in-memory should be always equal or larger than maximum-number-of-pages-to-show, otherwise G2 will forcedly align maximum-number-of-pages-to-keep-in-memory with the value of maximum-number-of-pages-to-show to be able to hold all shown pages in memory.

Displaying the Native Logbook

The prefer-native-logbook attribute determines whether to use the native G2 logbook in Telewindows or whether to use the G2 classic logbook. By default, the G2 Operator Logbook appears in a Windows docking pane when viewed through Telewindows and is docked to the upper-right corner of the overall window.

Note The default value is yes, except when loading KBs saved in G2 Version 8.1 or earlier, in which case the value is no.

Note The native logbook is only supported in Telewindows Next Generation (`twng.exe`).

If the lift-logbook-to-top-when-new-pages-are-added? attribute in the Logbook Parameters system table is no and prefer-native-logbook is yes, the native logbook

is initially hidden. When lift-logbook-to-top-when-new-pages-are-added? is yes, the native logbook is initially visible and is shown whenever a message is added.

The native logbook pane accepts the following keyboard and mouse commands:

- Left click — Select message.
- Right click — Display message menu.
- Left drag on unselected area — Select text region.
- Left drag on selected text — Drag and drop text to another application, such as Word.
- Control + C — Copy selected text.
- Control + A — Select all text.
- Tab — Select next message.
- Shift + Tab — Select previous message.
- Escape — Deselect all.
- Control + - (minus) — Zoom out.
- Control + + (plus) — Zoom in.
- Control + 0 — Normal zoom.
- PageUp, PageDown, Home, End, UpArrow, DownArrow — Scroll the view.

Include Date in Messages

The include-date-in-messages attribute controls whether to include date in logbook messages. If yes, each message will have current date (year, month, day) included. If no, only current time (hour, minute, second) is included.

Default Docking Position

The default-docking-position attribute controls the default docking position (top, bottom, left, right) of the logbook, by default the docking position is right.

Class-Specific Attributes for Logbook Parameters

The class-specific attributes of the Logbook Parameters system table are:

Attribute	Description
width-for-pages	Determines the logbook page width

Attribute	Description
Allowable values:	*integer*
Default value:	345
height-for-pages	Determines the logbook page height.
Allowable values:	*integer*
Default value:	400
margin-for-pages	Determines the amount of space around the messages that appear on the logbook.
Allowable values:	*integer*
Default value:	5
x-offset-for-next-page	Determines where new pages are horizontally positioned relative to the previous page.
Allowable values:	*integer*
Default value:	-5
y-offset-for-next-page	Determines where new pages are vertically positioned in relation to the previous page.
Allowable values:	*integer*
Default value:	-28

Attribute	Description
x-offset-for-logbook	Determines where the logbook is horizontally positioned in relation to the G2 window.
Allowable values:	*integer*
Default value:	10
y-offset-for-logbook	Determines where the logbook is vertically positioned in relation to the G2 window.
Allowable values:	*integer*
Default value:	-10
maximum-number-of-pages-to-show	Controls the number of logbook pages that G2 keeps visible on the screen.
Allowable values:	*integer*
Default value:	3
number-of-pages-to-shed-at-limit	Controls the number of logbook pages that G2 discards when the maximum number of pages to show has been reached.
Allowable values:	*integer*
Default value:	1
spacing-between-entries	Controls the amount of vertical spacing between message units. Changes to this parameter do not affect existing messages.
Allowable values:	*integer*
Default value:	10

Attribute	Description
log-inform-messages?	Determines whether inform messages appear on both the message board and the logbook or just the message board. When set to no, messages appear only on the message board.
Allowable values:	{yes | no}
Default value:	no
maximum-number-of-pages-to-keep-in-memory	Controls the number of logbook pages that G2 keeps in memory, including the pages that are currently visible at any time.
Allowable values:	*integer*
Default value:	200
lift-logbook-to-top-when-new-pages-are-added?	Specifies how G2 adds new pages to the logbook. If yes, G2 stacks all of the existing logbook pages on top of each other and puts the new page on top of the stack. If no, G2 staggers the pages so that you can see a part of each one.
Allowable values:	{yes | no}
Default value:	yes
prefer-native-logbook	Specifies whether to use the native Windows logbook or the classic G2 Operator Logbook. If yes, displays the G2 Operator Logbook in a Windows docking pane when viewed through Telewindows. If no, display the classic G2 Operator Logbook.
Allowable values:	{yes | no}
Default value:	yes
include-date-in-messages	Specifies whether to include date in logbook messages. If yes, each message will have current date (year, month, day) included. If no, only current time (hour, minute, second) is included.

Attribute	Description
Allowable values:	{yes | no}
Default value:	no
default-docking-position	Specifies the default docking position of the logbook.
Allowable values:	{top | bottom | left | right}
Default value:	right

Log File Parameters

The Log File Parameters system table lets you write logbook messages to a file, called a **log file**. Messages in the log file have the same format as those on the logbook page. Although logbook page headers are not included in the file, you can infer the date of the message from the write date of the file.

Saving a Log File

The log-file-enabled? attribute specifies whether logbook messages should be written to a log file. Setting this option to yes immediately creates a log file, but does not begin writing to the file. Instead, it buffers data that will be written to the file when this option is set back to no.

When this option is yes, G2 begins to buffer messages, regardless of whether G2 is running or whether the tracing-and-debugging-enabled? attribute of the debugging-parameters system table is set to yes. If you reset, restart, or pause G2, logbook messages are still buffered as long as log-file-enabled? remains yes.

When you change the option to no, G2 automatically writes the buffered information to the log file and closes the current log file. Subsequently setting this option to yes and then to no appends interim data to the original log file, until G2 reaches the limit set in the when-to-close-current-log-file-and-open-next-one attribute, described in Defining When to Close a Log File. If an error occurs in locating or writing to the log file, G2 signals an error and sets log-file-enabled? to no.

Specifying the Log File Directory Location

The directory-for-log-files attribute specifies the directory in which G2 writes log files. The default value, default, indicates the directory from which you loaded the current KB. If you have not started a KB, the default directory is the one from which you started G2. Enter a directory name as a text value and complete the pathname with a closing path delimiter as in:

```
"c:\myname\test-kbs\"
```

```
"/home/myname/test-kbs/"
```

You cannot edit this attribute if the log-file-enabled? attribute is yes.

Specifying a Log File Root Name

The root-name-for-log-files attribute specifies the naming convention G2 uses when generating each log file and its version number. You can enter a fully qualified pathname, or a prefix file name as follows:

Value	Description
"g2-log-"	Specifies that G2 prefaces the log file name with: "g2-log-" and places the file in the directory specified in the directory-for-log-files attribute.
file pathname	Must be a string specifying that the name contains a root name (given by you) and a positive number. The number indicates the number of files already written plus the current file. It has the form: root-name number-of-files-written For example, mylog2 has root name mylog, and the file is the second of two files written.

You cannot edit this attribute if the log-file-enabled? attribute is set to yes.

Specifying the Current Log File

The current-log-file attribute displays the name of the current log file. You cannot edit this attribute.

G2 selects the current log file according to the following criteria:

- If no log files have yet been written, using the current root and directory names, the current file has the form *root-name1*.
- If log files have already been written, using the current root and directory names, the current file has the form:

root-name number-of-files-already-written + 1

When G2 reaches the limit specified in the maximum-number-of-log-files attribute, the next file is: root name1. Subsequent log files have the form shown in the second bullet. Thus the original files are overwritten.

Defining When to Close a Log File

The when-to-close-current-log-file-and-open-next-one attribute controls when to close the current log file and create the next file. You can specify that G2 should close the log file after:

- A specific number of messages.
- A given time interval since the file was opened.
- A specific number of messages *or* a given time interval since the file was opened, whichever occurs first.

Examples are:

after 3 minutes
after 100 messages
after 3 minutes or 100 messages, whichever comes first
after 100 messages or 5 minutes, whichever comes first

However, when you set the log-file-enabled? attribute to no, G2 closes the current log file automatically, regardless of whether the criteria you set has been met. If you then reset log-file-is-enabled? to yes without changing the directory and root names, the current log file remains unchanged. G2 then appends the succeeding messages to the end of the existing log file.

Defining When to Back Up Log Files

The when-to-back-up-current-log-file-other-than-when-closing attribute controls when G2 should back up the current log file other than when it closes the file. Backing up closes and reopens the file for appending. You can specify that G2 should back up the log file after a specific number of messages, or after a given time interval since the file was opened, or both, in either order, whichever occurs first. Here are examples:

after 3 minutes
after 100 messages
after 3 minutes or 100 messages, whichever comes first
after 100 messages or 5 minutes, whichever comes first

Class-Specific Attributes of Log File Parameters

The class-specific attributes of the Log File Parameters system table are:

Attribute	Description
log-file-enabled?	Determines whether logbook messages are written to a log file.
Allowable values:	{yes | no}
Default value:	no
directory-for-log-files	Specifies the directory in which G2 writes log files.
Allowable values:	any directory path name as a string default
Default value:	default
root-name-for-log-files	Specifies the naming convention for log files.
Allowable values:	any directory or file path name as a string
Default value:	"g2-log-"
current-log-file	The name of the current log file.
Allowable values:	current log file name none
Default value:	none

Attribute	Description
when-to-close-current-log-file-and-open-next-one	Controls when to close the current log file and create the next.
Allowable values:	See Defining When to Close a Log File.
Default value:	after 100 messages or 1 day, whichever comes first
when-to-back-up-current-log-file-other-than-when-closing	Controls when G2 should back up the current log file other than when it closes the file.
Allowable values:	See Defining When to Back Up Log Files.
Default value:	never
maximum-number-of-log-files	Specifies the maximum number of log files with the specified root name that G2 can write to the specified directory. After G2 reaches the maximum number of files, the earliest files are overwritten as needed.
Allowable values:	any positive integer less than 1000 none
Default value:	10

Menu Parameters

The Menu Parameters system table controls how menus are displayed in G2 and how the menu selections appear within the menu box.

Specifying How to Align Menu Choices

The alignment-for-menu-choices attribute specifies how G2 displays menu selections. The left value specifies a left-justified display, right specifies a right-justified display, and center specifies a centered display. The default for this attribute is left.

Note The alignment-for-menu-choices attribute only affects the alignment of menu choices in the g2-5.x window style. This attribute has no effect on the standard and standard-large window styles. See G2 Window Styles for a discussion of the three window styles.

Allowing Multiple Menus to Display

The when-to-allow-multiple-menus attribute determines whether you can display on a workspace more than one copy of the same menu at a time, or whether you can display more than one menu at a time as follows:

This value...	Causes G2 to...
always	Allow as many copies of a menu as you want to position on the workspace by clicking with your mouse.
never	Allow one copy of a menu for one purpose to be displayed at any given time.
for different selections	Display more than one menu at a time if the menus are for different items. For example, you can display the Main Menu and the Logbook Page menu at the same time.

Allowing Walking Menus

The walking-menus? attribute determines whether G2 displays walking menus. When the menu attribute is set to yes, G2 lets you choose from submenus by dragging the mouse, leaving the original menu visible. When the attribute is set to no, you must click on a choice and dismiss the original menu to display a submenu.

The walking-menus? attribute defaults to a value of no for KBs created in previous versions of G2.

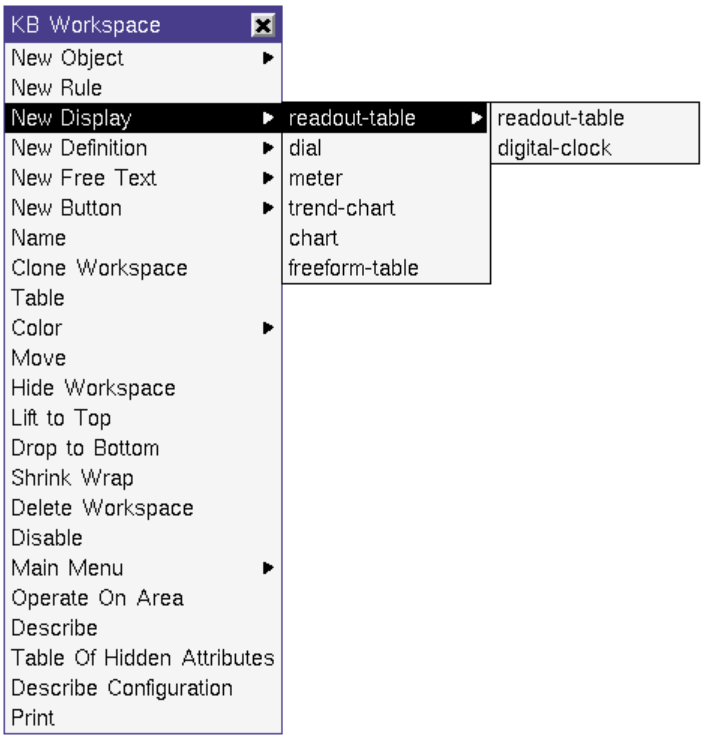
The next figure shows part of the KB Workspace menu with two submenus. This display results from:

1 Opening the KB Workspace menu.

2 Dragging the mouse pointer to the right-hand portion of the New Display menu choice. This causes G2 to display the New Display submenu.

3 Dragging the mouse pointer to the readout-table menu choice. This causes G2 to display the next choose a class submenu.

This figure shows walking menus in standard window style. The standard window styles do not have a title bar on submenus.



Controlling the Display of Developer Menu Bar

The automatically-show-developer-menu-bar attribute allows you to control when the developer menu bar appears in Telewindows. The default value is on pause, reset, or initial connection, which displays the developer menu bar when the KB is paused or reset, and when the initial connection is to G2 occurs. The other options are: on, which always shows the developer menu bar, and never, which never shows it.

Class-Specific Attributes of Menu Parameters

The class-specific attributes of the Menu Parameters system table are:

Attribute	Description
alignment-for-menu-choices	Specifies how G2 displays menu selections.
Allowable values:	{left | right | center}
Default value:	left
when-to-allow-multiple-menus	Determines whether you can display more than one menu at a time.
Allowable values:	{never | always | for difference selections }
Default value:	never
walking-menus?	Determines whether G2 has walking menus.
Allowable values:	{yes | no}
Default value:	yes
automatically-show-developer-menu-bar	Determines whether the developer menu bar appears in Telewindows.
Allowable values:	on pause, reset, or initial connection | on | never
Default value:	on pause, reset, or initial connection

Message Board Parameters

The Message Board Parameters system table lets you control the width of the message board, its height, the amount of spacing between entries, the maximum number of entries, and whether or not G2 highlights new messages.

These are the class-specific attributes of the Message Board Parameters system table. The values of the first four attributes after item-configuration are expressed in workspace units. A workspace unit is one pixel when the workspace is scaled to full size, and proportionally smaller when the workspace is scaled down. Changes to this system table do not affect existing items.

Defining the Minimum Display Interval

The minimum-display-interval attribute defines the length of time that the message appears. If the validity interval of the antecedent of the rule is longer than the value of this attribute, that is how long the message displays. The default is indefinite.

If you set the minimum-display-interval to 0, messages appear only for as long as they are true. However, since the validity interval for some messages can be very short, you will probably want to set a minimum display interval to give the operator time to read the message.

Displaying the Native Message Board

The prefer-native-message-board attribute determines whether to use the native G2 Message Board in Telewindows or whether to use the classic G2 Message Board. By default, the G2 Message Board appears in a Windows docking pane when viewed through Telewindows and is docked to the upper right corner of the overall window.

Note The default value is yes, except when loading KBs saved in G2 Version 8.1 or earlier, in which case the value is no.

Note The native message board is only supported in Telewindows Next Generation (`twng.exe`).

The native Message Board pane accepts the following keyboard and mouse commands:

- Left click — Select message.
- Right click — Display message menu, which includes the go to message origin menu choice.

- Left drag on unselected area — Select text region.
- Left drag on selected text — Drag and drop text to another application, such as Word.
- Control + C — Copy selected text.
- Control + A — Select all text.
- Tab — Select next message.
- Shift + Tab — Select previous message.
- Escape — Deselect all.
- Control + - (minus) — Zoom out.
- Control + + (plus) — Zoom in.
- Control + 0 — Normal zoom.
- PageUp, PageDown, Home, End, UpArrow, DownArrow — Scroll the view.

Class-Specific Attributes of Message Board Parameters

The class-specific attributes of the Message Board Parameters system table are:

Attribute	Description
initial-width-of-message-board	Controls the initial width of the message board. The width can change to accommodate long messages.
Allowable values:	*integer*
Default value:	345
initial-height-of-message-board	Controls the initial height of the message board. The height changes if it needs to show all of the current messages.
Allowable values:	*integer*
Default value:	400

Attribute	Description
spacing-between-entries	Controls the amount of vertical spacing between messages.
Allowable values:	*integer*
Default value:	10
maximum-number-of-entries	Controls the maximum number of messages that can appear on a message board. After G2 reaches the limit set by this attribute, it deletes the oldest message to make room for each new message.
Allowable values:	*integer*
Default value:	10
highlight-new-messages?	Controls whether G2 highlights a new message. If yes, G2 highlights each new message for the first second that it appears on the message board. If no, G2 does not highlight new messages.
Allowable values:	{yes | no}
Default value:	yes
minimum-display-interval	Indicates how long the message should appear on the message board after an inform action.
Allowable values:	{*time-interval* | indefinite}
Default value:	indefinite

Attribute	Description
prefer-native-message-board	Specifies whether to use the native Windows message board or the classic G2 Message Board. If yes, displays the G2 Message Board in a Windows docking pane, which is docked to the upper-right corner of the overall window. If no, display the classic G2 Message Board.
Allowable values:	{yes | no}
Default value:	yes

Miscellaneous Parameters

The Miscellaneous Parameters system table lets you control various aspects of the KB.

Defining Whether to Repeat the Random Function

The repeat-random-function-on-reset? attribute defines whether G2 shuffles the function upon a KB reset. If the attribute is set to yes, G2 does nothing and the random function returns the same sequence of random numbers after each reset, if it has the same argument.

If this attribute is set to no, G2 seeds the random function so that it returns a different sequence of random numbers after each reset.

Specifying the Workspace Margin

The initial-margin-for-workspaces attribute specifies how close you can place icons at the edge of a workspace. The margin must be a non-negative integer value, measured in workspace units. The smaller the integer, the closer to the edge of the workspace you can place an icon.

Starting a KB Automatically After KB Load

The start-kb-after-load? attribute determines whether G2 starts the KB whenever it is loaded. G2 automatically starts the KB if this attribute is set to yes and displays an operator logbook message indicating that the KB has been started because of the system table setting.

The Load KB option never start afterwards overrides the start-kb-after-load? setting. For more information on this option and others, see Selecting Options when Loading a KB File.

Determining the KB Run State

The g2-run-state attribute determines the current run state of the KB. It has a symbol value which can be reset, running, or paused.

KB developers can query the value of this attribute to determine the current run state of the KB, for example:

```
initially inform the operator that
    "G2 is [the g2-run-state of miscellaneous-parameters]"
```

Enabling the Explanation Facilities

Specifying yes for the enable-explanation-controls attribute enables you to:

- Statically display one level of forward and backward chaining for a variable.
- Dynamically display:
  - All invocations of backward-chaining rules for a variable.
  - All invocations of rules for an item that contain a generic reference to that item.
  - All invocations of a particular rule.
- Cache explanation data for variables, parameters, and rules and create explanation items that display the data on explanation trees.

Determining Connection Caching

The connection-caching-enabled? attribute determines whether graphical connections between items should be cached.

G2 caches connections when this attribute is set to yes. Caching makes expressions that reference connections execute faster, but causes connection changes to take longer. When this attribute is set to no. G2 does not cache connections and connection expressions take longer, but changing connections is faster.

Either behavior may be preferable, depending on your particular application. See Controlling Connection Caching for more information.

Determining Connection Inactivity

The dead-connection-timeout attribute lets you configure the amount of seconds necessary to declare a Telewindows client that is not responding to G2 server as dead (inactive). This situation could happen due to network problems. Valid values for this parameter are positive integers greater than 0 and default value is 200.

It is possible to close such connections and free resources by setting disconnect-dead-connections? attribute value to yes.

It is important to remark that the condition for inactive connections is evaluated each 30 seconds. This means that although we set a dead-connection-timeout value of 10 seconds it could take up to 30 seconds more to mark it as dead.

Changing the Backward Compatibility

The backward-compatibility-features attribute lets you revert certain changes made in G2 since previous versions.

The changes that you can revert by completing this attribute are:

- ignore duplicate list element error
- extra vertices in g2-get-connection-vertices
- inconsistent behavior of move in configurations

Tip By default, loading a KB created by an earlier version of G2 changes the value of this attribute to include both of these options.

Ignoring Duplicate List Element Error

The ignore-duplicate-list-element-error value causes G2 to disregard a change made to the insert action. The change to the action causes G2 to signal an error any time an attempt is made to insert duplicate elements into a list that disallows them.

List items can allow or disallow duplicate elements. In previous G2 releases, attempting to insert a duplicate element into list items that disallowed them caused G2 to signal an error *unless* the insert action specified an element location of either:

- at the beginning of the list.
- at the end of the list.

Attempting to insert duplicate elements into a list that disallows them now causes G2 to signal an error consistently. This change of behavior can affect existing KBs.

Entering the value ignore duplicate list element error essentially reverses the change to the insert action for list elements to its previous behavior. Use this value if your KB relies on the previous behavior.

Returning Additional Connection Vertices

In previous G2 versions, the g2-get-connection-vertices system procedure returned the exact number of vertices of which a connection consisted. The purpose of this system procedure is to populate a list with the connection vertices

of an existing connection, and then to use that list with the create connection action. The create action, however, does not require or expect the exact number of vertices. Instead, it requires only a minimum number of vertices. To recreate a connection, the create connection action determines the last one or two vertices from the position of the item to which a connection is being joined.

The g2-get-connection-vertices system procedure currently returns the minimum number of vertices that the create connection action requires. However, existing KBs may rely on the previous behavior or having the system procedure return the exact number of vertices. Specifying the value:

extra vertices in g2-get-connection-vertices

in the backwards-compatibility-features attribute causes the system procedure to behave as it did in previous releases.

For more information about...	See...
Inserting elements into lists	Inserting into Lists with Duplicate Elements
Creating connections using the g2-get-connection-vertices system procedure	Creating an Existing Connection Programmatically
The g2-get-connection-vertices system procedure	*G2 System Procedures Reference Manual*

Configuring “Implies Move” for Workspaces

In previous releases, configuring the item configuration of a workspace as selecting any item implies move resulted in inconsistent behavior, depending on the selected item. If the selected item restricted the move menu choice, then selecting the item moved the workspace rather than the item. In general, all items that are transferable, that is, all items that have the transfer menu choice, as well as all connections, exhibit this behavior, whereby selecting the item moved the workspace instead of the item.

The current version of G2 changes the behavior when the item configuration of a workspace is configured as selecting any item implies move. Now, selecting any item moves the item, not the workspace, regardless of whether the move menu choice has been restricted for the item.

To revert to the previous behavior, add the following option to the backward-compatibility-features attribute:

inconsistent behavior of move in configurations

Displaying the Native G2 Login and Change Mode Dialogs

The prefer-native-login-dialog attribute determines whether to use the native Windows G2 Login and Change Mode dialogs in Telewindows or whether to use the classic G2 dialogs. By default, Telewindows uses the native dialogs.

Confirming Run State Changes

The confirm-run-state-changes attribute determines whether G2 posts a confirmation dialog for any attempt to start, restart, reset, resume, or pause G2. The dialog is posted on the window where the request was made. The default is no.

Use Unicode for Filenames

The use-unicode-for-filenames? parameter value yes makes G2 system procedures that deals with file operation capable of using g2-strings to specify filenames. This option should always work for supported Windows and usual Linux configurations, allowing the user to call these functions with Unicode characters:

```
file-exists = call g2-file-exists ("c:\日本.txt")
```

The value no means that characters in the filename will be 8-bit and implies the user will have to know the encoding used by the OS for non-ascii character filenames:

```
file-exists = call g2-file-exists(export-text("c:\日本.txt", sjis ))
```

The above example will not work if the encoding in the file system is not SJIS (Shift-JIS).

Class-Specific Attributes of Miscellaneous Parameters

The class-specific attributes of the Miscellaneous Parameters system table are:

Attribute	Description
repeat-random-function-on-reset?	Controls whether the random function is scrambled when G2 is reset.
Allowable values:	{yes | no}
Default value:	no
initial-margin-for-workspaces	Controls the size of the margin for workspaces.
Allowable values:	*integer*
Default value:	30
start-kb-after-load?	Controls whether G2 is started immediately after loading a KB. If this attribute is set to yes, G2 is started after loading the KB.
Allowable values:	{yes | no}
Default value:	no
g2-run-state	Determines the run state of the current KB.
Allowable values:	{reset | running | paused}
Default value:	reset

Attribute	Description
backward-compatibility-features	Lets you disregard certain changes made in recent G2 releases.
Allowable values:	{none | ignore duplicate list element error | extra vertices in g2-get-connection-vertices}
Default value:	none
show-uuids-in-attribute-tables	Controls whether the uuid attribute of all items should be displayed on attribute tables. By default, the value of this attribute is no, and G2 displays only the uuid attributes of items that inherit from unique-identification class.
Allowable values:	{yes | no}
Default value:	no
enable-explanation-controls	Enables the explanation facilities which statically and dynamically display the invocation of rules for variables and parameters.
Allowable values:	{yes | no}
Default value:	no
connection-caching-enabled	Determines whether graphical connections are cached.
Allowable values:	{yes | no}
Default value	no
prefer-native-login-dialog	Determines the type of G2 Login and Change Mode dialogs to use in Telewindows. The default value is yes, which uses the Windows dialogs. To use the classic G2 dialogs, set prefer-native-login-dialog to no.
Allowable values:	{yes | no}

Attribute	Description
Default value:	yes
confirm-run-state-changes	Determines whether G2 posts a confirmation dialog for any attempt to start, restart, reset, resume, or pause G2.
Allowable values:	{yes | no}
Default value:	no
float-to-text-default-format	Determines the float-to-text format. default is compatible with G2 8.x. For more explanation on float formats, please refer to system procedure g2-float-to-text in *G2 System Procedures Reference Manual*
Allowable values:	{default | float | exponent | best | force-zero}
Default value:	default
float-to-text-default-precision	Specifies either the number of digits to the right of the decimal point, or the significant digits, depending on the output-format value (not applicable in default and force-zero format).
Allowable values:	0 - 16
Default value:	3
allow-only-one-table-display-for-item?	Determines whether G2 should use only one display for each item per window. By default, showing the table of an item will always bring new tables. Using this new option, now exist opened tables were reused if user tried to show the table of an item.
Allowable values:	{yes | no}
Default value	no

Module Information

The Module Information system table lets you define a top-level module for your KB.

Modules are a convenient method of saving small KBs that typically contain distinct and manageable pieces of a larger KB's knowledge. For example, in a development environment with several G2 developers, one developer could be creating the class hierarchy, while another was creating procedures and methods for those classes. Using modules, each developer could save his or her work in a module, and then, using a top-level module, combine the modules into a single, modularized KB.

A Module Information system table exists for each module you create, because every module has associated with it a unique set of system tables. You can install only one set of system tables, and therefore only a single Module Information system table, in a KB at one time.

For a complete description of using modules, and the role that system tables play within them, see Modules and System Tables.

Specifying a Module File Name

The module-file-name attribute specifies the pathname of a KB file in which to save the module. When the value is default, the module file name is the value of the top-level-module attribute with a `.kb` extension, which G2 saves in the directory specified when you save the KB.

When the value is other than default, it should be a file name to use when the module is saved. Note that this file name is not synchronized with the module name, which is generally not recommended. Therefore, we recommend that you use the default value, which is default.

If you want to override the module file name with a name other than the top-level module name, specify a relative or fully qualified path name as the name, entered as text in quotation marks (" "). However, be aware that if you do this and you later decide to include the module file in a different module hierarchy in a new location, you are responsible for manually changing the names of all relevant module files.

Specifying the Top-Level Module

The top-level-module attribute specifies the module KB that is at the top of the module hierarchy. G2 loads the system tables associated with the top level module. Usually, the top level module requires other modules, specified in the next attribute.

Specifying the Required Modules

The directly-required-modules attribute specifies the modules that the top level module requires directly. A module requires another because of definitions that are contained in the required module.

Class-Specific Attributes of Module Information

The class-specific attributes of the Module Information system table are:

Attribute	Description
module-file-name	Specifies the name of the module, either as the pathname and file name where the module and its system table information are stored, or default.
Allowable values:	*"filename"* default
Default value:	default
top-level-module	Specifies the top-level module for the KB. G2 uses the name you specify when identifying modules to load into the hierarchy.
Allowable values:	*symbol* unspecified
Default value:	unspecified
directly-required-modules	Lists the modules, if any, required by the module named as the top-level-module in this table. The required modules are loaded in order as they are listed for this attribute.
Allowable values:	any valid module name
Default value:	none

Attribute	Description
module-annotations	You can add any information you wish to this savable attribute as long as it conforms to the allowable attribute syntax.
Allowable values:	[, *symbol* is *value*] | none For example: track-inventory-procedure is undefined; list-of-stable-workspaces is sequence (the symbol class-definitions-ws, the symbol method-ws)
Default value:	none

Printer Setup

You can print one or more workspaces directly from within G2. The Printer Setup system table controls how G2 produces printed images of workspaces for output on PostScript or PostScript-compatible printers, or to JPEG picture files[1].

Specifying the Printing Details

The printing-details attribute controls these aspects of printing:

- workspace scaling
- color conversion

Workspace Scaling

The workspace scaling setting controls the scale at which G2 creates workspaces for printing, as in this example:

workspace scaling: 100 workspace units per inch

If you choose scale-to-fit-single-page and the workspace is very large, the workspace items may be illegible when you print them, because of their very small size. If a workspace cannot fit on to a single printed page, G2 automatically prints different parts of the workspace on separate pages, in order of left to right and top to bottom. By attaching the printed pages together, you can assemble a paper display of your entire KB.

1.JPEG support was added since April 2013 release.

Color Conversion

The color conversion setting controls how G2 converts colors for printing. Possible values are:

Color Conversion Setting	Result
black-and-white	Prints the workspace in black and white.
grays	Prints the workspace in shades of gray.
full-color	Prints the workspace in color when you use a color PostScript printer.

An example is:

```
color conversion: black-and-white
```

Tip Before printing a workspace with a full-color background image on a non-color printer, change the color conversion setting to black-and-white. Setting color conversion in this way reduces the size of the image data in the resulting print file.

Specifying the Printer Page Layout

The page-layout attribute lets you specify six different print settings:

Page-Layout Setting	Description
Paper size	Any valid page size, as the next section describes.
Paper orientation	portrait landscape
Left margin	Any number of inches or centimeters. You can specify the unit. For example, you can specify 0.75 inch or 3 centimeter. The default value is 0.5 inch.
Top margin	Same as those specified for Left margin.
Right margin	Same as those specified for Left margin.
Bottom margin	Same as those specified for Left margin.

Paper-Size Setting

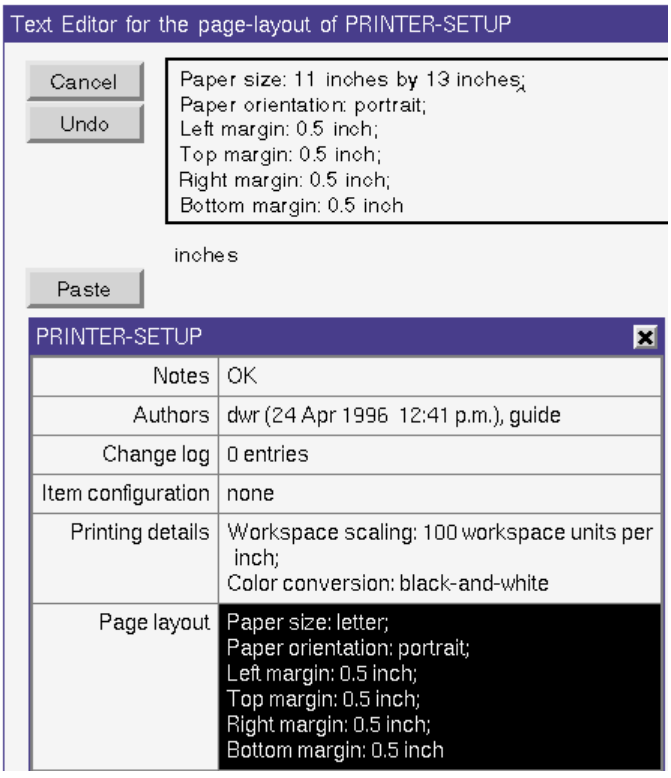
You can specify the dimensions of the physical page that receives the output from the next print job. You can enter either the absolute page dimensions in inches, centimeters, feet, or millimeters, or specify a standard paper size (such as letter, legal, ledger, A3, and so on).

To be valid, a paper-size setting must be at least 1.0" by 1.0" of printable area *plus the* dimensions of the margins and an allowance for portrait or landscape orientation.

The names and dimensions of the standard paper sizes that you can specify are:

- letter (8.5" by 11.0")
- legal (8.5" by 14.0")
- ledger (11.0" by 17.0")
- a0, a1, a2, a3, a4, a5
- b0, b1, b2, b3, b4, b5

This example shows how to specify the page-layout attribute, including an absolute page size:



Specifying How to Spool the Print File

The print-spooling attribute lets you specify three spooling settings:

Spooling Setting	Description
Spooled filename template	The directory to which directory G2 writes the print job file.
Spool file to printer	Determines whether to spool print file. This feature is currently not supported.
Printer identification	Destination printer.

G2 only prints your print job file if your system is configured to spool files to a printer. Otherwise, G2 creates the print job file, but you must spool that file to the printer.

Spooled-Filename-Template Setting

The default for this setting is the directory from which you start G2. You can override the default by specifying a different directory pathname. By default, G2 names the file:

```
print-*.ps
```

where the asterisk represents a number that G2 increments each time it creates a new print file to form a unique name. The first file you print is named print-1.ps, the second print-2.ps, and so on. If you edit the spooled-filename-template setting to write the file to another directory or to use a different file name, remember to keep the asterisk (*) in the name; otherwise G2 does not uniquely name each file.

Spool-File-to-Printer Setting

If the value of this specification is yes, G2 automatically sends the resulting print job to your printer, discarding the print job after printing is complete. If the value is no, you must queue the print job manually. G2 does not know how to spool to the printer on all platforms. This feature is not currently supported.

Printer-Identification Setting

Specifies the name of the printer on which you want to print. This specification is useful when multiple printers are connected to your computer or network. G2 displays the string "unknown" if your computer is not connected to a printer.

An example on UNIX is:

```
Spooled filename template: "/usr/g2/print.ps";
Spool file to printer: no;
Printer identification: "unknown"
```

Controlling the Printing Priority

The printing-priority attribute lets you control the KB background printing priority. The default priority is 8. For more information about scheduling and priorities, see The G2 Scheduler.

Note The system procedure, g2-work-on-printing, lets you further control background printing. For more information, see the *G2 System Procedures Reference Manual*.

Determining the Print File Format

Three print file formats are available: postscript, encapsulated postscript and jpeg. The default format is postscript.

For more information about these two formats, see Printing a Workspace.

Printing a Workspace without Borders

The page-economy-mode attribute allows you to print workspaces without borders. When this attribute is set to yes, G2 does not print workspace borders unless there is a frame style defined for the workspace. Also, G2 does not print blank pages and suppresses the multipage indicator. Use this option to save paper when printing workspaces.

Class-Specific Attributes of Printer Setup

The class-specific attributes of the Printer Setup system table are:

Attribute	Description
printing-details	Controls the scaling of workspace size to paper and color conversion.
Allowable values:	See description following table.
Default value:	Workspace scaling: 100 workspace units per inch; Color conversion: black-and-white

Attribute	Description
page-layout	Controls the page layout for the printer.
Allowable values:	See description following table.
Default value:	Paper size: letter; Paper orientation: portrait; Left margin: 0.5 inch; Top margin: 0.5 inch; Right margin: 0.5 inch; Bottom margin: 0.5 inch
print-spooling	Controls the default file specification, spooling capabilities, and printer information.
Allowable values:	See description following this table.
Default value:	Spooled filename template: ‘print-*.ps’; Spool file to printer: no; Printer identification: ‘unknown’
print-priority	The default priority at which G2 services print requests.
Allowable values:	1 – 10
Default value:	8
printing-file-format	Determines whether to print a PostScript or Encapsulated PostScript format.
Allowable values:	postscript encapsulated postscript jpeg
Default value:	postscript

Attribute	Description
page-economy-mode	Determines whether to print a workspace with or without borders.
Allowable values:	yes, no
Default value:	no

Saving Parameters

G2 uses the attributes on the Saving Parameters system table to display current file information for the KB module, and to determine the change logging behavior that module.

Defining the Priority for KB Saving

The default-priority-for-runtime-saving attribute defines the priority at which G2 schedules the task of saving a KB while it is running.

Note This is not the default priority for the g2-save-kb or g2-snapshot system procedures, nor is it the priority at which a KB save operation occurs while the KB is reset or paused.

You can set the default priority at any value from 1 – 10. For a description of scheduling tasks and priorities in G2, see The G2 Scheduler.

Identifying the Current KB

The identifier-of-basis-kb attribute displays this information about a module file:

- The base file name.
- The machine ID of the platform it was saved from.
- The date and time when the module file was saved.

Identifying the KB File Name

The filename-of-basis-kb attribute displays the full pathname of a module file.

Adding Comments to a KB

You can add comments to your KB in the kb-file-comments attribute. This attribute accepts text, but does not require quotation marks (").

To add comments to a KB:

1 Choose Main Menu > System Tables > Saving Parameters.

2 Edit the KB-file-comments attribute.

You can add whatever comments you wish to this attribute. Your comments are saved at the beginning of the KB file as lines of readable text preceded by a semicolon. The next example shows two comments in the Saving Parameters kb-file-comments attribute. The author has preceded each comment with the date:

KB file comment	6/16/2000: This is a test KB to demonstrate kb-change-logging and comments. Comments for a KB let you document global issues. 6/17/2000: Added a new initially rule to LOGS module for changing color at startup.

Viewing KB Version Information

The kb-version-information-for-change-logging attribute of each module provides version information for that module (or KB) when change logging is enabled.

You cannot edit this attribute, though you can query it for informational purposes.

Using KB Change Logging

You can keep a record of certain changes made to a KB during processing. This facility is called KB change logging.

The enable-KB-change-logging? attribute is a truth value, whose default is no. Changing the value to yes enables change logging.

You enable KB change logging for any module (or a KB if it is not yet modularized) to track each change made to the system tables and **definitional items**. Definitional items include rules and all of the items you can create from the KB Workspace New Definition menu.

To enable change logging for the top-level module:

1 Choose Main Menu > System Tables > Saving Parameters.

2 Change the value of the enable-KB-change-logging attribute to yes.

When KB change logging is enabled in a given module, only edits made interactively through the text editor to the attributes of definitional items and system tables in that module are recorded in those items' change-log attribute. You can review and revert changes at any time.

Logging Changes in All Modules

To enable change logging for all modules, the following procedure iterates over each Saving Parameters system table and changes the value of its enable-KB-change-logging? attribute to yes (true).

```
start-change-logging()
SP: class saving-parameters;
begin
    for SP = each saving-parameters do
        conclude that the enable-kb-change-logging of SP is true
    end
end
```

In addition to keeping previous attribute values, the change log also saves the author, the date, and the version of the KB or module at the time of the edit.

Tracking KB Versions

When KB change logging is enabled, G2 keeps track of relevant changes by assigning a KB version number. Whenever a KB or module is saved, G2 increments the current version number. Changes to the attributes of definitional items in the module or KB then correspond to their appropriate version.

KB version information is also stored in the Saving Parameters system table in the kb-version-information-for-change-logging attribute. The next example shows the portion of the Savings Parameter system table where KB version information appears:

Enable KB change logging	yes
KB version information for change logging	Version 2 (31 May 2000 8:40 a.m.); Version 1 (31 May 2000 8:39 a.m.)

Viewing the Change Log for an Item

The number of changes made to each system table and definitional item in a module appear in the item's change-log attribute as a number of entries. For example, if you edit a rule twice, the value of the rule's change-log attribute will be 2 entries. You cannot edit the value of the change-log attribute; it is for purely informational purposes.

If the value of the change-log attribute is greater than one, you can view the item change log.

When editing an item produces no changes to the item, G2 does not add an entry to the change log.

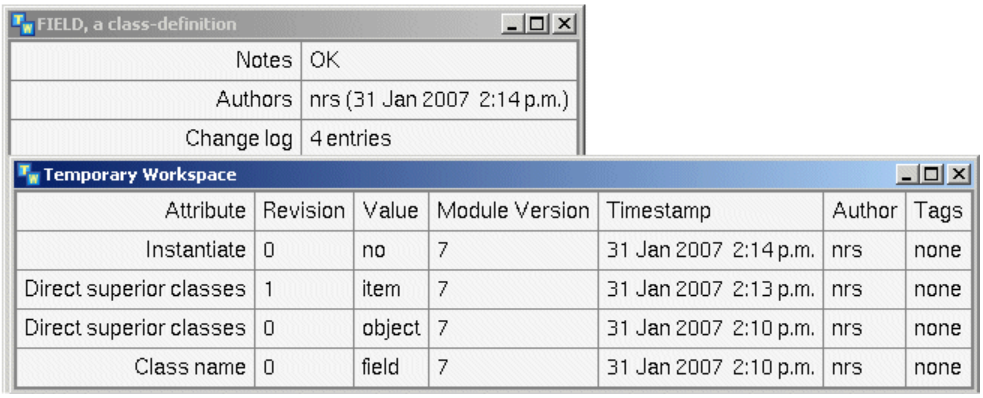
To see an item's change log:

1 Open the item attribute table.

2 Click on the name of the change-log attribute to display its submenu.

3 Choose view change log.

The following example shows the change log display of a class-definition.



Each change log entry consists of:

- Attribute — The name of the changed attribute.
- Revision — The revision number for the change.
- Value — The value of the attribute for that revision.
- Module Version — When change-logging is enabled on a particular module, each time the module is saved, it is given a unique version number. The module version and corresponding date and time of the save are visible in the Saving Parameters system table.
- Timestamp — The date and time of the edit.
- Author — The user name of the author.
- Tags — User-defined tags.

You can also access change-log information programmatically and use text "diff" tools on change log entries. For more information, see Application Deployment Operations in the *G2 System Procedures Reference Manual*.

For information on using the Inspect facility for version control, see Version Control.

Reverting Item Changes

Using the change log, you can restore former attribute values at any time.

To revert a change to an item:

1 Open the change log for the item whose value you wish to revert.

2 Edit the attribute value you want to change.

3 Delete the text of the attribute value.

4 Select the text of the change log attribute value that you wish to restore. The text is inserted into the editor.

5 Click End.

For information about removing change logging and version information before deploying and distributing your KB, see Removing KB Change Logging and Version Information.

Class-Specific Attributes of Saving Parameters

The class-specific attributes of the Saving Parameters system table are given in the table below. Only attributes that have an *Allowable values* description are user-editable. The *Default value* specification for each attribute is the value the attribute has when G2 is initialized for the start of a new G2 process or after the current KB is cleared.

Attribute	Description
default-priority-for-runtime-saving	Allows you to specify the priority for the task of saving your running KB.
Allowable values:	1 – 10
Default value:	8
identifier-of-basis-kb	Provides three items of information for a module file: the base file name, the machine ID of the platform the module was saved on, and the time of the save.
Default value:	none when starting a new G2 process new-kb after clearing the current KB
filename-of-basis-kb	Displays the file path for the module.
Default value:	none
KB-file-comment	Allows you to enter any comment you wish to save with the current KB.
Allowable values:	any text
Default value:	blank

Attribute	Description
enable-KB-change-logging	Use this attribute to specify whether you want change logging to be in effect for the KB.
Allowable values:	no | yes
Default value:	no
KB-version-information-for-change-logging	Displays version information about the KB when change logging is in effect.
Default value:	none
current-file-for-module	Shows the file path of the currently loaded module file.
Default value:	none

Server Parameters

From the Server Parameters system table you can specify preferences that pertain to your G2 process independent of the resident KB. Your preferences persist in the G2 process until you explicitly change them because, unlike other system tables, the Server Parameters table does not lose its non-default attribute values when the KB is cleared. The table is created by G2's initialization process when you first launch G2 and remains in residence throughout the G2 process. It is not saved with the KB.

To access the Server Parameters system table:

➔ Choose Main Menu > System Tables > Server Parameters.

Specifying a Module Search Path

By editing the module-search-path attribute, you specify what file directories G2 searches for locating your required KB modules. This attribute accepts quoted file paths separated by commas, or it accepts the value none.

For example:

"/home/user/support-modules/", "/development/required-kbs"

For complete module-search-path syntax, see Module Search Path Syntax.

Controlling Edits to Read-Only Module Files

The restrict-edits-to-read-only-files attribute enables or disables G2's editing-prohibition and warning behavior when editing is attempted on a read-only module file. The module-file-is-read-only attribute of the module's Saving Parameters system table tells you whether a module file is read-only.

You specify G2's behavior by editing the unsavable-change-protection and default-unsavable-change-protection attributes of the Savings Parameter system tables.

When restrict-edits-to-read-only-files is set to yes, G2 enforces your preferences. When this attribute is no, G2 will neither prohibit nor warn when there is an attempt to edit a read-only module.

Specifying the Default Window-Style

You specify the default window-style for the G2 process by editing the g2-window-style attribute. The syntax for this attribute is:

default | standard-large | g2-5.x | standard

Specifying default is the same as specifying standard because standard is G2's default window style. You override this default by editing the g2-window-style attribute of the g2-window item associated with your local G2 process or your Telewindows process, as described in Overriding the Default Window Style for the Current Window. Alternatively, you can edit the G2 window style field of your login dialog.

Determining if G2 is Secure

The hidden attribute named g2-is-secure allows you can test for a truth-value to determine whether or not a G2 is secure. This attribute is read-only, which means that you can access it but not set it via a conclude action.

To view the attribute interactively, choose Inspect, then enter show on a workspace server-parameters, choose table of hidden attributes, and view the value of g2-is-secure.

Programmatically or in a readout-table, use this expression to determine the value of the hidden attribute:

the g2-is-secure of server-parameters

Class-Specific Attributes of Server Parameters

The class-specific attributes of the Server Parameters system table are all user-editable. They are:

Attribute	Description
module-search-path	Determines the directories G2 searches to locate required-module files.
Allowable values:	The value none or one or more quoted file paths separated by commas. For example: "/home/user/test-kbs", "/development/current"
Default value:	The file path(s) given by your `-module-search-path` command-line option or environment variable, or the file path from which G2 was launched.
restrict-edits-to-read-only-files	Enables and disables G2's editing-prohibition and warning behavior when editing is attempted on a read-only module.
Allowable values:	yes | no
Default value:	no
g2-window-style	Determines the default window-style for a G2 process.
Allowable values:	default | standard-large | g2-5x | standard
Default value:	default
g2-is-secure	Determines whether G2 is secure. This attribute is a hidden and read-only.
Allowable values:	true | false
Default value:	false

Simulation Parameters

The Simulation Parameters system table controls the G2 Simulator, a superseded capability. By default, the G2 Simulator is off. For information about the G2 Simulator, see Appendix F, Superseded Practices.

Timing Parameters

The Timing Parameters system table controls several scheduler settings and other computational parameters.

Defining the Scheduler Mode

The scheduling-mode attribute defines the timing mode of the scheduler (how the G2 clock ticks), and how tasks are scheduled. G2 has three scheduler modes:

- real time
- simulated time
- as fast as possible

Note A clock tick is a measure of time within G2 that may or may not be equivalent to one second of real time. The relationship between a clock tick and real time is determined by the value of the scheduler-mode attribute.

Real Time

If the scheduler mode is real time, a clock tick corresponds to one second of real time. If G2 completes all of the tasks that are scheduled for a particular second before the second ends, G2 waits until the second is over before starting to process tasks scheduled for the next second.

If G2 has tasks left over at the end of a second, it begins processing tasks scheduled for the next second, anyway. When this happens, G2 schedules the remaining old tasks with the tasks scheduled for the new second, preserving the priority of all tasks, and preserving temporal ordering within priorities. G2 performs tasks from the previous second before it performs tasks with an equal priority from the next second. It does not, however, perform lower priority tasks from the previous second before higher priority tasks from the next second. The default mode is real time.

Simulated Time

Simulated time always attempts to match real time. However, when running in simulated time, G2 completes all of the tasks scheduled for one second before moving on to the tasks scheduled for the next second. As a result, a second of

simulated time may last longer than a second of real time. Consequently, the simulator clock may run slower than the real-time clock.

If you pause a KB, reach a breakpoint, or suspend G2, the simulated time stops. When you resume, the simulated time does not leap ahead to match real time. Thus, simulated time lags behind real time as a result of such interruptions.

As Fast As Possible

When the scheduler-mode is set to as-fast-as-possible, and all tasks in the current task queue are complete, the scheduler checks to see if tasks are scheduled on the future task queue. If tasks are scheduled, the scheduler ticks the clock forward all the way to the time of the next scheduled task and starts its execution cycle. If no tasks are in the future task queue, the scheduler becomes idle and does not tick the clock until tasks appear on the future task queue. For more information about the scheduler and the current and future task queues, see Task Scheduling.

As fast as possible time is a convenient scheduler mode for discrete event simulations, so that tests that might otherwise require hours to complete require only minutes.

Caution The G2 clock has a limit of 17 calendar years. Reaching that limit, for example when running simulations using the as fast as possible mode, will abort G2.

Specifying the Minimum Scheduling Interval

The minimum-scheduling-interval attribute specifies the length of time for a clock tick, which determines how long the scheduler has to perform tasks between clock ticks. The default value is the time interval 1 second. Possible values for the attribute are any non-negative-number time interval, or continuous.

If you are entering a time interval (rather than continuous as the interval), the interval value must be a multiple of a second or must divide evenly into a second. If you enter another kind of value, G2 rounds the value up to the next valid minimum scheduling interval. For example, if you enter .333 seconds as the minimum scheduling interval, G2 rounds that number up to .334 upon completing the edit. Entering .666 seconds causes G2 to behave as if the value were 1.0, rounding it to 1 second.

The minimum value is .002 (2 milliseconds). Setting a lower value for the minimum scheduling interval means you can take advantage of faster machine speeds. It also means the gap between the minimum value for this attribute and a value of continuous is much less.

When the minimum-scheduling-interval attribute is set to a time interval, the scheduler advances the G2 clock by multiples of that amount (for instance, .333 seconds, or 1 second). When a time interval is in effect, G2 rounds the execution times of scheduled tasks up to the next clock tick. For example, if the current

subsecond time is 5.0, and the minimum-scheduling-interval is set to 0.25 seconds, G2 schedules the action start update report after 0.6 seconds to run at 5.75 seconds, current subsecond time.

If the interval is continuous, the scheduler ticks the clock at the task schedule times, and remains idle between those tasks. Continuous scheduling incurs clock ticks of various lengths, as tasks are scheduled. For example, if the scheduler ticks the clock, completes the tasks on the current task queue, and then sees that there is something scheduled on the future task queue in .2 seconds, the clock ticks at that time. Conversely, if nothing is scheduled on the future task queue for 5 minutes after the current task completes, the scheduler does not tick the clock until then.

Specifying the G2-Meter Lag Time

The meter-lag-time attribute tells G2 how many seconds of data to use when computing values for G2 meters.

Frequently, performance values are expressed in events per second or are measured in seconds. For example, a G2 meter can track the number of formulas G2 evaluates each second, or how long a clock tick actually lasts in terms of seconds. G2 meters can compute such values for the most recent clock tick, or they can compute values based on an average result of recent clock ticks. The meter-lag-time attribute controls how many seconds worth of data G2 uses in evaluating each meter.

The meter-lag-time attribute holds a value of 0 seconds or any longer time interval. If it holds 0 seconds, G2 meters reflect only the activity in the most recently completed clock tick. As its value increases, the values of G2 meters change more smoothly over time.

G2 computes lagged values as follows:

new lagged value =	(1 - β) * previous lagged value + (β * current value)
where: β	Is equal to min (1.0, clock tick length / meter lag time)

This is an Euler approximation of first-order delay. Note that if the meter lag time is zero or is less than the latest clock tick length, then β = 1.0, and the new lagged value = the current value, with no lag.

Specifying the Interface Mode to Use

The interface-mode attribute specifies which interface mode G2 uses, as follows:

This interface mode...	Causes G2 to...
always service interface first	Make responding to the mouse and keyboard a priority over all scheduled events.
interruptible interface service	Give an equal share of computing time to the keyboard and mouse, the Inference Engine, the G2 Simulator and other data servers. The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices.

Adjusting the G2 Clock

The clock-adjustment-in-minutes attribute adjusts G2's clock as follows:

This clock adjustment...	Causes G2 to...
positive number	Set the clock forward to a value derived by adding the specified positive number of minutes to the current real time.
negative number	Set the clock backward to a value derived by adding the specified negative number of minutes from the current real time.
0	Leave the clock unchanged.

Setting the clock-adjustment-in-minutes to a positive or negative number sets the clock but does not cause any of the events that where scheduled during that adjustment period to occur.

Controlling the Foreign Function Timeout Interval

The foreign-function-timeout-interval attribute controls the interval of time that G2 waits for a return value after calling a foreign function. If the interval is exceeded, G2 signals an error. Specify an integer to represent the interval in seconds.

Note Setting the timeout-interval attribute of an individual foreign function definition overrides the value set for the foreign-function-timeout-interval attribute of the Timing Parameters system table for that function.

Controlling Foreign Image Reconnection

The reconnect-to-foreign-image-after-timeout? attribute controls whether G2 reconnects to a foreign image (a group of foreign functions) after the foreign function timeout interval expires. If the value for this attribute is yes, G2 makes a single attempt to reconnect to the foreign image. If the value is no, G2 does not attempt to reconnect.

Setting the Uninterrupted Procedure Limit

The uninterrupted-procedure-execution-limit attribute sets a limit on the amount of execution time a procedure can use without entering a wait state that allows other processing to occur. Specify an integer to represent the number of seconds, or none. The actual limit for this attribute is 24 hours. Note that setting the execution time limit locally for a procedure overrides the limit set for this attribute.

G2 maintains a tally of the cumulative execution time per invocation of each executing procedure.

Scheduling Attribute Table Updates

By default, attribute tables are updated whenever a change occurs in the value of an attribute or to the class-specific attributes of the defining class. When attribute changes are occurring at a very fast rate, continuous attribute-table updates place a considerable load on G2, its clients, and the network between them.

You can direct G2 to update attribute tables only at specific intervals instead of continuously. G2 defers updates until the specified interval of time has elapsed, then updates the table with the latest changes, thus avoiding the overhead of updating intermediate value changes.

Class-Specific Attributes of Timing Parameters

The class-specific attributes of the Timing Parameters system table are:

Attribute	Description
scheduler-mode	The current mode in which the scheduler is running.
Allowable values:	{real time | simulated time | as fast as possible}
Default value:	real time
minimum-scheduling-interval	The length of the G2 clock tick.
Allowable values:	{*subsecond-interval* | continuous}
Default value:	1 second
milliseconds-to-sleep-when-idle	Controls the interval of time, in milliseconds, that G2 sleeps while the G2 process is idle.
Allowable values:	any positive integer use default
Default value:	use default
meter-lag-time	The number of seconds of data to use when computing G2 meter values.
Allowable values:	*time-interval*
Default value:	10 seconds
interface-mode	Specifies which interface mode G2 uses.
Allowable values:	{interruptible interface service | always service interface first}
Default value:	interruptible interface service

Attribute	Description
clock-adjustment-in-minutes	The number of minutes to adjust G2's clock.
Allowable values:	*integer*
Default value:	0
foreign-function-timeout-interval	The interval of time that G2 waits for a return value after calling a foreign function.
Allowable values:	{none | *time-interval*}
Default value:	30 seconds
reconnect-to-foreign-image-after-timeout?	Whether G2 reconnects to a foreign image (a group of foreign functions) after the foreign function timeout interval expires.
Allowable values:	{yes | no}
Default value:	no
uninterrupted-procedure-execution-limit	A limit on the amount of execution time a procedure can use without entering a wait state that allows other processing to occur.
Allowable values:	{*time-interval* | none}
Default value:	30 seconds
attribute-display-update-interval	Specifies the frequency with which to update attribute tables.
Allowable values:	continuous | float (between 0.0 seconds and 0.5 seconds)
Default value:	continuous

7

Configurations

Describes how configurations override the default behavior of items.

Introduction

Configurations are declarations that determine the interactive behavior and certain other properties of items. Using configurations you can:

- Customize how items respond to mouse clicks and to drag-and-drop mouse operations.
- Assign custom keystrokes to G2 commands and operations.
- Customize which choices appear on the menus of items.
- Customize which attributes appear in the tables of items, as well as which choices appear on the menus of those tables.
- Prohibit and allow access to items, and to the entire KB, by other G2 processes, by G2 Gateway bridge processes, and by Telewindows processes.
- Enable or disable compilation properties, and other miscellaneous properties, of items.
- Add comments to items.

For instance, you can use configurations to restrict how any item of a particular class responds to being selected, or to restrict access to the proprietary knowledge within your KB.

Most importantly, you use configurations to associate specialized behaviors with different categories of users, namely, end users, developers, and administrators.

Note The current KB's configurations are in effect at all times, regardless of whether it is running, paused, or reset.

Declaring Configurations for Items

You declare configurations for items by entering configuration statements in their item-configuration and instance-configuration attributes.

You use an **item configuration** to customize the behavior of an item, based on its location within the current KB's workspace hierarchy. You declare an item configuration by entering configuration statements into the item-configuration attribute of an item. Items of every class have an item-configuration attribute.

You use an **instance configuration** to customize the behavior of a class of items, based on the position of their class in the KB's class hierarchy. You declare an instance configuration by entering configuration statements into the instance-configuration attribute of a class definition.

Only class definitions have an instance-configuration attribute. Therefore, you can use instance configurations only to customize items of user-defined classes.

The next figure shows a configuration statement. It declares that clicking the mouse on any class definition causes G2 to create an instance of the class.

```
configure the user-interface as follows:
when in developer mode:
    pressing any mouse button on any class-definition
        implies create-instance
```

You can also declare global configurations by entering them in the KB Configuration system table. Some configurations exist there by default, which you can change as you require.

Kinds of Configuration Statements

An item-configuration or instance-configuration attribute can contain one or more configuration statements. There are five general types of configuration statements, summarized in the following table. For information on cooperative combinations, see Combining Cooperatively.

Configuration Statement	Support Cooperative Combinations?	Purpose
configure the user interface as follows	Yes	Determines how an item responds to interactive operations.
restrict proprietary items as follows	Yes	Determines how a proprietary item responds to interactive operations and certain programmatic operations.
set up network access as follows	No	Determines access to an item (or to the entire KB) by other G2 process and by G2 Gateway bridge and Telewindows processes; effects read, write, execute, inform for items and the entire KB, and connect access to the G2 process.

Configuration Statement	Support Cooperative Combinations?	Purpose
declare properties ... as follows	No	Declares an item as: disabled, text-stripped, stable-for-dependent-compilations, independent-for-all compilations, stable-hierarchy, inlineable, or not. In instance configurations, configures items to support: activatable-subworkspace, external-simulation, manual-connections, or subworkspace-connection-posts. The editor prompts include optimizable configuration syntax, but it no longer has any effect in G2. The grammar is maintained in order to prevent older KBs from incurring compilation errors. The G2 Simulator, which can provide external simulation, is a superseded capability. For more information, see Appendix F, Superseded Practices.
comment as follows	No	Declares comment text in a configuration.

A configuration statement can have more than one clause. Within one configuration statement, use a semicolon (;) to separate the statement's clauses. Do not append a semicolon to the last configuration statement.

Configurations propagate through your KB's class and workspace hierarchies. Thus, there is no need to specify configurations in each item of your KB.

Note The item-configuration and instance-configuration attributes are compiled attributes. G2 saves a compiled version of the attribute's text in the item, not the exact text that you enter.

Scope of Configurations

The **scope** of a configuration means the items to which it applies.

Items can inherit the configurations that are declared for items higher in the class and workspace hierarchies. Thus, one configuration can apply to many items in your KB. For example, the following configuration statement optimizes the compilation of all tracked-vehicle items in a KB:

```
declare properties of any tracked-vehicle as follows :
    stable-for-dependent-compilations
```

Different configurations can overlap, such as when you declare configurations on two items on the same branch of the KB's workspace hierarchy. Thus, more than one configuration can also apply to the same item.

You can also declare a configuration that applies to only one item. Such a configuration is not inherited down the class or workspace hierarchies. In the configuration statement, instead of naming the class of items that the configuration applies to, include the this item phrase. For example:

```
declare properties of this item as follows :
    stable-for-dependent-compilations
```

Precedence of Configurations

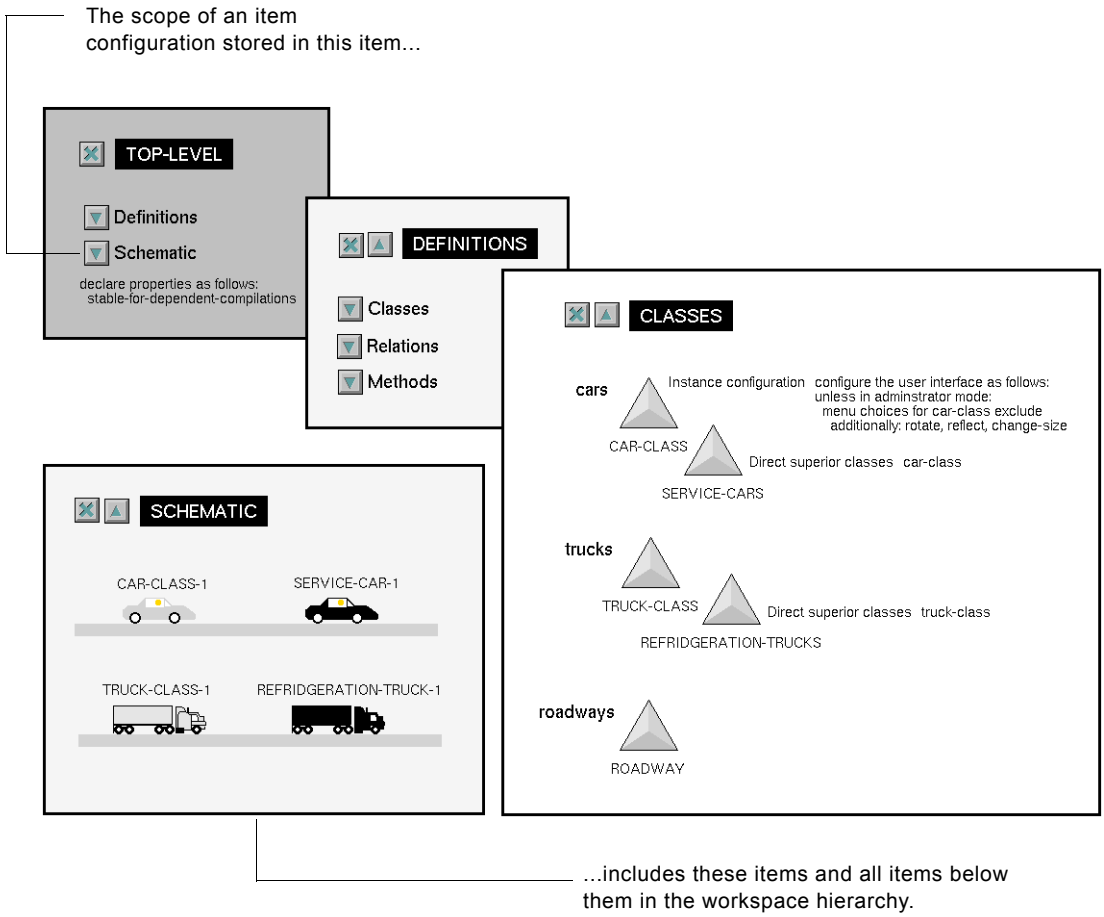
If two configurations apply to the same item, it is possible that they declare conflicting behaviors for that item. For example, one configuration declared for any item of the vehicle class might declare that each vehicle item must respond to a mouse click by rotating, and another configuration declared for the vehicle item named security-vehicle might declare that it not respond to mouse-clicks at all.

In this situation, G2 uses a predictable mechanism to determine which of two or more conflicting configurations to use. G2's **precedence** rules for configurations are based on the current KB's class and workspace hierarchies. By default, if configurations for an item conflict, G2 uses the configuration declared closest to the target item in the class and workspace hierarchy, and ignores the conflicting configurations.

Example of the Scope of Configurations

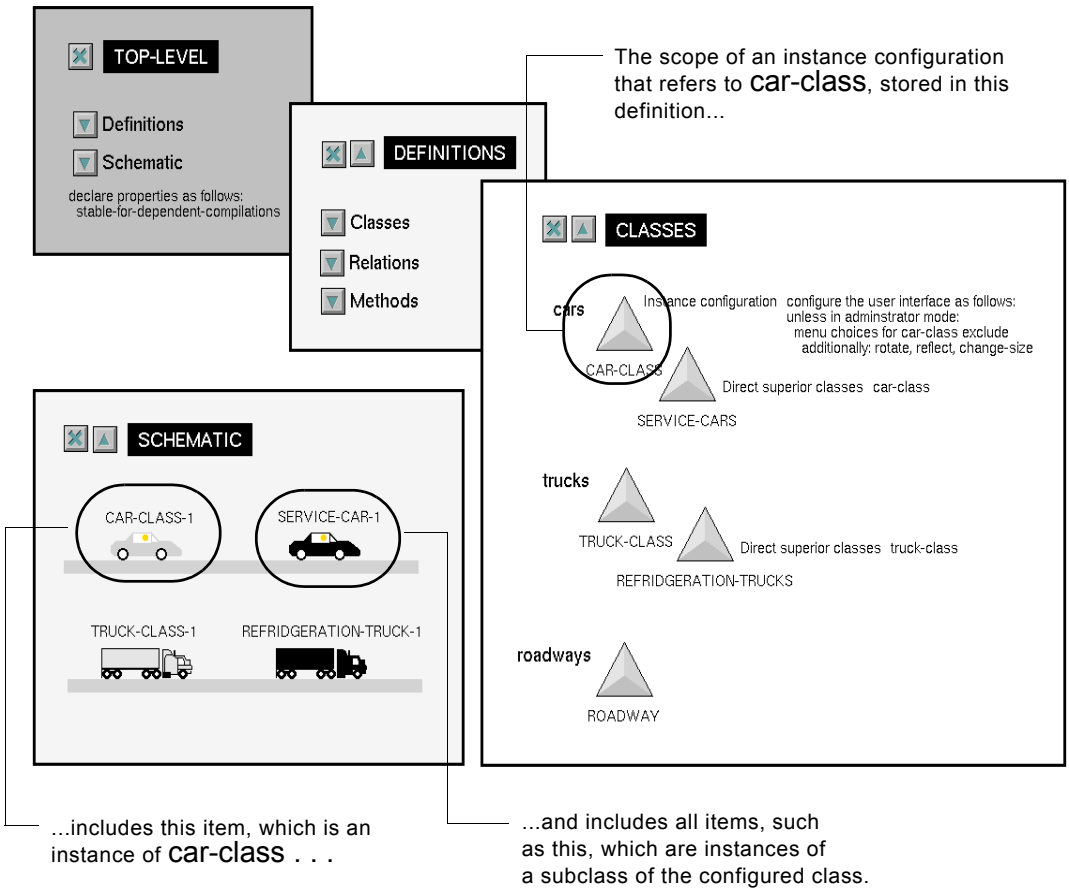
The following example illustrates how the KB's class and workspace hierarchies determine the scope of an item configuration. Consider a KB that monitors vehicles. Assume that one of the KB's top-level workspaces is named Top Level, and that it contains two navigation button items, Definitions and Schematic, each of which has a subworkspace of the same name.

The Schematic navigation button declares an item configuration, which optimizes the compilation of the navigation button itself, and all items below it in the workspace hierarchy. Thus, by default, all items on the Schematic workspace are automatically optimized for compilation.



The next figure shows that an instance configuration stored in a class definition applies to all items of that class *and* to all items of *any subclass* of that class. In this way, one instance configuration can affect a large set of items.

The figure shows that an instance configuration declared in the car-class definition applies to all instances of car-class *and* to all items of the service-cars class, which is a subclass of car-class. The instance configuration restricts the menu choices to exclude rotate, reflect, and change size. Thus, the two cars on the Schematic workspace both inherit the configuration in the car-class definition.



How G2 Searches for Applicable Configurations

When an item is the target of a user gesture, before G2 performs the operation associated with that user gesture, G2 must determine which configurations apply to the item at that particular moment in time. To do so, G2 searches in the following order for configuration statements in the following items:

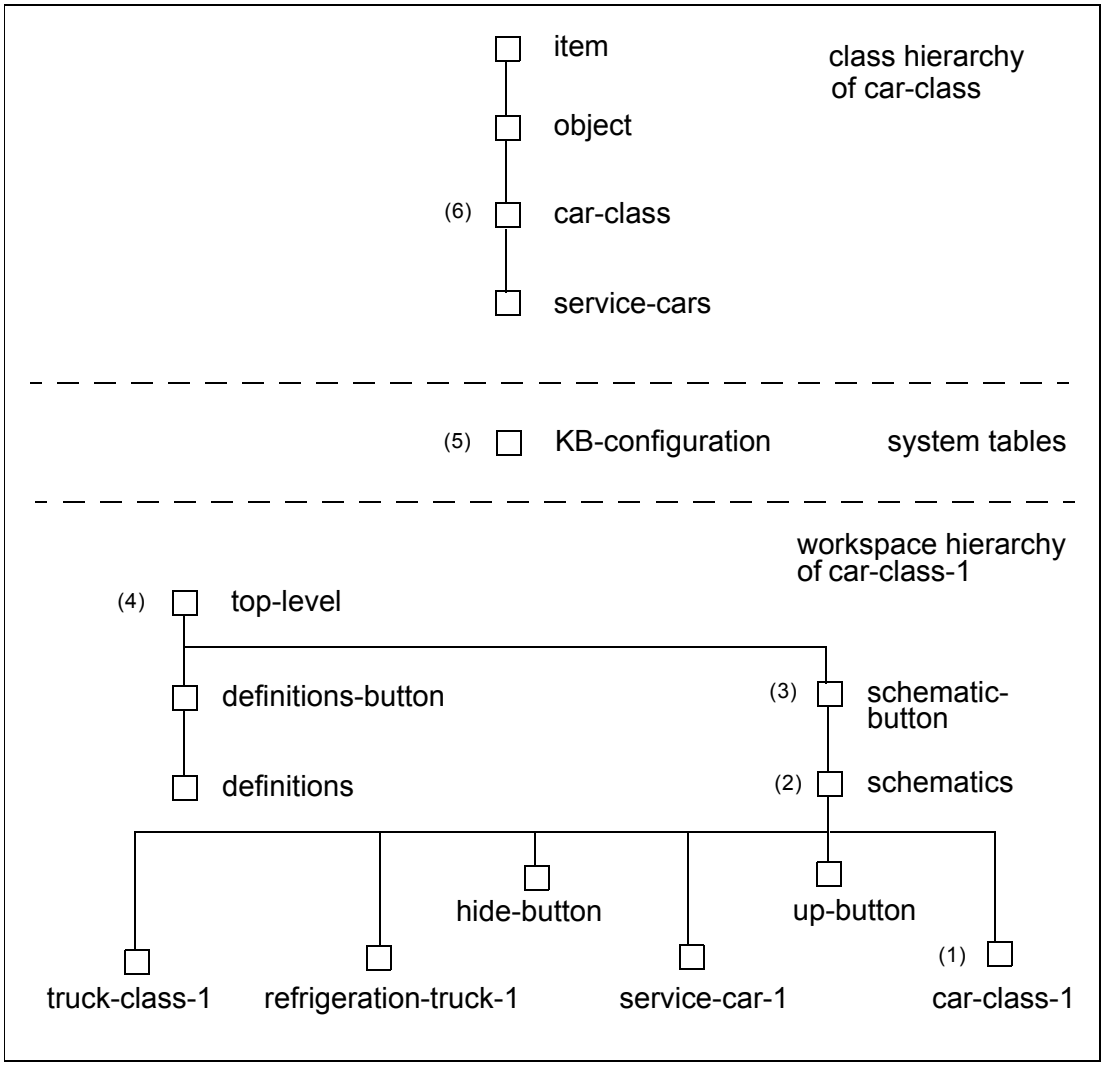
1. Configuration statements in the item-configuration attribute of the item.
2. Configuration statements in the item-configuration attribute of each item that is above the current item in the workspace hierarchy.

   If the item that declares the configuration is contained in an attribute of some other item, G2 searches all workspaces above the item containing the configured item.
3. Configuration statements in the item-configuration attribute of the KB Configuration system table.
4. If the class of the item is user-defined, configuration statements in the instance-configuration attribute of the class definition that defines the item's class.
5. If the class of the item is user-defined, for each definition item that declares a class in the class-inheritance-path attribute of the class definition, configuration statements in the instance-configuration attribute of that definition item.

Given the KB shown in the previous figures, the following figure shows how G2 searches for the configurations that apply when a user clicks the mouse on car-class-1. G2 reacts to this user gesture as follows:

1. Checks whether the item-configuration attribute of the car-class contains configuration statements.
2. Checks whether any configuration statements apply to car-class-1 in the item-configuration attribute of the schematic workspace.
3. Checks the Schematic navigation button for an item-configuration.
4. Checks the Top Level workspace for an item-configuration, which is the superior item of the workspace hierarchy for car-class-1.
5. Checks whether any configurations declared in the item-configuration attribute of the KB Configuration system table apply to car-class-1.
6. Finally, since the car-class class is user-defined, G2 checks whether the instance-configuration attribute of the car-class definition contains configuration statements that apply to car-class-1.

Items Searched for Configurations that Apply to Car-Class



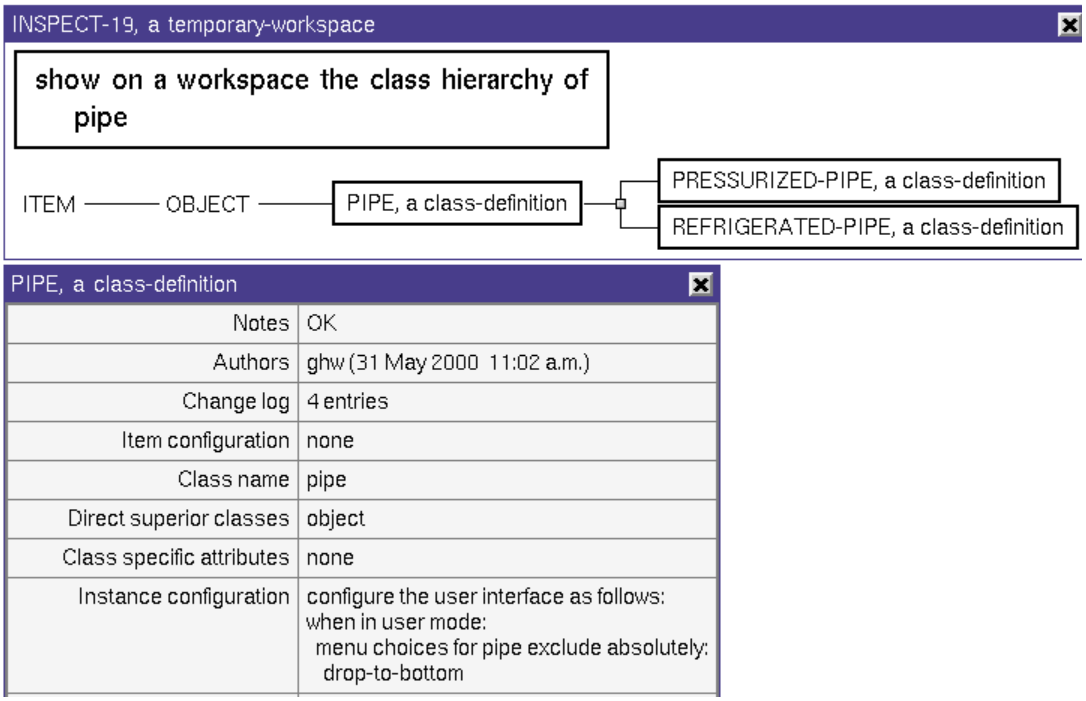
Because the direct superior class of car-class is object class, which is a foundation class, G2 stops searching for additional instance configurations. (A foundation class is a G2 system-defined class that can be the direct superior class of a user-defined class.) Otherwise, G2 would search for any instance configuration declared in the definitions of all user-defined superior classes of car-class.

As stated in Precedence of Configurations, if two configuration statements, or clauses within a configuration statement conflict for an item, the *last* statement or clause takes precedence.

Instance Configurations and Definition Items

An instance configuration does *not* apply to the class-definition item itself, *nor* to the class-definition items that define subclasses of the configured class.

For example, in the figure below, an instance configuration declared in the pipe class-definition includes in its scope all instances of the pipe class *and* all instances of the pressurized-pipe and refrigerated-pipe classes. However, the scope of that instance configuration does *not* include the class-definitions themselves.



Configuring the User Interface of Items

You configure how one or more items respond to interactive operations with a statement that begins with this phrase:

configure the user interface as follows :

Each configure the user interface as follows statement incorporates clauses that fall into four distinct categories:

- Configuring menu choices and attributes in tables.
- Configuring mouse clicks.
- Constraining movement.
- Configuring keystrokes.

Specifying the Applicable User Modes

Each configure the user interface as follows statement must include at least one **user modes clause**. This clause names one or more user modes under which the configuration does or does not apply.

Use the when in phrase to list the user modes that apply to a set of configuration clauses. Use the unless in phrase in a user modes clause to list the user modes that *do not* apply to a set of configuration clauses. For example:

```
configure the user interface as follows :
    when in developer or end-user mode : { inclusive }
        ... ;
    unless in administrator, developer, or end-user mode : { exclusive }
        ...
```

As shown above, when specifying only two user modes, separate the mode identifiers with the or reserved word. When specifying three or more modes, separate the identifiers with commas, and include the reserved word or before the last identifier.

Note You cannot specify a configure the user interface as follows statement that applies only under administrator mode. You can specify configurations that apply when *not* in administrator mode. This prevents you from unintentionally restricting access to a portion of your own KB.

Declaring User Modes in Configurations provides more information about user modes. Describing Configurations provides more information about configuring the user interface in the various user modes, including a complete example.

Specifying Appropriate Operations for the Target Class

If a clause in a configure the user interface as follows statement refers to a menu choice, attribute, or low-level G2 operation that is not appropriate for the class of the target item, G2 ignores that reference in the configuration clause.

For example, suppose you create this configuration statement:

```
configure the user interface as follows :
    unless in developer mode :
        menu choices for rule or custom-object exclude : hide
```

During the KB's processing, G2 ignores the reference to the rule class in the menu choices for rule exclude clause, because hide is not one of the system-defined menu choices for the rule class. However, G2 does *not* ignore the entire clause, because hide could be a user-menu-choice for the user-defined custom-object class.

Configuring Menu Choices and Attributes in Tables

The following clauses affects which menu choices can appear, which attributes and table menu choices can appear in an attribute table, and which non-menu choices can appear for an item:

Clause	Purpose
attributes visible for	Configures which of an item's attributes are displayed in the item's table.
menu choices for	Configures which system-defined menu choices and user-menu-choices are displayed in the item's menu.
non-menu choices for	Configures which system-defined interactive operations for an item, for example, move, scale, show, click-to-edit, and so on.
table menu choices for	Configures which menu choices are available when you select attributes in an item's table.

These four clauses have a similar syntax:

- Each applies to one or more classes of items.
- Each supports cooperative combinations with similar clauses in other configuration statements or in other configurations, as explained in Combining Configurations.
- Each clause can specify that nothing is included or excluded.

For example, the following configuration statement might appear in the instance-configuration attribute of the definition for the petro-valve class. The statement summarizes the kinds of features that you can configure with these four clauses:

```
configure the user interface as follows :
    when in developer or end-user mode :
        attributes visible for any petro-valve include additionally :
            notes, names, current-status ;

        menu choices for any petro-valve include additionally :
            diagnose-error-condition, show-status ;

        non-menu choices for any petro-valve exclude additionally :
            move-object, click-to-edit, full-editor,
            option-buttons-for-edit-in-place,
            menus-for-edit-in-place, do-not-clear-text-for-edit-in-place ;
```

```
{ This clause configures the table menu choices for the item's
  entire table. }
table menu choices for any petro-valve exclude additionally :
     transfer

{ This clause configures the table menu choices for a particular
  attribute in the item's table. }
table menu choices for the notes of any petro-valve exclude
  additionally : show-attribute-display
```

Configuring Attributes That Appear in Tables

In the developer's environment, every G2 class defines a set of attributes for items based on that class. Use the attributes visible for clause to determine the attributes that appear in the attribute tables for a set of items identified by class. For example:

```
configure the user interface as follows :
     when in developer or end-user mode :
          attributes visible for any petro-valve include :
               notes, names, current-status
```

In the Text Editor, when you specify the attributes visible for clause, G2 prompts you to specify which attributes to include or exclude. The Text Editor presents the following prompts:

- any system-defined-attribute-name: Selecting this prompt displays a list of all system-defined attributes for all system-defined classes.
- any attribute-name: Selecting this prompt displays a list of system-defined attributes.

Configuring Menu Choices

Each G2 class has a system-defined set of menu choices. Use the menu choices for clause to list the menu choices to include or exclude from the menus of a set of items identified by class. For example:

```
configure the user interface as follows :
     when in developer or end-user mode :
          menu choices for any petro-valve include :
               diagnose-error-condition, show-status
```

Configuring Non-Menu Choices

Each G2 system-defined class has a system-defined set of non-menu-based operations. **Non-menu choices** are operations that are not performed in response to the user's selections from menus, such as selecting items, dragging the mouse, input from the keyboard, and other user gestures like showing, hiding, resizing, and scaling workspaces.

For example, to restrict the ability to use standard mouse gestures to copy items, you can exclude non-menu options for selecting an object and selecting an area. When items are restricted in this way, the user cannot execute any commands that apply to the current selection.

These are the non-menu choices that you can configure:

This non-menu option...	Provides the ability to...
select-object	Left-click an item to select it.
select-area	Drag in the open area of a workspace to select all items in the rectangular area.
move-object	Move an item by selecting it with the mouse and dragging.
move-objects-beyond-workspace-margin	Move an item further than the current workspace edge to expand the workspace size.
move-connection	Click on a connection and move the connection on the workspace.
move-workspace	Move the workspace in the current window. Excluding this option prevents the user from moving the workspace.
move-workspaces-beyond-window-margin	Move the workspace beyond the current window margin.
show-workspace	Show a workspace. Excluding this choice removes named workspaces from the list of workspaces available by choosing Main Menu > get-workspace.
scale-workspace	Scaling a workspace. Excluding this choice prevents the user from scaling the workspace with keystrokes such as Control + b and Control + s.
click-to-edit	Enter the Text Editor automatically when a user selects, for example, an attribute value. Excluding this option presents a menu from which the user can choose to edit.
full-editor	Invoke the Text Editor when editing an attribute value. Excluding this option causes G2 to invoke a partial editor in place.

This non-menu option...	Provides the ability to...
option-buttons-for-edit-in-place	Remove buttons from a partial in-place editor when the full-editor option is being excluded. This option thus works in conjunction with excluding the full-editor, further restricting editing capabilities.
menus-for-edit-in-place	Remove the edit in place menu.
do-not-clear-text-for-edit-in-place	Remove the text when editing in place.
allow-selection-of-outside-text-from-editor	Select text from a location outside of the Text Editor and to use that text in the current editing session. Excluding this option prevents the user from sliding over a piece of G2 text and have it appear in the editor.
allow-selection-of-text	Permit text to be selected.

Use the non-menu choices for clause to list one or more non-menu choices to allow or prohibit for a set of items identified by class. For example:

```
configure the user interface as follows :
    when in developer or end-user mode :
        non-menu choices for any petro-valve exclude :
            move-object, click-to-edit, full-editor,
             option-buttons-for-edit-in-place,
             menus-for-edit-in-place, do-not-clear-text-for-edit-in-place
```

You cannot use the non-menu choices for clause to *add* a custom non-menu choice for a class of items.

Configuring Table Menu Choices

In the developer's environment, after you display an item's table, you can click the mouse on the table to display a table menu. Use the table menu choices for clause to name one or more system-defined menu choices to include or exclude from the menu of an item's table. For example:

```
configure the user interface as follows :
    when in developer or end-user mode :
        { This clause configures the table menu choices for the item's
          entire table. }
        table menu choices for any petro-valve exclude additionally :
            transfer
```

Because some table menu choices apply only to the attribute shown on the row where the mouse was clicked, you can also use the table menu choices for clause to include or exclude attribute-specific table menu choices. For example:

```
configure the user interface as follows :
    when in developer or end-user mode :
        { This clause configures the table menu choices for a particular
          attribute in the item's table. }
        table menu choices for the notes of any petro-valve exclude
          additionally :
            show-attribute-display
```

Configuring Attribute Displays

You can restrict access to the attribute displays of items. Unless you restrict access to these displays, users can click on them to open the Text Editor and edit the attribute values themselves.

To do so, specify a configuration clause that names table-item, an internal class that defines the characteristics of attribute displays. You can restrict attribute-display access for a single item, for all the items on a workspace, or for all the items in the KB. You do this by editing the item-configuration attribute of an item, a workspace, or the Kb Configuration system table.

For example, to restrict attribute-display edit access to all the items on a workspace, enter this statement in the item-configuration attribute of the workspace:

```
configure the user interface as follows:
    unless in administrator mode:
        selecting any table-item does nothing
```

Adding this phrase to the statement above also prohibits the movement of the attribute displays:

```
non-menu choices for table-item exclude: move-object
```

Note Item configurations are propagated down the KB workspace hierarchy. For example, entering table-item configurations on an item with a subworkspace makes any items on the subworkspace, as well as any existing items further down the workspace hierarchy, subject to the item configuration.

To restrict edit access to attributes from the attribute-table of all instances of a class, enter this statement in the instance-configuration attribute of the class-definition:

```
configure the user interface as follows:
    unless in admnistrator mode:
        table menu choices for any class include: nothing
```

Configuring Keystrokes

The typing ... implies clause associates keystrokes with G2 operations. When the user displays the Help screen (by typing CTRL + /) the Help information includes any keystroke associations currently in effect.

Use a typing ... implies clause to configure:

- Alphabetic and numeric keys, optionally modified by any combination of the ALT, CTRL, or SHIFT keys.
- The function keys F1 through F12, modified or unmodified.
- The cursor-movement keys (left-arrow, right-arrow, up-arrow, down-arrow), modified or unmodified.
- Other named keys (insert, delete, home, end, page-up, page-down), modified or unmodified.

A typing ... implies clause can associate a keystroke with an operation that targets one item, more than one item, or no items.

You can configure a keystroke to invoke a user menu choice or any of G2's system-defined menu choices.

You can bind printable characters, for example, to execute a menu choice by pressing a single character or to prevent displaying the table for an item when pressing the space bar.

Constraints on Configuring Keystrokes

You cannot use configurations to associate certain keystrokes that are intercepted by your platform's window manager.

For example, Microsoft Windows traps several keystrokes. You cannot associate the following keystrokes with a G2 operation when using the typing ... implies clause:

ALT + ESC
ALT + TAB
ALT + -
ALT + SHIFT + TAB
ALT + [SPACE]
CTRL + ESC

You can configure other standard Windows accelerator keystrokes by using the typing ... implies clause.

Note that when configuring printable characters without any modifier keys, the character bindings are not valid when the text editor is active.

Considering the Target of a Configured Action

When you use the typing ... implies clause to associate a keystroke with an action, consider whether the action requires a target item. A keystroke can apply to:

- A particular item, for example, any kb-workspace.
- The current KB as a whole or the current G2 environment, depending on the G2 operation, in which case, the clause contains no target item.

When a configured action applies to an item, a user must move the mouse pointer over the target item before executing the keystroke.

Example of Configuring Keystrokes

The first clause below associates a keyboard keystroke with a G2 operation that applies to a workspace; the second associates a character with the G2 operation that presents the New Object menu; the third associates a keystroke with the G2 operation that resets the current KB; and the fourth prevents attribute table of an item from displaying when the user presses the Space bar.

```
{ target is any kb-workspace }
    typing alt + f on any kb-workspace implies full-scale ;

{ target is the entire KB }
    typing n implies new-object;

{ target is the G2 developer's environment }
    typing f1 implies reset

{ target is the entire KB }
    typing space does nothing
```

Configuring Mouse Gestures

These clauses declare an association between a **mouse gesture** and a low-level G2 operation or user menu choice. A mouse gesture includes selecting an item, pressing and releasing the mouse, clicking or double-clicking the mouse, rolling the mouse wheel, dragging the mouse, or hovering the mouse.

Note The pressing item configuration clause is invoked for either a mouse-down event or a double-click event. To configure two different actions for a single-click and a double-click event, ensure that the double-clicking clause appears *after* the pressing clause in the item configuration statement. That way, the double-clicking clause takes precedence when determining the behavior for the double-click event.

When you display the Help screen (by typing CTRL + /), the Help information describes any mouse-click associations currently in effect.

Clause	Purpose
selecting ... implies	Associates selecting an item of the specified class (or classes) with a system-defined menu choice, a user-menu-choice, or a null operation.
pressing ... implies	Associates pressing a mouse button with a system-defined menu choice, a user-menu-choice, or a system-defined non-menu operation.
releasing ... implies	Associates releasing a mouse button with a system-defined menu choice, a user-menu-choice, or a non-menu operation.
clicking ... implies	Associates clicking (pressing and releasing) a mouse button with a system-defined menu choice, a user-menu-choice, or a system-defined non-menu operation.
double-clicking ... implies	Associates double-clicking (pressing and releasing twice quickly) a mouse button with a system-defined menu choice, a user-menu-choice, or a system-defined non-menu operation.
rolling ... implies	Associates rolling the mouse wheel with a system-defined menu choice, a user-menu-choice, or a system-defined non-menu operation.
pressing ... on ... starts	Invokes a procedure that tracks movement of the mouse, to support state-based operations such as drag-and-drop.
hovering ... implies	Associates hovering the mouse over an item of the specified class with a system-defined menu choice, a user-menu-choice, or a system-defined non-menu operation.

Note You cannot use configurations to associate certain mouse clicks that are intercepted by your platform's window manager. For example, holding down the ALT key and left-clicking is meaningful for the HP-Vue window manager on Hewlett-Packard 9000 Series workstations.

Syntax Summary

The item-configurations syntax for mouse gestures are:

selecting [on any *class*] implies *action*

pressing [*modifiers*+] (any | the (left | middle | right)) mouse button
[on any *class*] implies *action*

releasing [*modifiers*+] (any | the (left | middle | right)) mouse button
[on any *class*] implies *action*

clicking [*modifiers*+] (any | the (left | middle | right)) mouse button
[on any *class*] implies *action*

double-clicking [*modifiers*+] (any | the (left | middle | right)) mouse button
[on any *class*] implies *action*

rolling [*modifiers*+] the mouse wheel (forward | backward)
[over any *class*] implies *action*

pressing on [*modifiers*+] (any | the (left | middle | right)) mouse button
[on any *class*] implies *action*

hovering [*modifiers*+] the mouse [over any *class*]
{implies *action* | does nothing}

where:

- *modifiers* are control, alt, or shift modifier keys.
- *class* is the class name to which the item configuration applies, which is optional.
- *action* is the action to perform when the event occurs.

Statement	Description
selecting...implies	Pressing and releasing the left mouse button on an item.
pressing...implies	Pressing the mouse button down.
releasing...implies	Lifting the mouse button up.

Statement	Description
clicking...implies	Pressing the mouse button down and up without moving the mouse, within some tolerance.
double-clicking...implies	Pressing, releasing, and pressing the same button without moving the mouse much and within the time limit for double clicks as set by the window system (typically 300-500ms).
rolling...implies	Moving the mouse wheel forward or backward, where forward means rolling the wheel towards the front of the mouse.
hovering...implies	When the mouse does not move more than a certain amount for a period of time, which is determined by the operating system. On Windows, the default is 4 pixels and 400 milliseconds, respectively.

Example

This example configures the user interface in all modes except administrator, as follows:

- Holding down the CTRL key and left-clicking any item displays its table.
- Left-clicking any icon clones the icon.
- Double-clicking any item displays its table.
- With the right mouse button, double-clicking any item displays its table.
- With the middle mouse button, double-clicking any item displays its table.
- Rolling the mouse wheel forward scrolls the workspace up.
- Holding down the CTRL key and rolling the mouse wheel forward scrolls the workspace left.

```
configure the user interface as follows:
    unless in administrator mode:
        pressing control+the left mouse button implies table;
        clicking the left mouse button on any icon implies clone;
        double-clicking on any item implies table;
        double-clicking the right mouse button on any item implies table;
        double-clicking control+the middle mouse button on any item implies table;
        rolling the mouse wheel forward implies scroll-up;
        rolling control+the mouse wheel forward implies scroll-left
```

Associating Selection with a Menu Choice or User Menu Choice

Use the selecting ... implies clause to configure selection operations that apply to *one item*.

Selection means the pair of mouse clicks, mouse-down and mouse-up, which occur in order over the same item.

For example, the following configuration statement associates the system-defined create-subworkspace operation with selecting an item of the conveyor-station class:

```
selecting any conveyor-station implies
    create-subworkspace
```

Tip You should differentiate between the selecting ... implies clause, which is appropriate for configuring operations directed at *one* item, such as create-by-cloning, and the typing ... implies clause, which is appropriate for configuring operations not directed at any particular item, such as save-KB.

You can also specify a selecting ... absolutely implies clause to override all other configurations that use a selecting ... implies clause for the specified class in the same hierarchy. G2 resolves configurations in the same hierarchy with selecting ... absolutely implies clauses that conflict according to G2's precedence rules for configurations.

For example, the following configuration statement, which associates the system-defined go-to-subworkspace operation with selecting an item of the navigate-down-button class, overrides any other conflicting configurations in the configured item's hierarchy:

```
selecting any navigate-down-button absolutely implies
    go-to-subworkspace
```

Associating a Mouse Click with the Miscellany Menu

Use the selecting ... implies miscellany clause to control the display of the Main Miscellany or Workspace Miscellany menus.

The Workspace Miscellany Menu is a short-menu version of the Workspace Menu. It omits some menu choices that are on the Workspace Menu such as the new-item, move, hide, disable, and operate-on-area menu choices. When the class reference in the configuration clause names a workspace class, the clause governs the display of the Workspace Miscellany Menu.

Here is an example:

```
selecting any definition-workspace implies miscellany
```

When the class reference in the configuration clause names a non-workspace class, the clause governs the Main Miscellany Menu. For example:

```
selecting any special-object imples miscellany
```

Associating a Mouse Click with an Operation

You can associate the following types of mouse clicks with an operation, using the following configuration statements:

- pressing ... implies associates a mouse-down event with an operation.
- releasing ... implies associates a mouse-up event with an operation.
- clicking ... implies associates a mouse-up and mouse-down event with an operation.
- double-clicking ... implies associates mouse-down, mouse-up, mouse-down events done in quick succession, with an operation.

For each statement, you can specify the following mouse clicks:

- any mouse button
- left mouse button, middle mouse button, or right mouse button
- Any combination of the control, alt, or shift modifier keys with any mouse button.

You can associate the mouse-down, mouse-up, mouse-click, or mouse-double-click events with an operation that is targeted on:

- The current item:

  ```
  pressing the right mouse button on this item does nothing
  ```

  ```
  releasing the right mouse button on this item implies lift-to-top
  ```

- An item of any class:

  ```
  clicking the left mouse button on any kb-workspace implies hide-workspace
  ```

  ```
  pressing the right mouse button on any workspace implies select-area
  ```

- Not an item, such as the G2 window's background tiling pattern:

  ```
  double-clicking any mouse button implies inspect
  ```

In the Text Editor, when you specify any mouse click with an operation, G2 prompts you to enter the name of a user menu choice, or to select from a list of system-defined menu choices, system-defined workspace-oriented operations, and system-defined KB-wide operations.

Associating a Mouse-Wheel Event with an Operation

Use the rolling ... implies clause to configure mouse wheel events with an operation. The configuration clause can include:

- rolling the mouse wheel forward, that is, toward the front of the mouse.
- rolling the mouse wheel backward, that is, toward the back of the mouse.

You can associate the mouse-wheel event with an operation that is targeted on:

- The current item:

  ```
  rolling the mouse wheel forward on this item implies lift-to-top
  ```

- An item of any class:

  ```
  rolling the mouse wheel forward on any kb-workspace implies scroll-down
  rolling the mouse wheel backward on any kb-workspace implies scroll-up
  ```

- Not an item, such as the G2 window's background tiling pattern:

  ```
  rolling the mouse wheel backward implies inspect
  ```

You can include any combination of the control, alt, or shift modifier keys, for example:

```
rolling control+the mouse wheel forward implies scroll-left
rolling control+the mouse wheel backward implies scroll-right
```

Associating a Mouse Click with a Mouse-Tracking Procedure

Use the pressing ... on ... starts clause to declare that mouse clicks on items of one or more classes cause G2 to call a user-defined **mouse-tracking procedure**. You code this procedure to respond to a change in the mouse pointer's location *within a particular window*, until the next mouse-click event within that window.

This allows your KB to support state-based, user-interface operations, including drag-and-drop operations such as a simple drawing command and opening and selecting from pulldown menus.

For example, you can code a phrase like this:

```
pressing any mouse button on any custom-object starts
  track-mouse-over-custom-object as the mouse tracks over any item
```

This phrase causes G2 to call the track-mouse-over-custom-object procedure after the user depresses any mouse button over any custom-object, and to call that procedure again each time the mouse pointer passes over any other item in G2's own window.

You can also code a phrase like this:

```
pressing control + any mouse button on any custom-object starts
  track-mouse-over-custom-object as the mouse tracks
  continuously over any item
```

This phrase causes G2 to call the track-mouse-over-custom-object procedure after the user simultaneously depresses Control and any mouse button over any custom-object; to call that procedure again each time the workstation's window manager updates the mouse position; and to call the procedure again each time the mouse pointer passes over any other item in G2's own window.

You can specify the following mouse clicks in a pressing on ... starts clause:

- any mouse button
- left mouse button, middle mouse button, or right mouse button
- Any combination of the control, alt, or shift modifier keys with any mouse button.

The clause must refer to:

- A *trigger-class*: G2 calls your *mouse-tracking-procedure* when the user clicks the mouse on an item of this class.
- A *tracked-class*: G2 calls your *mouse-tracking-procedure* when the mouse pointer passes onto, off of, or continuously over items of this list of classes.
- A *mouse-tracking-procedure*: A user-defined procedure (must be of the procedure class) that G2 calls to respond to the triggering *mouse-click-event* and to the mouse pointer's subsequent movement onto, off of, or over items of the *tracked-class*.

Note You cannot specify an item of the method or method-declaration class as the *mouse-tracking-procedure*. However, your mouse-tracking procedure can call or start a method.

Coding the Mouse-Tracking Procedure

Your mouse-tracking procedure must conform to the following syntax:

```
mouse-tracking-procedure
    (event: symbol , tracked-window: class g2-window,
      trigger-item: item-or-value, tracked-item: item-or-value,
      x-mouse-position: integer, y-mouse-position: integer,
      event-timing-in-milliseconds: integer, state-of-modifier-keys: integer )
```

Argument	Description
event	Is one of: • The symbol start-tracking • The symbol enter • The symbol motion • The symbol leave • The symbol stop-tracking
tracked-window	Represents the g2-window item that is associated with the window in which G2 tracks the mouse. **Note:** It is possible that the tracked-window argument of a user mouse-tracking procedure does not exist. If your mouse-tracking procedure uses this argument, the procedure should first check for its existence, using a statement such as if tracked-window exists. If your procedure does not perform this check, logging out of Telewindows or deleting a g2-window while a user mouse-tracking procedure is active can cause G2 to signal a stack error.
trigger-item	Represents the item over which the mouse is pressed first, or the value false if that item has been deleted.
tracked-item	Represents the item the mouse is entering or leaving, or the value false if that item has been deleted. *tracked-item* also can be the value false if the mouse is entering or leaving a non-item component of the G2 environment (such as an Operator Logbook page).
x-mouse-position	Represents the x position of the mouse, in the coordinate system of the workspace.
y-mouse-position	Represents the y position of the mouse, in the coordinate system of the workspace.

Argument	Description
event-timing-in-milliseconds	Represents the time-stamp in milliseconds of the *mouse-click-event*. Use this value to determine the time interval in milliseconds between consecutive *mouse-click-events*, such as to determine a double-click. By design, this value reaches integer overflow and wraps to zero every few hours.
state-of-modifier-keys	Represents the pressed-or-released state of the three modifier keys (Alt, Control, and Shift). Only the least significant three bits of this value are meaningful. G2 returns values in the range 0 (zero) to 7 (seven), as presented in the next table.

Position of Modifier Keys

Alt	Control	Shift	Return Value
Up	Up	Up	0
Up	Up	Down	1
Up	Down	Up	2
Up	Down	Down	3
Down	Up	Up	4
Down	Up	Down	5
Down	Down	Up	6
Down	Down	Down	7

You must code your mouse-tracking procedure to respond to these events:

- start-tracking event: When the configured *mouse-click-event* occurs over an item of *trigger-class*.
- enter event: When the mouse pointer passes onto any item of *tracked-class*.
- motion event: (If the configuration includes the continuously keyword) when the host platform's window manager notifies G2 with a new location of the mouse pointer.
- leave event: When the mouse pointer passes off of any item of *tracked-class*.

- stop-tracking event: When the user releases the mouse button.
- abort-tracking event: When the user interrupts mouse-tracking by pressing Control + a.

When the configured *mouse-click-event* occurs over any *trigger-class* item, G2 automatically calls the *mouse-tracking-procedure*, the first time, and passes to it:

- The symbol start-tracking.
- The g2-window item that is associated with the window in which the *mouse-click-event* occurred.
- The item over which the mouse was pressed.
- The item over which the mouse was pressed (again).
- The *x, y* position of the mouse-pointer.
- The timing of this *mouse-click-event* in milliseconds.
- The position of the modifier keys (Alt, Control, and Shift) as a three-bit integer.

Note G2 does *not* call the mouse-tracking procedure if a mouse-click event occurs over a *disabled* item.

For each *tracked-class* item onto which the mouse pointer passes, G2 automatically calls the *mouse-tracking-procedure* again, and passes to it:

- The symbol enter.
- The g2-window item that is associated with the window in which the enter event occurred.
- The original item over which the mouse was pressed.
- The item over which the enter event occurred.
- The *x, y* position of the mouse pointer, with respect to the workspace of the item over which the enter event occurred.
- The timing of this *mouse-click-event* in milliseconds.
- The position of the modifier keys (Alt, Control, and Shift) as a three-bit integer.

For each *tracked-class* item from which the mouse pointer passes, G2 automatically calls the *mouse-tracking-procedure* again, and passes to it:

- The symbol leave.
- The g2-window item that is associated with the window in which the leave event occurred.

- The original item over which the mouse was pressed.
- The item over which the leave event occurred.
- The *x, y* position of the mouse pointer, with respect to the workspace of the item over which the leave event occurred.
- The timing of this *mouse-click-event* in milliseconds.
- The position of the modifier keys (Alt, Control, and Shift) as a three-bit integer.

When the user releases the mouse button, G2 automatically calls the *mouse-tracking-procedure* again, and passes to it:

- The symbol stop-tracking.
- The g2-window item that is associated with the window in which the *mouse-click* event occurred.
- The original item over which the mouse was pressed.
- The item currently under the mouse-pointer.
- The *x, y* position of the mouse-pointer, with respect to the workspace of the item over which the stop-tracking event occurred.
- The timing of this *mouse-click-event* in milliseconds.
- The position of the modifier keys (Alt, Control, and Shift) as a three-bit integer.

If the configuration specifies the phrase continuously over, then for each *tracked-item* over which the mouse pointer passes, G2 automatically calls the *mouse-tracking-procedure* each time the workstation's window manager updates the mouse position. When G2 calls the *mouse-tracking-procedure*, G2 passes to it:

- The symbol motion.
- The g2-window item that is associated with the window in which the motion event occurred.
- The original item over which the mouse was pressed.
- The item over which the motion event occurred.
- The *x, y* position of the mouse pointer, with respect to the workspace of the item over which the motion event occurred.
- The timing of this *mouse-click-event* in milliseconds.
- The position of the modifier keys (Alt, Control, and Shift) as a three-bit integer.

Example of Mouse-Tracking Procedure

For example, suppose you have a pull-down-menu class, whose instances are related to instances of the pull-down-menu-choice class. You can use a configuration statement with a pressing ... on ... starts clause, as follows, to implement operations that support selection of pulldown menu choices:

```
configure the user interface as follows :
    unless in administrator mode :
        pressing alt + any mouse button on any pull-down-menu starts
          start-procedure use-pull-down-menu as the mouse tracks over
          any pull-down-menu-choice or pull-down-menu
```

Including this pressing ... on ... starts clause in the configuration requires that you write a procedure named use-pull-down-menu, as follows:

```
use-pull-down-menu ( event : symbol, tracked-window : class g2-window,
    trigger-item : item-or-value, tracked-item : item-or-value,
    x-mouse-position : integer, y-mouse-position : integer,
    event-time-stamp : integer, keys-mask : integer ) = ( )

begin
    { Respond to the four mouse-tracking procedure events ... }
    case ( event ) of
        START-TRACKING :
        begin
            call highlight-a-pull-down-menu ( trigger-item ) ;
            call select-a-pull-down-menu ( trigger-item ) ;
        end ;

        ENTER :
        { First, verify that a "tracked-item" value was returned }
          if tracked-item has a current value then
            begin
                { Second, perform this operation only if the "tracked-item"
                  still exists. }
                if tracked-item exists then
                    call highlight-a-choice-on-pull-down-menu
                      ( tracked-item ) ;
            end ;

        MOTION :
        { This case is necessary only if the relevant configuration
          statement specifies "continuously over". }

        { First, verify that a "tracked-item" value was returned }
        if tracked-item has a current value then
            begin
                { Second, perform this operation only if the "tracked-item"
                  still exists. }
                if tracked-item exists then
                    call display-menu-for-traversed-item ( tracked-item ) ;
            end ;
```

```
            LEAVE :
            { First, verify that a "tracked-item" value was returned }
            if tracked-item has a current value then
                begin
                    { Second, perform this operation only if the "tracked-item"
                      still exists. }
                    if tracked-item exists then
                        call unhighlight-a-choice-on-pull-down-menu
                          ( tracked-item ) ;
                end ;

            STOP-TRACKING :
            { First, verify that a "tracked-item" value was returned }
            if tracked-item has a current value then
                begin
                    { Second, perform this operation only if the "tracked-item"
                      still exists. }
                    if tracked-item exists then
                        case ( the class of tracked-item ) of
                            pull-down-menu-choice :
                            { Perform this operation only if the "tracked-item"
                              still exists. }
                            if tracked-item exists
                                call select-choice-on-pull-down-menu
                                  (tracked-item) ;
                            otherwise :
                            { If mouse-up event occurs over other than a
                              pull-down-menu-choice ... }
                            call unselect-the-selected-pull-down-menu ( ) ;
                        end { case of }
                end { begin }

            ABORT-TRACKING :
            { This procedure does not support responding to aborts
              (the Control + a keypress) during mouse-tracking. }
            begin
            end
        end { case of }
    end { begin }
```

In this sample procedure, notice that:

- When the user releases the mouse button, the mouse might not appear over an item of the *tracked-class*. Therefore, code under the stop-tracking: case must discriminate between items of applicable and non-applicable classes.
- In each situation in which G2 can return a *tracked-item*, the mouse-tracking procedure must check, first, that G2actually returned a value, and second, that the *tracked-item* still exists.
- If the *trigger-item* and the *tracked-item* are the same item, your KB might cause that item to be deleted after the start-tracking event but *before* the enter, leave,

or stop-tracking events. In this case, G2 passes *no value* for the *trigger-item* argument.

Note If the mouse button is released over the *tracked-window*'s background tiling pattern, G2 supplies the g2-window that is associated with this process window (belonging to G2 or Telewindows) as the value for *tracked-item* and returns *x-mouse-position* and *y-mouse-position* as zero (0).

If *tracked-item* was deleted during mouse-tracking, *trigger-item* is passed as the value false for all subsequent mouse-tracking events.

If *tracked-item* was deleted between when G2 detects the enter and leave events, the *tracked-item* argument has the value false for the leave event.

Conflicts between Mouse-Tracking and Other User Interface Operations

Other user-interface operations can also occur after mouse-tracking has begun and before mouse-tracking ends. For example, the KB's processing can perform a transfer ... to the mouse action after mouse-tracking processing has begun; after the user executes mouse-down to drop the item, the mouse-tracking processing resumes.

Constraining the Movement of Items

These two clauses restrict where you can move items upon a workspace:

Clause	Purpose
constrain moving ... to the rectangle	Limits the movement of an item to an invisible rectangle.
constrain moving ... such that the item aligns on a grid	Limits where an item can be moved upon its workspace.

These constrain moving clauses do *not* restrict:

- The placement of cloned items on a workspace.
- The placement of items transferred to a workspace.
- The movement of items within an operate on area region.

Note You cannot use this configuration statement for workspaces, because the coordinate systems are workspace coordinates. Use this configuration for items upon a workspace.

Aligning Items to an Invisible Rectangle

The constrain moving ... to the rectangle clause restricts moving an item to within an invisible rectangle whose left, right, top, and bottom edges you specify.

Specify the four edges of the invisible rectangle as *x*, *y* coordinates upon the workspace. For example, the following clause restricts moving upright-beam items outside of a particular region of a workspace that displays the floor-plan of a building:

```
constrain moving any upright-beam to the rectangle (-100, 100, -100, 100)
```

Aligning Items on an Invisible Grid

The constrain moving ... such that the item aligns on a grid clause specifies an invisible grid within the workspace. G2 forces placement of items of the specified class at the intersection points on this grid. This capability is similar to the snap feature in software packages for drawing schematic diagrams.

G2 measures the spacing between the intersection points on this grid in G2 workspace units. G2 uses the center of the item as the reference point for alignment.

For example, the following clause restricts placement of upright-beam items in a workspace that displays the floor-plan of a building:

```
constrain moving any upright-beam
  such that the item aligns on the grid ( 50, 50 )
```

In this example, assume that the x-grid-length and y-grid-length attributes of floor-plan have the value 50, which represents a length in G2 workspace units. As you use the mouse to move an upright-beam within a workspace, G2 changes the item's location within its workspace only after the mouse moves at least 50 workspace units, either horizontally or vertically.

Tip The G2 system procedures g2-set-movement-limits, g2-get-movement-limits, and g2-clear-movement-limits also programmatically restrict an item's movement within a workspace.

Configuring the User Interface of Proprietary Items

G2 allows you to identify a workspace as proprietary. A proprietary workspace and the items below it in the KB's workspace hierarchy are called **proprietary items**. You can use configurations to restrict access to proprietary items. For more information about creating a proprietary KB, see Package Preparation.

G2 defines the proprietary status of workspaces independently of the user mode of the g2-window that is associated with the current process window (for G2 or Telewindows). Because of this, you can configure proprietary items independently of any user modes declared in the KB's configurations. By making a workspace proprietary, you can effectively lock that workspace's items from any access whatsoever, including access by a programmatic action that isn't associated with a user mode.

Use the restrict proprietary items as follows statement to configure proprietary items. This statement supports these configuration clauses:

menu choices for	table menu choices for
non-menu choices for	pressing ... implies
selecting ... implies	pressing ... on ... starts
attributes visible for	releasing ... implies
typing ... implies	

Tip These clauses conform to the same syntax, and have the same limitations, as described for the configure the user interface as follows statement, described in Configuring the User Interface of Items.

For example, the following configuration statement restricts the menu choices and non-menu choices available for proprietary tracked-vehicle items, which are tracked-vehicles placed under any proprietary workspace in the KB's workspace hierarchy:

```
restrict proprietary items as follows :
    menu choices for tracked-vehicle include additionally :
        create-instance ;
    non-menu choices for tracked-vehicle include :
        nothing
```

Configuring Access to and from Other G2, G2 Gateway, and Telewindows Processes

Use the set up network access as follows configuration statement to allow or prohibit access to the entire KB or to one or more items in the KB, by other G2 processes, by G2 Gateway bridge processes, and by Telewindows processes.

Use the set up network access as follows statement as summarized in this table:

Type of Access	Short Description	Relevant Items or Classes
read	Other G2 processes can use items in the current KB as a source of data service for variables in their own current KBs. Not applicable to G2 Gateway or Telewindows.	KB-wide, items of any system-defined class, or items of any user-defined class.
write	Other G2 processes can set a new current value of a variable of a user-defined class. Not applicable to G2 Gateway or Telewindows.	KB-wide, variables of user-defined classes that mix in either the g2-to-g2-data-service or gsi-data-service class.
execute	Other G2 processes or G2 Gateway bridge processes can call a G2 procedure in the current KB.	Items of the procedure class.
inform	Other G2 processes can target an inform action on a variable of a user-defined class.	Items of user-defined classes that mix in the g2-to-g2-data-service class. Though the editor permits you allow or prohibit inform access to or from G2 Gateway, such access is inappropriate, because there is no way for G2 Gateway to inform a G2 process.
connect	Other G2 processes, G2 Gateway and Telewindows processes can connect to this G2 process.	*KB-wide only*: Include a setup network access as follows configuration statement *only* in the item-configuration attribute of the KB Configuration system table.

Allowing or Prohibiting Network Access

By default, all G2 processes, G2 Gateway, and Telewindows allow network access to the current KB. You can allow or prohibit *all* access to the current KB by other G2 processes, G2 Gateway, and Telewindows.

Because network access applies to the entire KB, you must include a global configuration statement such as the following in the item-configuration attribute of the KB Configuration system table.

To prohibit network access to a KB:

➔ Enter a configuration statement in the KB Configuration system table such as:

```
set up network access as follows :
    prohibit connect access by g2 and gsi and telewindows
```

G2 does not allow this configuration statement in any other item-configuration or instance-configuration attribute.

Note To prohibit access to G2 Gateway, you must use the symbol gsi. Also, prohibiting connect access by G2 Gateway prevents G2 Gateway from initiating a connection to G2, but it does not prevent G2 from connecting to G2 Gateway through a gsi-interface item in G2.

Allowing connect access permits another G2 process, G2 Gateway, or Telewindows to establish a connection with this G2 process. Because connect access applies to the entire KB, it restricts or enables all items in the current KB.

To allow or restrict connect access:

➔ Include a configuration statement in the item-configuration attribute of the KB Configuration system table such as:

```
set up network access as follows :
    allow connect access by g2 and telewindows
```

Note Because connect access does not apply to particular items, G2 does not support the absolutely keyword for this phrase.

Allowing Read and Write Access

Allowing read and write access permits other G2 processes to access items in the G2 granting the access:

- Allowing read access to a set of items in the current KB permits those items to be the source of values for variables in the current KB of the other G2 process.
- Allowing write access to a set of variables in the current KB permits the other G2 process to set those variables' values with set actions.

Variables of a user-defined class that mixes in g2-to-g2-data-service are candidates for allowing write access.

To restrict read or write access to items by other G2 processes:

➔ Enter a configuration statement such as:

```
set up network access as follows :
    prohibit read or write access to any vehicle by g2
```

Allowing Execute Access

Allowing execute access to procedures in the current KB permits another G2 or a G2 Gateway bridge to invoke them using a Remote Procedure Call (RPC).

To restrict execute access:

➔ Enter a configuration statement such as:

```
set up network access as follows :
    prohibit execute access to update-vehicle-direction by g2 and gsi
```

Note Prohibiting execute access on a rule that can be activated by forward chaining from a variable does *not* stop the rule from firing. To do this, you must instead declare a configuration that prohibits access to the variable.

Allowing Inform Access

Allowing inform access to variables in the current KB permits another G2 process to pass messages to those variables through an inform action.

To restrict inform access to items:

➔ Enter a configuration statement such as:

```
set up network access as follows :
    prohibit inform access to any custom-message-receiving-variable by g2
```

Configuring Properties of Items

You can declare that an item has one or more properties, using the statement.

declare properties ... as follows

- These item properties relate to optimization and the compilation of rules, procedures, and certain system-defined attributes:

inlineable	Whether the procedure or method can be inlined.
stable-hierarchy	For methods and related-items, stable-hierarchy implies that neither the class-hierarchy nor the method will be specialized.
independent-for-all-compilations	For items with compilation dependencies on other items, whether compilation of the item's attributes depends on the stability of those other items
stable-for-dependent-compilations	Whether an item is the basis for the stability of other items that have a compilation dependency on it

- These item properties relate to subworkspaces:

activatable-subworkspace	Whether an item's subworkspace can be activated and deactivated (using the activate and deactivate actions)
subworkspace-connnection-posts	Whether an item can be connected to other items on its subworkspace via connection posts on the subworkspace

- This property relates only to items that are using connections:

manual-connections	Whether the user can interactively draw connections to or from an item

Specifying the Scope of the Declared Properties

You specify declare configuration statements that have different scopes, as shown in these examples:

```
{ Scope: Follow the KB's workspace hierarchy or class hierarchy ... }
declare properties as follows :
    inlineable ;

{ Scope: Only the configured item is affected by the configuration. }
declare properties of this item as follows :
    independent-for-all-compilations ;

{ Scope: Within the KB's workspace hierarchy or class hierarchy, any item
  of the specified class(es) are to be affected by the configuration. }
declare properties of any tracked-vehicle as follows :
    stable-for-dependent-compilations
```

Specifying Exceptions to the Declared Properties

To specify a statement that represents an exception to a declare statement found in a configuration placed on an item higher in the hierarchy, begin the statement with the phrase:

```
declare properties ... as follows : not ...
```

Examples:

```
declare properties as follows :
    not inlineable ;
declare properties for this item as follows :
    not stable-for-dependent-compilations
declare properties for any tracked-vehicle as follows :
    not independent-for-all-compilations
```

Declaring a Procedure to be Inlined

An **inlined procedure** is one whose compiled code is embedded in any compiled code that calls the procedure. Inlining a procedure improves performance by eliminating the need to execute a call when the procedure is invoked: control instead passes directly to the embedded code for the procedure. The trade-off is increased compiled code size due to redundant inlined copies of the procedure.

When you inline a procedure, you *must* also use the configuration clause: stable-for-dependent-compilation.

By default, all items in the current KB are configured as:

```
declare properties as follows :
    not inlineable
```

To declare that a procedure can be inlined:

➔ Add this item configuration to the procedure:

```
declare properties as follows : inlineable,
    stable-for-dependent-compilations
```

Inlining a procedure is further described in Using Compilation Configurations.

Declaring a Method to be Inlined

An inlined method is identical to an inlined procedure: its compiled code is embedded in any compiled code that calls the method. When you inline a method, you *must* also include the configuration clauses stable-hierarchy and stable-for-dependent-compilations.

A method of stable-hierarchy guarantees that a more specialized method will not be added below the current method in the method hierarchy. If the method includes return values, the stable-hierarchy declaration additionally guarantees the return value types.

By default, all items in the current KB are configured as:

```
declare properties as follows :
    not inlineable
```

To declare that a method can be inlined:

➔ Add this item configuration to the method:

```
declare properties as follows : inlineable, stable-hierarchy,
    stable-for-dependent-compilations
```

Inlining a method is further described in Using Compilation Configurations.

Declaring Items as Stable Hierarchy

Declaring an item as stable-hierarchy indicates that neither the class hierarchy of the item, nor the item itself, will be specialized. If a method is declared with stable-hierarchy, then G2 may be able to compile more efficiently the procedures or methods that call the inlined method.

You can also declare classes as stable-hierarchy, which may let G2 compile any methods of that class more efficiently.

By default, all items in the current KB are configured as:

```
declare properties as follows :
    not stable-hierarchy
```

To declare an item to have a stable hierarchy:

➔ Add this configuration statement to the item:

```
declare properties as follows :
    stable-hierarchy
```

Note This statement must be used when declaring a method as inlineable.

Using the stable hierarchy configuration is further described in Using Compilation Configurations.

Declaring an Item Independent for All Compilations

Declaring an item as independent-for-all-compilations means that the item's knowledge does not depend on the knowledge in any other item in the KB. This allows G2 to compile that item more efficiently.

By default, all items in the current KB are configured as:

```
declare properties as follows :
    not independent-for-all-compilations
```

To declare an item independent for all compilations:

➔ Add this configuration statement to the item:

```
declare properties as follows :
    independent-for-all-compilations
```

This feature is described in detail under Using Compilation Configurations.

Declaring an Item Stable for Dependent Compilations

Declaring an item as stable-for-dependent-compilations means that certain parts of the item's knowledge will not change during the KB's processing. This allows G2 to compile more efficiently other items that depend on that knowledge.

By default, all items in the current KB are configured as:

```
declare properties as follows :
    not stable-for-dependent-compilations
```

To declare an item as stable for dependent compilations:

➔ Add this configuration statement to the item:

```
declare properties as follows :
    stable-for-dependent-compilations
```

This feature is described in detail in Using Compilation Configurations.

Declaring an Activatable Subworkspace for an Item

Most G2 classes can support an activatable subworkspace. Activatable subworkspaces support the programmatic activation and deactivation. You use the activate and deactivate actions to activate and deactivate an activatable subworkspace. By default, all items in the current KB are configured as:

```
declare properties as follows :
    not activatable-subworkspace
```

To declare that an item supports an activatable subworkspace:

➔ Add this configuration statement to the item:

```
declare properties as follows :
    activatable-subworkspace
```

Note If items of the configured definition item's class do not support creating an associated subworkspace, G2 ignores that declare properties as follows : activatable-subworkspace configuration statement.

Declaring Subworkspace Connection Posts for Items

If items of a class can have a subworkspace, and (for a user-defined class) if the icon-description attribute of the class's definition defines connection stubs, you can configure items of that class so that G2 automatically creates permanent connection-posts on the subworkspaces of items of that class.

Through an item's subworkspace connection-posts, you can connect the item of the configured class to items on the subworkspace. Using subworkspace connection-posts is described in Creating Connection Posts on Subworkspaces Automatically.

By default, this property is configured as:

```
declare properties as follows :
    not subworkspace-connection-posts
```

To declare that an item supports subworkspace connection posts:

➔ Add this configuration statement to the item:

```
declare properties as follows :
    subworkspace-connection-posts
```

Note If the configured items do not have a subworkspace, G2 ignores the declare properties as follows : subworkspace-connection-posts configuration statement.

Disallowing Manual Connections for an Item

For any subclass that is defined to support connections, you can prohibit KB users from drawing connections interactively to or from items of that class, except where pre-existing stubs exist.

By default, this property is configured as:

```
declare properties as follows :
    manual-connections
```

To declare that an item disallows manual connections:

➔ Add this configuration statement to the item:

```
declare properties as follows :
    not manual-connections
```

Note If items of the configured class do not support manual connections, G2 ignores that declare properties as follows : manual-connections configuration statement.

Including Comments in Configurations

G2 allows you to store tagged text as comments in item configurations and instance configurations. Each comment has a tag symbol and an associated text string, such as:

```
comment as follows :
    configuration-purpose : "A comment describing this configuration's
      purpose" ,
    author-of-configuration : "HCC" ,
    configuration-last-modified : "18 Jul 1997" ,
    latest-benchmark-statistics-load : "3.243" ,
    latest-benchmark-statistics-save : "5.56" ,
    latest-benchmark-statistics-run : "14.8973"
```

A comment as follows statement can contain one or more symbols, each of which identifies a string that contains the text of the comment.

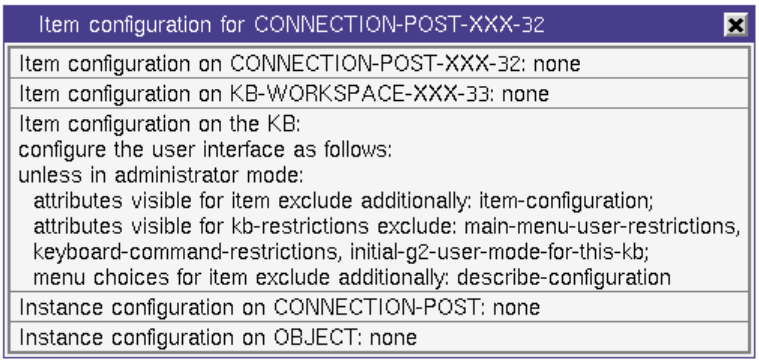
The symbol can represent a keyword that is significant for your application; the text can represent textual information that identifies a significant fact or feature for the items within the scope of that configuration.

Tip This feature is intended to support comments consisting of free-form text. For example, you can assign symbols for use as search keywords.

Describing Configurations

By default, when you click the mouse on an item of any system-defined or user-defined class, the menu that appears includes the describe configuration choice.

Selecting describe configuration presents a table that lists the configurations that apply to that item. For example, after creating a new connection post, select describe configuration from its menu. G2 displays a table like the one shown below:



This table lists all configuration statements that G2 finds applying to this item. Each entry indicates whether its statements are stored in an item configuration or instance configuration. The entry at the top shows the statements that have highest precedence; the entry at the bottom shows the statements that have lowest precedence.

Tip As you develop configurations in your KB, use describe configuration to trace which configurations actually apply to a particular item.

Declaring User Modes in Configurations

Certain configuration statements declare categories of usage, or **user modes**, for your KB. Specifically, you refer to user modes in the configure the user interface as follows statement, as described in Configuring the User Interface of Items.

Each user mode can represent a style of interaction with the KB's knowledge. The meaning of each style depends on how your application organizes its knowledge.

For instance, if your application organizes its knowledge into concentric *layers* of knowledge, for example, with outer layers representing unrestricted knowledge and inner layers representing restricted knowledge, each user mode can represent a security level into one or more layers of the KB's knowledge.

Alternatively, if your application organizes its knowledge according to the *roles* of those that work with the application, perhaps distinguishing among

developers, users, and site administrators, each user mode can represent the set of workspaces accessible to persons acting in a particular role.

You declare a user mode simply by referencing it in a configure the user interface as follows statement in an item configuration or instance configuration. Your KB's configurations can declare as many user modes as your application requires.

Associating User Modes with G2-Window Items

G2 associates a user mode not with an individual user, but rather with a particular window. This is because the value of each g2-window item's g2-user-mode attribute indicates the user mode in effect for the window with which it is associated.

Recall that launching a G2 process causes G2 to create one g2-window item. G2 automatically associates this g2-window item with the visible window (produced by the workstation's window manager software) that displays the contents of the current KB.

Finally, recall that when a user starts or connects to a G2 process that uses a secure G2 authorization file, G2 presents the login dialog, in which the user enters a user name and optionally a user mode, default language, and so on. In this case, G2 first verifies that the specified combination of user name and user mode are registered in the G2 authorization file. If they are, G2 creates a g2-window item and sets the value of its g2-user-mode attribute to the user mode indicated in the login dialog.

When logged into a secure G2 with a KB loaded, the user can only change to a user mode that is explicitly mentioned in a configuration statement in the KB. If the user attempts to change to a user mode that is not explicitly mentioned in the KB, an error occurs in the editor indicating the user mode is unknown, even if the specified user mode has been authorized for that user in the OK file.

Tip See G2-Windows for more information about g2-window items and their relation to visible windows. See the *Telewindows User's Guide* for more information about starting and operating Telewindows.

Unless your G2 process is using a secure G2 authorization file, the g2-user-mode attribute of each g2-window item that G2 creates has the default value administrator. The administrator user mode is always declared and in use while a G2 process is running.

Note The window associated with a g2-window item whose g2-user-mode attribute contains the value administrator can access and display all the knowledge in the current KB. To prevent you from mistakenly restricting access to your own KB, you cannot use configurations to affect the behavior of items under administrator mode.

Associating User Modes and Users

If your KB uses configurations that declare user modes, you must consider how to associate each user of the KB with a user mode. If you require the users of your G2 application to login, G2 can automatically associate an appropriate user mode with each user account. For more information, see G2-Windows.

Tip The knowledge in the G2 authorization file, typically named `g2.ok`, determines whether users log into G2 or not. The G2 site administrator is responsible for maintaining the `g2.ok` file. For more information, see the `readme-g2.html` file and the *G2 Bundle Release Notes*.

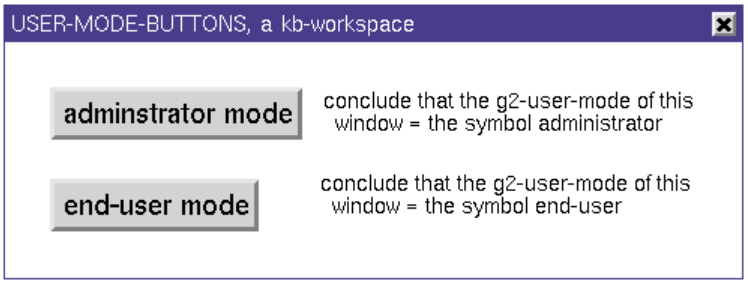
If users do not login to your G2, your KB must be responsible for associating users with user modes. Given this design, your KB could provide a dialog that allows the user to switch among alternative views of the KB's knowledge.

Example of Configuring the User Interface of an Item

You can understand how configurations work by following a simple example. The example shows you how to configure a symbol-list so that it behaves differently when appearing in different windows associated with g2-window items that have different settings in their respective g2-user-mode attributes.

This example also illustrates G2's rules of precedence for configurations, in that you will store configurations in items at different levels in the KB's class and workspace hierarchies.

This example uses action buttons to change the target g2-window item's g2-user-mode attribute: one action button for administrator mode and one for end-user mode. The figure below shows how you should define each action button.



To configure the user interface of an item:

1 Start the current KB.

2 Click the mouse on your action button to change the user mode of this G2 window to administrator.

3 Create a new workspace.

4 Create a symbol-list on the workspace named my-symbol-list by choosing KB Workspace > New Object > g2-list > value-list > symbol-list.

5 Open the table for the symbol-list, and edit its name to my-symbol-list.

In my-symbol-list's table, notice that you can modify the allow-duplicate-elements attribute by selecting the change-to-no and change-to-yes table menu choices.

6 In the table for the symbol-list, select describe configuration.

G2 lists the configurations in effect for the item, as follows:

- No item configurations currently apply for my-symbol-list.
- No item configurations apply for my-symbol-list's parent workspace.
- The KB Configuration system table contains G2's default KB-wide item configurations.

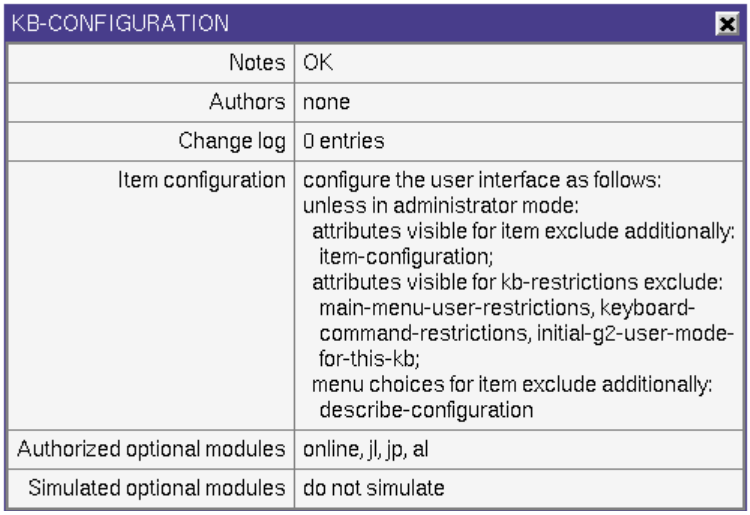
7 Hide my-symbol-list's table by clicking the mouse in the table's title bar.

8 Click the action button to change the user mode setting for this g2-window item to end-user.

9 Open my-symbol-list's table again.

Notice that the item's item-configuration attribute does not appear. This is because, after you launch G2 with an empty current KB (or after you clear the

current KB), G2 includes the following configuration statement in the item-configuration attribute of the KB Configuration system table:



KB-CONFIGURATION

Notes	OK
Authors	none
Change log	0 entries
Item configuration	configure the user interface as follows: unless in administrator mode: attributes visible for item exclude additionally: item-configuration; attributes visible for kb-restrictions exclude: main-menu-user-restrictions, keyboard-command-restrictions, initial-g2-user-mode-for-this-kb; menu choices for item exclude additionally: describe-configuration
Authorized optional modules	online, jl, jp, al
Simulated optional modules	do not simulate

10 Hide my-symbol-list's table again, and change the user mode to administrator.

11 Display the table for the workspace that contains my-symbol-list.

12 In this table, edit the item-configuration attribute so that it contains:

```
configure the user interface as follows :
    when in end-user mode:
        attributes visible for g2-list exclude additionally :
            element-type;
        table menu choices for the allow-duplicate-elements of any
         g2-list exclude :
            change-to-no, change-to-yes, edit
```

13 Change the user mode to end-user, then display my-symbol-list's table again.

Notice that neither the item-configuration attribute nor the element-type attribute appears. This is due to the item configuration you added to my-symbol-list's parent workspace. Due to the item configuration you added to my-symbol-list, you cannot modify the item's allow-duplicate-elements attribute.

14 Change the user mode to administrator, and display my-symbol-list's menu.

Notice that the describe configuration menu choice appears in the item's menu.

15 Change the user mode to end-user, and again display my-symbol-list's menu.

Now notice that the describe configuration menu choice does not appear. This demonstrates how a configuration placed at a more specific level in your KB's class or workspace hierarchy overrides a configuration placed at a more general level in that hierarchy.

Obtaining the Attributes Visible for a User Mode Programmatically

To obtain the attributes that are visible for an item in a particular user mode:

➔ g2-get-attributes-visible-in-mode
(*class-or-item*: item-or-value, *user-mode*: symbol)
-> *list-of-attributes*: sequence

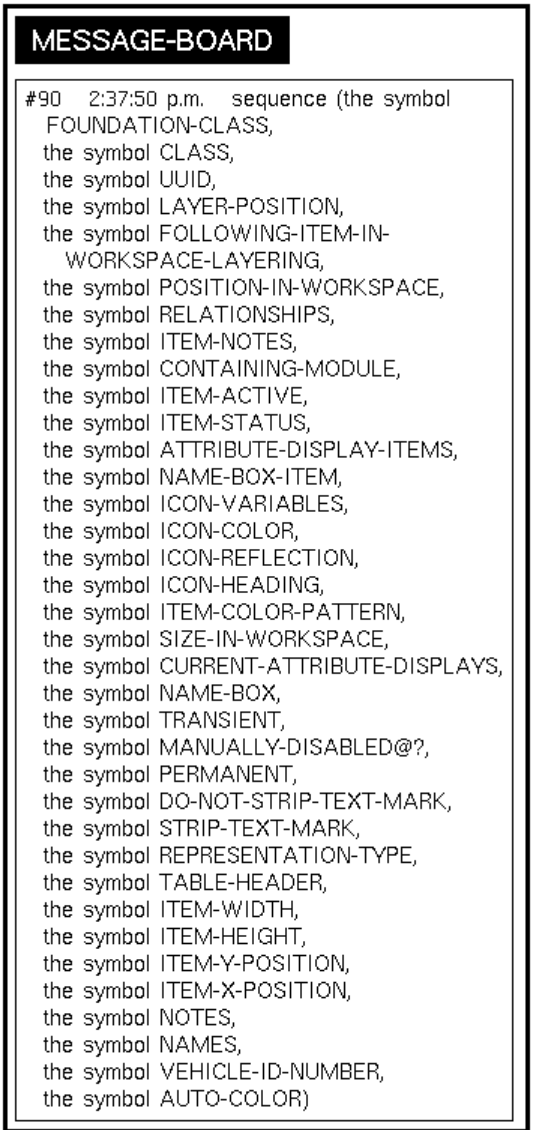
As an example, the geo-classic automobile class defines its instance-configuration attribute as follows:

```
configure the user interface as follows:
    when in inventory-checker mode:
        attributes visible for geo-classic exclude absolutely: test-case
```

To obtain a list of visible attributes for a particular user mode:

```
get-attributes-in-mode (geo-classic-instance: class geo-classic)
value-for-mode: sequence;

begin
    value-for-mode =
        call g2-get-attributes-visible-in-mode(geo-classic-instance,
            the symbol inventory-checker);
        change the text of message1 to "[value-for-mode]"
end
```

This procedure returns and displays the returned sequence containing the attributes visible in the inventory-checker user mode. Notice that the system procedure returns *all* of the attributes that a class inherits from every class in its inheritance path, including those that are displayed on the table of hidden attributes for an item:



For information about hidden attributes, see Hidden Attributes.

Note All configurations that exist for an item, through item- and instance-configurations, and proprietary settings, remain in effect during attribute access operations.

Declaring Generic and Exception Configurations

After you enter configurations that apply very generally within your KB, you might find that you must configure a unique behavior for some subset of the KB's items or for particular items. These configurations represent exceptions to your KB's more generally applicable configurations.

For example, in the KB shown in the figures in Example of the Scope of Configurations, assume that:

- An item configuration stored in the Top Level workspace *prohibits* network access to the navigation buttons labelled Definitions and Schematic, those buttons' subworkspaces, and all items contained in those subworkspaces.
- An item configuration stored in the Schematic workspace *allows* network access to the items car-class-1 and service-car-1, and so on.

Following G2's rules of precedence for configurations, among the configurations that apply to car-class-1, the item configuration stored in the schematic workspace *overrides* the item configuration stored in the top-level workspace.

You can also store configurations with a smaller scope that supplement configurations with a larger scope. This allows you to declare a unique behavior for a subset of items, without affecting the configurations for all other items in your KB.

Combining Configurations

As described in Precedence of Configurations, given two configurations that each affect some feature of an item in two distinct ways, G2 applies only the configuration with the higher precedence to the item.

However, you can use G2's precedence rules for configurations in a more complex manner. Consider the user gesture of clicking the mouse on an item to display its menu. Do you prefer to show all menu choices configured for the menu or just those mentioned in the *first* menu configuration that G2 finds in its search up the class and workspace hierarchies? The same question arises for which attributes to show when displaying an item's table.

G2 allows fine control over these cases by using the phrases include, include additionally, and exclude absolutely.

Combining Cooperatively

You can use two configuration statements, configure the user interface as follows and restrict proprietary items as follows, to define configurations that G2 applies in a cooperative manner. These two kinds of statements configure the interactive behavior of an item.

You can also use these statements to configure a feature for a set of items without overriding the configurations on the same feature that are of a lower precedence.

To illustrate, suppose you create a KB that represents the foundation software layer for applications built by G2 developers, with these requirements:

- Your KB must define new user-defined, item classes, and you must declare a set of default user-interface characteristics for those new classes, using instance configurations. Your instance configurations *prohibit* the display of system-defined menu choices (and perhaps also configure new user menu choices) for items of your new classes.
- Other G2 developers must define new subclasses based on the new classes you provide. The other developers must also be able to add their own instance configurations in definitions of their own new subclasses. These configurations will configure their own user menu choices.
- The other developers' configurations must be allowed to *supplement* the configurations that you have defined for your new classes. You want to prevent the other developers from adding configurations to their own subclasses that conflict with the configurations you provided, but you must allow the other developers to add their own configurations that are particular for the subclasses that they will define.

G2 supports this kind of application development scenario by allowing you to declare configurations that cooperate with other configurations without being overridden.

Combining Additionally

A configure the user interface as follows or restrict proprietary items as follows statement can *additionally include* or *exclude* one or more user-interface features for the target item.

Including Additionally

This instance configuration statement causes the menus for conveyor-station items to present only two of the menu choices (delete and create-subworkspace) already defined for that class:

```
configure the user interface as follows:
    when in end-user mode:
        menu choices for conveyor-station include:
            delete, create-subworkspace
```

For a more specific class, you could enter the following instance configuration statement that additionally includes another menu choice (table):

```
configure the user interface as follows:
    when in end-user mode:
        menu choices for conveyor-station include additionally:
            table
```

The statement declares that, in addition to any menu choices included in configurations for conveyor-station items at more general levels in the KB's class or workspace hierarchies, the table menu choice is *additionally* available for items at this level and at more specific levels in those hierarchies.

Excluding Additionally

A configuration statement can *additionally exclude* a capability for a set of items. To do so, use the exclude additionally phrase in the clauses of a configure the user interface as follows statement or restrict proprietary items as follows statement.

For example, the following statement causes the menus of conveyor-station items to present *all but two* of the menu choices already defined for that class:

```
configure the user interface as follows:
    when in end-user mode:
        menu choices for conveyor-station exclude:
            clone, show-status
```

For a more specific class, you could enter the following statement that removes other menu choices:

```
configure the user interface as follows:
    when in end-user mode:
        menu choices for conveyor-station exclude additionally:
            table
```

The statement declares that, in addition to any menu choices excluded in configurations for conveyor-station items that are declared at a more general level in the KB's class or workspace hierarchies, the table menu choice is also *not* available for items at this level and at more specific levels in those hierarchies.

Implementing Localized Exceptions

You can use configuration statements with additionally clauses to alternately include and exclude system-defined or custom features of items at progressively more specific positions in your KB's class and workspace hierarchies. These nested configurations represent localized exceptions to configurations declared higher in those hierarchies. To do this use:

- An additionally configuration at a given position in the KB's class or workspace hierarchy *combines* with the configuration(s) in force at that position that contains an includes or excludes phrase.
- Any number of additionally configurations can combine with the configuration(s) in force that contains an includes or excludes phrase.
- An additionally configuration overrides a conflicting additionally configuration(s) at a higher position in the KB's class or workspace hierarchy, subject to G2's rules of precedence for configurations.

Combining Absolutely

You might want to prevent other G2 developers from restoring a particular feature that you have excluded for some set of items. For this reason, you can declare a configuration statement that *absolutely excludes* a capability.

To do so, include the exclude absolutely phrase in a configuration statement, as follows:

```
configure the user interface as follows:
    when in end-user mode:
        menu choices for conveyor-station exclude absolutely:
            delete, create-subworkspace ;
        selecting any conveyor-station absolutely implies:
            move
```

Because absolutely configurations cannot be overridden or supplemented, the statement above restricts conveyor-station items so that, when a user interacts with this KB via a window whose associated g2-window contains the value end-user in its G2-user-mode attribute:

- The delete and create subworkspace menu choices are never available for the target conveyor-station items.
- Selecting any of the target conveyor-station items always initiates an interactive move operation.

An absolutely configuration on a particular item feature does not combine cooperatively with other additionally configurations on the same feature. Instead, an absolutely configuration overrides all other configurations that include or exclude the same feature for the same item(s), *regardless of where you place the* absolutely *configuration in the KB's class or workspace hierarchies*. Further, you cannot supplement an absolutely configuration for a particular item feature and

for a particular set of items by other additionally configurations on the same feature.

Note You cannot specify a configuration statement that absolutely includes some item behavior.

An absolutely configuration is particularly valuable when securing a proprietary KB, since otherwise a user could restore access to the KB's proprietary knowledge.

Configuring the G2 Main Menu and Global Key Bindings and Shortcuts

Two attributes of the KB Configuration system table allow you to configure menu choices available on or under G2's Main Menu, and to configure access to G2's system-defined global key bindings and shortcut keys.

Configuring the G2 Main Menu

The main-menu-user-restrictions attribute can contain configuration statements that explicitly include or exclude choices that are, by default, available on the Main Menu or one of its System Tables, Run Options, or Miscellany submenus.

This figure shows the Main Menu choices that you can configure in this attribute:

If a user gesture results in selecting a choice that, by default, appears on the Main Menu, G2 searches for relevant configurations throughout the configurations search path, then searches for a relevant configuration in the main-menu-user-restrictions attribute.

Restricting Help

When using the G2 Online Documentation (GOLD) utility, the Main Menu can include a Help option. Typically, the Help option appears automatically when GOLD is loaded, but it is actually triggered by the presence of this procedure:

```
g2-launch-online-help (win: class g2-window)
```

Whenever a g2-launch-online-help procedure exists, with a single g2-window argument, you can configure the Main Menu to exclude the Help choice.

To restrict the Help menu choice on the Main Menu:

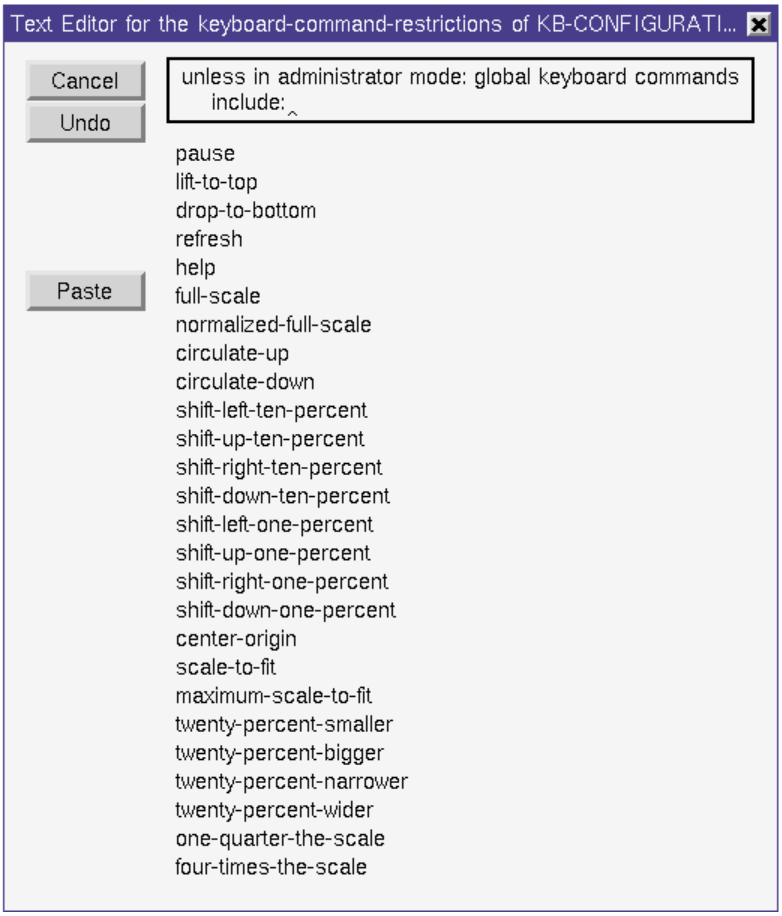
➔ Enter this configuration statement:

```
configure the user-interface as follows:
    unless in administrator mode:
    main menu choices exclude additionally: launch-online-help
```

Keyboard Command Restrictions

The keyboard-command-restrictions attribute can contain configuration statements that explicitly allow or disallow the use of global keyboard commands. These commands are designed, by default, to be available in almost all contexts.

This figure shows the global keyboard commands that you can configure in this attribute:



Using Configurations in Modularized KBs

As explained in Working with Modularized KBs, if your current KB is modularized, G2 installs into the current KB only the system tables loaded from the KB file that contains the top-level module. This means that any configurations you declare in the KB Configuration system tables in KB files that contain directly required modules are not installed.

Therefore, when organizing the configurations used in a modularized KB, you should:

- Declare all the relevant configurations for all directly and indirectly required modules, in the KB Configuration system table of the top-level module.
- Include knowledge that copies and assigns configurations from the KB Configuration system tables of directly required modules into the KB Configuration system table installed in the current KB.

8

G2-Windows

Describes how G2 associates g2-window items with visible windows.

Introduction

A **g2-window** is an item of the g2-window class. Within G2, a g2-window item represents knowledge about the window within which you interact with G2.

G2 can automatically associate a g2-window item either with your G2 window (the local window) or with the window displayed by a Telewindows connected to your G2 (a remote window).

Telewindows is a Gensym product that allows you to connect to a running G2 process, and to view and interact with the contents of its current KB. For information about Telewindows, see the *Telewindows User's Guide*.

Your KB can use the information in g2-window items to:

- Monitor the connections of the KB's users.
- Display the text of the KB, using a language translation appropriate for each user.

A **window** is a distinct display of information on a computer screen. The features of a window are determined by the window manager software installed on that computer. Commercially available window manager products include OpenWindows from Sun Microsystems, Motif from the Open Software Foundation, and the Windows product line from Microsoft Corporation.

Windows and G2-Windows

Window-system windows and g2-window items are sometimes confused with one another. Their purposes and relationship are as follows:

- A window is a display area on the screen that the operating system creates and manages on behalf of a client application such as G2.
- A g2-window is a G2 item that G2 uses to track a window that the operating system maintains on G2's behalf.

A g2-window acts in two ways:

- As an internal billboard that makes information about the corresponding window available within G2.
- As a command interface to the computer's window system: changing some attributes of a g2-window item causes G2 to tell the window system to correspondingly change the window itself.

A g2-window item provides the only interface through which an executing KB can poll and change a window within which G2 appears. The rest of the interface to the window system is platform-dependent, and is hidden within G2. Thus a g2-window item provides a platform-independent interface to the native window system.

Be careful not to confuse a window-system window with the g2-window item that represents it within G2. To keep the distinction clear, this book always refers to the latter explicitly as a g2-window, omitting the special font except in code examples.

Using Local Windows and Remote Windows

A G2 can display its current KB in a **local window**. A G2's local window appears, by default, on the same machine where the process is running. When you launch a G2 process, use the `-display` command-line option to cause the new G2's local window to appear on another computer. You can also start a G2 with *no* local window, using the `-no-window` command-line option. These options are described in Appendix A, Launching a G2 Process.

A G2 can display its current KB in one or more **remote windows**, or **telewindows**. To a G2 process, a remote window is the window belonging to a Telewindows process that has successfully connected to it. After accepting the connection, the G2 displays its current KB in the Telewindows process's own window. Telewindows Support describes the G2 features that accept and manage connections from Telewindows processes.

Representing Local and Remote Windows

When a G2 starts, by default it displays the current KB in a local window. By default, it also automatically creates a new g2-window item and associates it with that local window. Within G2, this g2-window item represents the visible window in which G2 displays the current KB. Thus, the KB can use the g2-window item's knowledge about the properties of the windows in which the KB's contents appear.

When a G2 accepts a connection from a Telewindows, it automatically creates a new g2-window item and associates it with the Telewindows window. When that Telewindows disconnects from a G2, that G2 automatically deletes its associated g2-window item.

Special Properties of Local and Remote Windows

A g2-window item that G2 creates automatically, for use as either a local or a remote (Telewindows) window, has several unique characteristics:

- It does not reside upon any KB workspace.
- It's status is neither permanent nor transient, and cannot be changed using the make permanent or make transient actions.
- It persists unchanged when a KB is reset, as with a permanent item.
- It is not saved when a KB is saved, as with a transient item.

Because the local g2-window is not saved with a KB, changes to its attributes remain in effect only during the current invocation of G2. Loading a KB does *not* restore any customized g2-window attributes that were in effect when the KB was saved.

Tip You can customize a g2-window by specifying various command-line options when you invoke G2, as described in Appendix A, Launching a G2 Process, or Telewindows, as described in the *Telewindows User's Guide*.

Displaying Independent Views of the Current KB

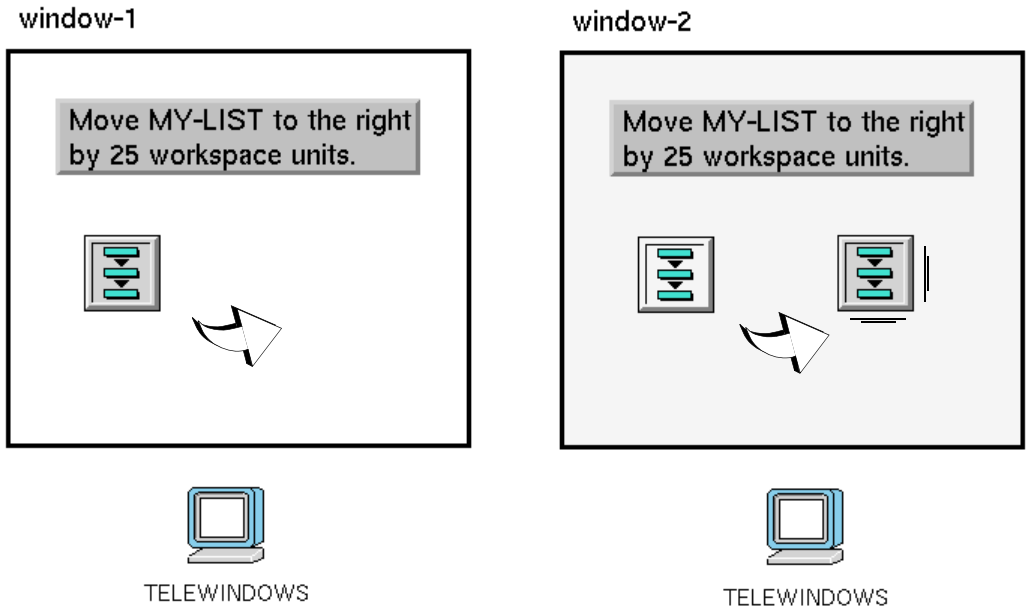
Each window that a G2 uses to display the current KB shows a distinct view of the KB. Each window:

- Displays a distinct set of the KB workspaces of the current KB, each at a position and scale that is distinct for this window.
- Has its own Scrapbook workspace.
- Has its own Text Editor and Inspect workspaces.

Otherwise, each window associated with or connected to a G2 process displays the same instance of the Operator Logbook and Message Board, and of everything else contained in the current KB.

If the same KB workspace is visible in two windows that are associated with or connected to a G2, the items upon that workspace appear the same size (allowing for any difference in workspace scale in the two windows), the same color, and at the same x, y location within that workspace. Further, if the same item is visible in those two windows, and the user at one window moves that item, changes its color, or otherwise update its knowledge, that change is also visible in the other window.

In the next figure, two telewindows connected to the same G2 are also displaying the same workspace. When an operation in the first window moves the G2 list item upon the workspace, that item also appears to move in the second window. Thus, depending on how your KB is organized, two users working separately at two windows connected to the same G2 process, can interact independently with the same current KB.



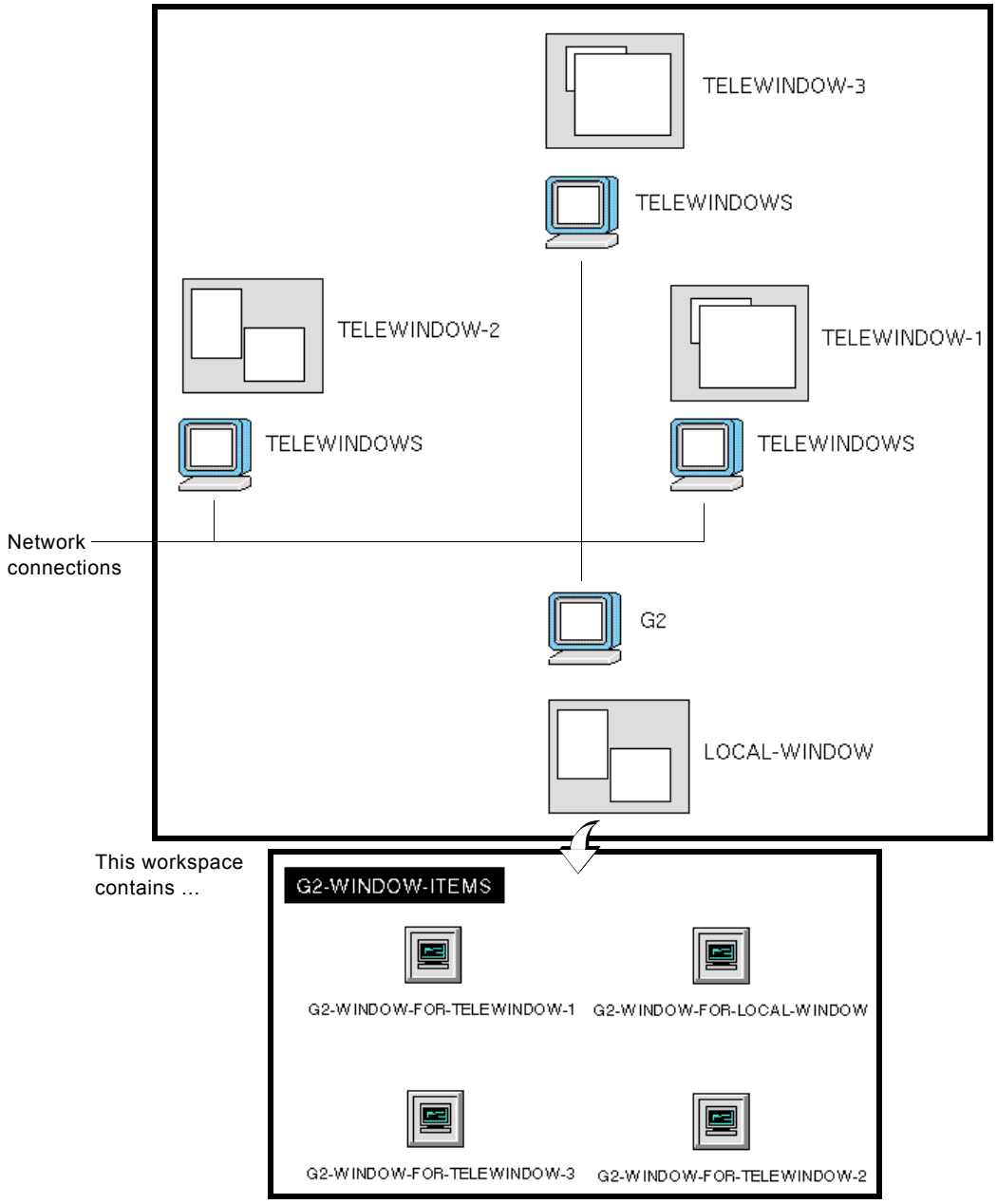
One way to differentiate the operations that two simultaneously connected users can perform within the same current KB, is to use configurations. See Configurations for more information.

The next figure illustrates the relationship between:

- A G2 process.
- Its local window.
- Four g2-window items upon a workspace in the G2's current KB.
- Three Telewindows processes and their respective windows.

The top part of the diagram shows a schematic view of four workstations, as follows:

- One workstation is running G2, and three other workstations are running Telewindows.
- The G2 has its own local window, which displays its own view of the current KB.
- Because each Telewindows process is connected to the G2, each has a window that can simultaneously display a distinct view of the current KB.



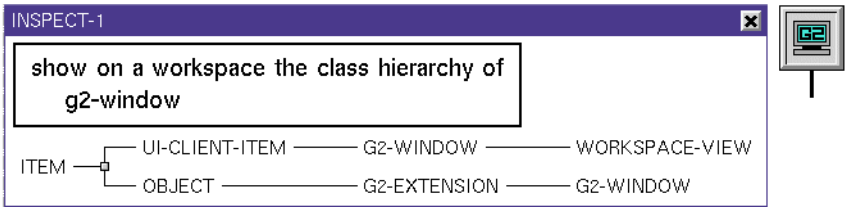
In the top part of the diagram, note that the display in the windows telewindow-1 and telewindow-3 are the same, and the display in telewindow-2 and in the G2 process's local window are the same. This reflects the fact that a G2 can display a different set of the current KB's workspaces in each window that the G2 is serving. Further, for each window that a G2 serves, G2 can display a given workspace at a different location within the window and at a different scale.

In the bottom part of the diagram, the G2's local window shows a KB workspace named g2-window-items, which contains four g2-window items. One of these

g2-windows is associated with the G2's own local window, and each of the other three is associated with a window produced by a Telewindows process that is connected to the G2.

The G2-Window Class

A g2-window item is an instance of the g2-window class. Its direct-superior classes are g2-extension and ui-client-item. This Inspect workspace shows the class hierarchy of the g2-window class and its icon:



Notice that, by default, the icon of a g2-window item displays a stub for a connection of class network-wire, which is a system-defined subclass of connection.

Your KB can draw connections between a g2-window item and other items. For instance, such a connection can indicate visually the portion of the current KB that a particular G2 user or Telewindows user is working with.

For information on the ui-client-item class, see Interfacing with Java Applications.

Attributes of the G2-Window Class

The next table summarizes the class-specific attributes of the g2-window class:

Attribute	Description
g2-user-name	Identifier under which an authorized user logs into a secure G2.
Allowable values:	none, or any other series of characters (only alphanumeric characters are recommended)
Default value:	none
Notes:	After establishing that a G2 user or Telewindows user is authorized, a secure G2 sets this attribute to the value specified in the User Name field of the login dialog.

Attribute	Description
g2-connection-status	Whether this g2-window item is associated with the window of either this G2 or a connected Telewindows process. (Read-only)
Allowable values:	connected connection-closed
Default value:	Determined by whether this g2-window item was created automatically by G2 as the resulting of starting G2 or of receiving a connection from a Telewindows process, or whether this g2-window already exists in the current KB independently of any connection to visible windows
Notes:	See Determining When G2 Associates a G2-Window with a Window.
g2-routing-information	This attribute is reserved for future use.
Allowable values:	Not applicable
Default value:	Not applicable
g2-user-mode	User mode currently in effect for this g2-window item.
Allowable values:	administrator or any application-defined user mode
Default value:	administrator, or value of initial-g2-user-mode-for-this-kb attribute in the KB Configuration system table
Notes:	See Determining the User Mode.
g2-window-style	Allows you to specify the window style you prefer for your interaction with G2.
Allowable values:	default | standard | standard-large | g2-5.x
Default value:	default
Notes:	See G2 Window Styles.

Attribute	Description
g2-window-specific-language	Name of a language translation item that determines, for the window associated with this g2-window item, the language of the text that G2 presents in system-defined menu choices, Text Editor buttons, and so on.
Allowable values:	english japanese korean Name of any language-translation item in the current KB.
Default value:	english
Notes:	The value of this attribute overrides, for this g2-window, the setting for the entire KB found in the current-language attribute of the current KB's Language Parameters system table. See Supporting a Window-Specific Language.
g2-window-management-type	Whether this g2-window item is the client of a G2 process or a Telewindows process. (Read-Only)
Allowable values:	local remote
Default value:	Determined by the event that triggered the creation of this g2-window item: local if created automatically after the G2 is launched, or remote if associated with a successful connection between the G2 and a Telewindows.
Notes:	See Determining Whether the Connection is Local or Remote.
g2-window-x	The x location of the window associated with this g2-window item. (Read-only)
Allowable values:	integer
Default value:	0
Notes:	The value is always 0.

Attribute	Description
g2-window-y	The y location of the window associated with this g2-window item. (Read-only)
Allowable values:	integer
Default value:	0
Notes:	The value is always 0.
g2-window-width	Width in pixels of the window associated with this g2-window item. (Read-only)
Allowable values:	0 to the maximum width in pixels of the user's workstation screen.
Default value:	90% of the maximum width in pixels of the user's workstation screen
Notes:	See Identifying the Dimensions of the G2 Window.
g2-window-height	Height in pixels of the window associated with this g2-window item. (Read-only)
Allowable values:	0 to the maximum height in pixels of the user's workstation screen
Default value:	90% of the maximum height in pixels of the user's workstation screen
Notes:	See Identifying the Dimensions of the G2 Window.
g2-window-x-resolution	Horizontal resolution in pixels per inch of the window associated with this g2-window item. (Read-only)
Allowable values:	50 to 200
Default value:	75
Notes:	See Identifying the Resolution of the G2 Window.

Attribute	Description
g2-window-y-resolution	Vertical resolution in pixels per inch of the window associated with this g2-window item. (Read-only)
Allowable values:	50 to 200
Default value:	75
Notes:	See Identifying the Resolution of the G2 Window.
g2-window-remote-host-name	A string containing the network ID of the workstation from which the connected Telewindows process (associated with this g2-window item) was launched. (Read-only)
Allowable values:	Determined by the range of network IDs permitted by your network.
Default value:	Determined by the event that triggered the creation of this g2-window item: an actual workstation's network ID if associated with a successful connection between the G2 process and a Telewindows process, otherwise none.
Notes:	See Determining the Remote Host Name.
g2-window-user-name-in-operating-system	A string containing the operating-system login ID under which the connected Telewindows process (associated with this g2-window item) was launched. (Read-only)
Allowable values:	Determined by the range of login IDs permitted by your operating system.
Default value:	Determined by the event that triggered the creation of this g2-window item: an actual user's login ID if associated with a successful connection between the G2 process and a Telewindows process, otherwise none.
Notes:	See Determining the Login Name at the Operating System. On the HP UX 11 platform only, changing users (su) does not update this attribute. Its value remains the original user.

Attribute	Description
g2-window-time-of-last-connection	Date and time when this G2 associated this g2-window item with either this local G2 window, or with this telewindow. (Read-only)
Allowable values:	Any time-stamp supported by the operating system under which the associated G2 (for a local G2 window) or associated Telewindows (for a telewindow) runs.
Default value:	Not applicable
Notes:	See Determining the Time of Connection.
g2-window-initial-window-configuration-string	A text value meaningful to your application; it is settable only by the g2-reroute-window system procedure for a reroutable Telewindows connection.
Allowable values:	Any G2 text string.
Default value:	none
Notes:	See Setting up Access to Telewindows.
g2-window-reroute-problem-report	Message returned from an unsuccessful rerouting of a telewindow. (Read-only)
Allowable values:	Not applicable
Default value:	Not applicable
g2-window-operating-system-type	A symbol designating the type of operating system under which the G2 window is running. (Read-only)
Allowable values:	Any symbol.
Default value:	No default value.
Notes:	See Determining the Operating System Type.

Attribute	Description
show-operator-logbook-in-this-window?	Whether the operator logbook is or is not displayed in the g2-window. When the value is yes, the logbook is displayed as specified in the Logbook Parameters system table.
Allowable values:	yes, no
Default value:	yes
Notes:	See Hiding and Showing Logbook Pages.

Attribute	Description
g2-window-user-is-valid	A truth value that indicates whether the user is authorized. (Read-only)
Allowable values:	true, false
Default value	false
Notes:	See Licensing and Authorization
g2-window-mode-is-valid	A truth value that indicates whether the user mode is valid.(Read-only)
Allowable values:	true, false
Default value	false
Notes:	See Licensing and Authorization

Hidden Attributes

The g2-window class defines the following hidden attributes:

Attribute	Description
selected-window-handle	A handle to the selected MDI child view in the native window.
Allowable values:	integer
Default value:	0
Notes:	See Window Handles and Views in User Interface Operations in the *G2 System Procedures Reference Manual*.

Attribute	Description
window-handles	A sequence of handles to all MDI child views in the native window.
Allowable values:	sequence
Default value:	sequence()
Notes:	See Window Handles and Views in User Interface Operations in the *G2 System Procedures Reference Manual*.
mouse-cursor	A symbol that describes the icon used for the mouse cursor.
Allowable values:	default, arrow, cross, hand, help, i-beam, circle-slash, size-all, size-ne-sw, size-ns, size-nw-se, size-we, up-arrow, wait
Default value:	default
Notes:	See Controlling the Mouse Cursor.
g2-window-client-version	A structure that provides the following information about the Telewindows client version: structure (program: *symbol*, {G2, TW, or TWNG} major-version-number: *integer*, minor-version-number: *integer*, revision: *integer*, build-identification-string: *text*) If the g2-window is not associated with any client, the value is an empty structure. For example: structure (PROGRAM: the symbol TWNG, MAJOR-VERSION-NUMBER: 8, MINOR-VERSION-NUMBER: 2, REVISION: 1, BUILD-IDENTIFICATION-STRING: "IC22")

Working with G2-Windows

A g2-window item has attributes that report various information about the window with which it is associated, including that window's connection status. Your KB can use this information to capture information about when your application is in use, from what locations, and by whom.

Accessing the G2-Window Item Associated with Your Interaction with G2

To access the g2-window associated with your Telewindows connection or local G2 window:

➔ Choose Main Menu > System Tables > This Window.

The table of the g2-window item that is associated with your interaction with G2 is displayed.

Overriding the Default Window Style

The default window style for the G2 process is determined by the g2-window-style attribute of the Server Parameters system table. You can specify a different window style for your interaction with the G2 process by editing the g2-window-style attribute of your g2-window item.

You edit the g2-window-style attribute to one of these four values: default, standard, standard-large, or g2-5.x. The g2-window-style of the Server Parameters system table determines your window-style when you specify default.

You can also use the g2-window-style field of the login dialog for specifying your window-style preference.

See G2 Window Styles for information on window styles.

Determining When G2 Associates a G2-Window with a Window

The g2-connection-status attribute indicates whether the g2-window is associated with any window, local or remote. This attribute is updated only by G2.

After a G2 creates a new g2-window item, G2 sets the value of its g2-connection-status attribute to connected. Your KB can detect when this value is set, and thus detect when a new g2-window item is created. For instance, this whenever rule detects when G2 creates a new g2-window item by detecting when the g2-window-connection-status attribute of any g2-window receives a value:

```
whenever the g2-connection-status of any g2-window G
  receives a value
```

```
then inform the operator that
  "User [the g2-user-name of G] has logged into the application."
```

Determining Whether the Connection is Local or Remote

If the g2-connection-status attribute of a g2-window item has the value connected, then the g2-window-management-type attribute indicates whether the associated window is local or remote. G2 assigns a value to this attribute only after it has associated a window with the new g2-window item.

When G2 creates a new g2-window item, the value of its g2-window-management-type attribute is none. Until the g2-window-management-type attribute has a value, any references to it will fail. Thus, before you refer in an expression to the value of g2-window item's g2-window-management-type attribute, first check whether its value exists.

For example, the following procedure statement performs processing based on whether the g2-window-management-type of a G2 window (in this case, passed as the window argument to this procedure) has the value local:

```
if the g2-window-management-type of window exists
  and the g2-window-management-type of window is local
    then ...
```

Determining the G2 User Name for a G2-Window

For the g2-window item associated with a G2 process's local window, a G2 that is not secure initializes the g2-user-name attribute based on the login name of the person who launched the G2. A secure G2 initializes the g2-user-name attribute based on the login name under which you log into G2.

For the g2-window item associated with a telewindow, a G2 that is not secure initializes the g2-user-name attribute based on the operating system login ID of the person who launched the Telewindows process. A secure G2 initializes the g2-user-name attribute based on the login name under which you log into G2.

G2 also uses the value of this attribute when updating the authors attribute of items whose knowledge is changed. If the g2-user-name attribute displays the value none, G2 uses the operating system login ID of the person who launched either the G2 or Telewindows process.

Note After the g2-window is associated with a local or remote window, the user working at that window can also change this attribute interactively at any time by editing the User Name field in the login dialog. See Using the Login Dialog.

A Telewindows user logging into a secure G2 must supply a user name in the login dialog and can optionally supply a user mode. If the supplied combination

of user name and user mode is authorized and after completing the connection with the Telewindows process, G2 creates a new g2-window item in the KB and assigns that user name and user mode to the new item's g2-user-name and g2-user-mode attributes. G2 also sets the g2-window-user-is-valid and g2-window-mode-is-valid attributes to true when the user name and mode are valid.

Determining the Login Name at the Operating System

For a g2-window item associated with a connected telewindow, the g2-window-user-name-in-operating-system attribute shows the login ID (or account name) under which the Telewindows process was launched.

Determining the User Mode

If the initial-g2-user-mode-for-this-kb attribute in the KB Configuration system table has the value none, then when a new g2-window is created, G2 initializes the g2-window's g2-user-mode attribute to the value administrator.

If the initial-g2-user-mode-for-this-kb attribute in the KB Configuration system table has a value other than none, then when G2 creates a new g2-window item, G2 also initializes the g2-user-mode attribute to that value.

Tip After G2 associates a g2-window item with a local or remote window, the user working at that window can use the login dialog to change the value of the g2-user-mode attribute of that g2-window. See Using the Login Dialog.

A Telewindows user logging into a secure G2 must supply a user name in the login dialog and can optionally supply a user mode. If the supplied combination of user name and user mode is authorized, then after completing the connection with the Telewindows process, G2 creates a new g2-window item and assigns that user name and user mode into the g2-user-name and g2-user-mode attributes of the new g2-window item. G2 also sets the g2-window-user-is-valid and g2-window-mode-is-valid attributes to true when the user name and mode are valid.

Determining the Remote Host Name

G2 initializes g2-window-remote-host-name attribute of a new g2-window item to the value none.

For a g2-window associated with a remote window, G2 automatically assigns the g2-window-remote-host-name attribute to the host name (established by the network administrator) of the computer from which a Telewindows user has connected to the G2.

Determining the Time of Connection

For a g2-window associated with a local window, the g2-window-time-of-last-connection attribute shows the date and time at which this user launched this G2.

For a g2-window associated with a remote window, the g2-window-time-of-last-connection attribute shows the date and time when the Telewindows user connected to this G2.

Determining the Operating System Type

The value of the g2-window-operating-system-type attribute is a symbol that indicates the type of the operating system on which the G2 window is running. The allowable values vary as different operating systems and versions become or cease to be supported by their manufacturers or G2.

Controlling the Mouse Cursor

The g2-window class has a new hidden attribute named mouse-cursor, whose value is a symbol with these possible values:

Symbol	Icon
default	
arrow	
cross	
hand	
help	
i-beam	
circle-slash	
size-all	

Symbol	Icon
size-ne-sw	
size-ns	
size-nw-se	
size-we	
up-arrow	
wait	

Note The icons have a somewhat different appearance on Windows and UNIX platforms.

Expressions that Refer to G2-Window Items

Because a g2-window is an item, your KB can use item reference expressions to refer to it and can use attribute reference expression to refer to its attributes.

The this window expression refers to the g2-window that is associated with the window in which a user-initiated event takes place. You can specify this expression only in the action attribute of an action button or user menu choice.

The power of the this window expression is to associate a g2-window, and therefore the set of knowledge it contains, with the initiation of a thread of processing. Thus, your application can associate a particular user-initiated event with a login account (in the g2-window's g2-window-user-name-in-operating-system attribute), computer identification (in the g2-remote-host-name attribute), current language (in the g2-window-specific-language attribute), and so on.

Specifying the Appearance of the G2 Window

Each g2-window item has read-only attributes that report the associated window's height and width in pixels, resolution in pixels per inch, and magnification.

G2 displays a window, using these defaults:

- *Height in pixels* of 90% of the screen's height in pixels, and *width in pixels* of 90% of the screen's width in pixels.
- *Resolution* of 75 pixels per inch.
- *Magnification* of one G2 workspace unit per pixel.

You can initialize these attributes by specifying command-line options when you launch a new G2 process or Telewindows process.

Specifying the Resolution and Magnification

The `-magnification` command-line option specifies the default magnification for KB workspaces at full scale. The optional `-resolution` command-line option informs a G2 process about the resolution (in pixels per inch) of the monitor on which the window appears. Together, these options determine the absolute size at which G2 displays a window on a given display device for a given platform.

By combining the settings of these two options properly, you can launch G2 processes on different computers having display devices of different resolutions and display the same KB at the same (or very nearly the same) absolute size. Alternatively, by specifying other settings in these options, you can launch a G2 process that displays a KB at the highest resolution allowed on a particular display device.

Tip For a description of the `-resolution` and `-magnification` command-line options resolution, and magnification.

For example, if you use this command to launch a G2 process:

```
g2 -resolution 75 -magnification 1.0
```

it is equivalent to this command line:

```
g2 -resolution 100 -magnification 0.75
```

Tip For best results, consider the dot pitch (that is, the ratio of width to height) of the pixels produced on the display devices on your G2 application's delivery platform.

Identifying the Dimensions of the G2 Window

The g2-window-height and g2-window-width attributes report the dimensions, in workspace units, of the G2 window.

These attributes are read-only, but you can refer to them in expressions. For example, this show action displays a KB workspace at a scale that has a particular ratio of height to width:

```
{ Scale a kb-workspace on a g2-window of arbitrary size
  to maintain the same relative size as if displayed on a window
  of 1036 by 810 workspace units. }

show help-workspace scaled by its current scale times
  min ( ( the g2-window-width of this window / 1036 ) ,
        ( the g2-window-height of this window / 810 ) )
```

You can initialize the g2-window-height and g2-window-width attributes of a new G2 window associated with a local window by using the `-height` and `-width` command-line options. For more information about these options, see height and width.

Otherwise, G2 automatically updates these attributes whenever you use the host window manager to resize the local or remote window associated with the G2.

Tip When you launch a G2 process, you can specify the `-fullscreen` command-line option to display the new G2's local window at full-screen size.

Identifying the Resolution of the G2 Window

The g2-window-x-resolution and g2-window-y-resolution attributes report the resolution (in pixels per inch) at which G2 displays the window associated with this g2-window item.

Note These attributes are read-only. The associated window's resolution does not change during the window's existence.

You can initialize the g2-window-x-resolution and g2-window-y-resolution attributes of a new g2-window item associated with a local window by using the `-resolution` command-line option, or by using the `-x-resolution` and `-y-resolution` pair of command-line options.

See also Appendix A, Launching a G2 Process for more information about these options.

Rerouting a Telewindow

A g2-window item has attributes that support switching or rerouting a telewindow, as described in Rerouting Telewindows Connections.

A G2 process *reroutes* a telewindow by passing its connection to another G2. A G2 reroutes a telewindow by executing the g2-reroute-window system

procedure. For more information, see the description of g2-reroute-window in the *G2 System Procedures Reference Manual*.

Tip The KB file `twtour.kb`, a sample KB shipped with your G2 product, demonstrates the features that a G2 application should support when rerouting a telewindow. See the *Telewindows User's Guide* for information about `twtour.kb`.

Setting up Access to Telewindows

The g2-window-initial-window-configuration-string attribute contains a text value that the KB, running in a G2 that receives a reroutable Telewindows connection, uses to set up access for the user to that KB. This attribute is only used by the g2-reroute-window system procedure.

For instance, in a G2 application designed to support access by users via reroutable telewindows, the KB running on one G2 can hand off a user's processing to another KB running on another G2. The initiating KB can log the user (via Telewindows) into another G2 and pass to its KB a g2-window-initial-window-configuration-string value that represents the state of that user's activity within the application.

For more information, see the description of the g2-reroute-window system procedure in the *G2 System Procedures Reference Manual*.

Reporting Errors

The g2-window-reroute-problem-report attribute is a read-only attribute that presents to a G2 an error message that returns from an unsuccessful rerouting of a telewindow to another G2. This capability of G2 is described under Rerouting Telewindows Connections.

Supporting a Window-Specific Language

Language translation items contain text that replaces the system-defined text that appears in G2 menu choices, Text Editor buttons, and so on. A KB that contains more than one language translation item can display G2's own text, as well as user-defined text, in more than one natural language.

The setting of the current-language attribute in the Language Parameters system table determines which of the current KB's language translation items governs the display of G2's system-defined text. For more information, see Using Language Translations for Localization.

For a current KB that contains more than one language translation item, the KB can programmatically associate a distinct language translation with each window that is associated with or connected this G2. The g2-window-specific-language

attribute of a g2-window item identifies the language translation that G2 is using to display system-defined text in that g2-window's associated window.

When you launch G2, use the `-language` command-line option to set the value of the g2-window-specific-language attribute of the g2-window item associated with G2's own local window.

Also, when a user launches a Telewindows process, that user can also specify a `-language` command-line option:

- If the user is connecting to a secure G2, specifying this option sets the value of the G2 Window Specific Language field shown in the login dialog. If the user successfully logs in, this field's setting determines the value of the g2-window-specific-language attribute of the g2-window item associated with the new Telewindows process's own window.
- If the user is connecting to a G2 that is not secure, specifying this option determines only the value of the g2-window-specific-language attribute of the g2-window item associated with the new Telewindows process's own window.

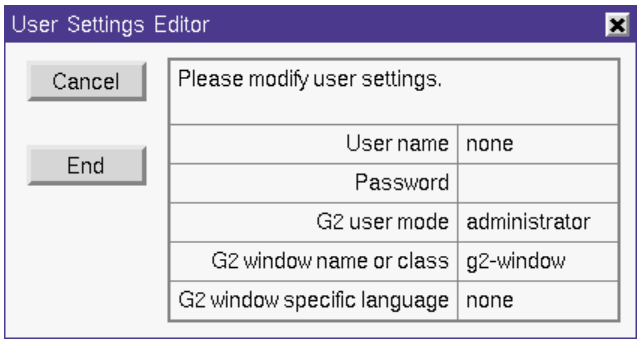
The g2-window-specific-language attribute of a new g2-window item interacts with the current-language attribute of the Language Parameters system table as follows:

- When the value of the g2-window-specific-language attribute is none, G2's system-defined menu choices appear in the language named in the current-language attribute of the Language Parameters system table.
- When the value of the g2-window-specific-language attribute is other than none, the language named in the g2-window-specific-language attribute overrides the setting of the current-language attribute of the Language Parameters system table.

This feature is especially useful for a KB that users access via telewindows. Based on the Telewindows user's login ID, the KB can assign the name of a particular language translation to the g2-window-specific-language attribute of the g2-window item associated with that Telewindows process's own window. As a result, multiple users can simultaneously interact with the same KB, but view the text portion of the KB's context in different languages.

Using the Login Dialog

G2's login dialog allows a secure G2 to gather the information required to authorize each user who attempts to connect to a running G2 process:



After a user has logged into a secure G2, G2 updates attributes in the g2-window item that G2 associated with the window that the user sees:

- G2 assigns the specified user name into the g2-user-name attribute.
- G2 assigns the specified G2 user mode into the g2-user-mode attribute.
- G2 assigns the specified window-specific language into the g2-window-specific-language attribute.

For a G2 installation that does not rely on a secure authorization file, a G2 user or Telewindows user can easily display the login dialog and use it to change attributes in the g2-window item associated with the window at which he or she is working.

By default, the Login Dialog displays as a native pane in Telewindows. For more information, see Displaying the Native G2 Login and Change Mode Dialogs.

Displaying the Login Dialog

A secure G2 displays the login dialog in G2's local window (if present) each time it is launched, and in a remote window each time a Telewindows user attempts a connection. For a description of how a secure G2 relies on the login dialog, see Accepting a Connection from a Telewindows Process.

In a G2 that is not secure, the user must explicitly display the login dialog by choosing Main Menu > Change Mode or by entering CTRL + y. Doing this is one way for a G2 or Telewindows user to change interactively the values of the g2-user-name, g2-user-mode, and g2-window-specific-language attributes for the g2-window item that is associated with the window at which he or she is working.

Determining Default Values in the Login Dialog

Your KB can determine the default values of some fields in the login dialog. For example, if the initial-g2-user-mode-for-this-kb attribute in the KB Configuration system table has a value other than none, G2 initializes the G2 User Mode field in the login dialog to that value. Also, the current-language attribute of the Language Parameters system table determines the default value of the G2 Window Specific Language field in the login dialog.

Notice that the fields in the login dialog correspond to attributes of the g2-window class, as follows:

Field in Login Dialog	Attribute of G2-Window Class
User name	g2-user-name
Password	Not applicable
G2 user mode	g2-user-mode
G2 window name or class	Not applicable
G2 window specific language	g2-window-specific-language

Logging Login Activities

This feature allows you to run a user-defined login handler whenever a user logs into your secure G2. You must register this procedure with G2, using the system procedure g2-system-register-login-handler. See the *G2 System Procedures Reference Manual* for a description of this system procedure. You can then use your login handler to perform whatever operations you wish on successful or failed logins.

Changing a user's password is not considered a login event and will not call your login handler.

Writing the Login Handlers

The login handler must accept a structure as an argument. The structure, which is returned by the system login function, has the following attributes:

Attribute	Value
success	true if the login succeeded, false otherwise.
system	The symbol tw for Telewindows.

Attribute	Value
status	A symbol describing the event.
user-name	A symbol.
user-mode	A symbol.
network-info	The `icp-connection-name` string for connections over the network and false otherwise.

The `icp-connection-name` string provides information about the protocol of the connection and the hostname of the machine attempting to connect.

The login handler may use this information in any way necessary. The following example shows a login handler that simply prints the information in the structure to the message board:

```
default-login-handler(login-information: structure)
msg: text;
begin
    if (the success of login-information)
        then msg = "succeeded"
        else msg = "didn't happen [the status of login-information];
    post "Login [msg] in system [the system of login-information]
        for user: [the user-name of login-information]
        in mode: [the user-mode of login information]
        from [the network-info of login-information]"
end
```

Registering the Login Handler

Before it can be called, the login handler must be registered with G2. To do this, use the system procedure g2-system-register-login-handler. Please refer to the *G2 System Procedures Reference Manual* for information about this procedure.

Associating an Existing G2-Window with a Telewindow

G2 supports the practice of associating a g2-window item that is created and maintained by your application, with the window that a Telewindows process opens after connecting to a G2 process. After the Telewindows process connects to G2, the Telewindows user can specify the name of an existing g2-window item, or the name of a user-defined subclass of the g2-window class, in the login dialog's G2 Window Name or Class field.

If the Telewindows user specifies a name that G2 finds is also the name of an existing g2-window item (or of an existing item whose class is a subclass of the g2-window class), then G2 initializes that item's attributes and associates it with the Telewindows process's own window.

On the other hand, if G2 finds that the name is also the name of a subclass of the g2-window class, then G2 automatically creates a new item of that class, initializes its attributes, and associates it with the Telewindows process's own window.

Note that it might limit the robustness of your application to require users to supply the name of a g2-window item (or of a subclass of the g2-window class) in the login dialog, after also supplying a user name, user mode, and optionally, a window-specific language.

Part III

Knowledge Representation

Chapter 9: Values and Types

Describes the role of values and types in a knowledge base.

Chapter 10: G2 Items

Presents the characteristics that are common to all G2 items.

Chapter 11: Attributes and Tables

Shows you how to use item attributes and the attribute tables that display them.

Chapter 12: Attribute Access Facility

Presents the capabilities of the attribute access facility.

Chapter 13: Classes and Class Hierarchy

Describes the principles, structure, and use of the G2 class hierarchy.

Chapter 14: Definitions

Describes class definitions and shows you how to use them.

Chapter 15: Variables and Parameters

Describes variables and parameters and how to use them within a KB.

Chapter 16: Lists and Arrays

Describes how to use lists and arrays.

Chapter 17: Hash Tables and Priority Queues

Describes how to use hash tables and priority queues.

Chapter 18: Connections

Describes connections, connection posts, and junction blocks.

Chapter 19: Relations

Describes how to associate items in a non-graphical way.

9

Values and Types

Describes the role of values and types in a knowledge base.

Introduction

A **value** is a piece of knowledge of a particular G2 type. Values consist of data structures that are generated as the result of expression evaluations and are associated with item attributes.

Values have a type, which can be:

- integer
- long
- float
- text

- truth-value
- symbol
- sequence
- structure

As the current knowledge base runs, it obtains values from entities such as the knowledge stored in user-defined and system-defined attributes and the local names and other values within the text attributes of executable items, such as procedures and rules.

Using Values Stored in Items

Your KB's activities work primarily with values stored in item attributes. Attributes can be user-defined to capture the values of user-defined items, or they can be system-defined and specify such item knowledge as the location of the item upon its workspace, its relations and connections, its current attribute displays, the value of variables and parameters, and so on. In some cases, the KB uses the value in one attribute to assign the value of another attribute. In other cases, the KB obtains values to write them to external files or to pass them to external processes, such as G2 Gateway bridge applications.

For description of the kinds of information that are part of an item's knowledge, see Understanding the Knowledge Contained in Items.

Using Attribute Values

An item stores values in its attributes. An attribute might also have no value, in which case G2 displays the symbol none as its value in an attribute table.

You can use the conclude and change actions to update the values of all user-defined attributes and most system-defined attributes as follows:

Use this action...	For system-defined attributes that are...
conclude that the *x* of *y* = *value*	Value-writable
change the text of the *x* of *y* to "*text-value*"	Text-writable

Each chapter of this manual that describes a system-defined class includes a section describing the characteristics of each system-defined attribute. Check there to find which attributes you can edit. Refer to the *G2 Class Reference Manual* for information about whether an attribute is value- or text-writable.

Using Text Attribute Values of Items

Some items include a text value, which is distinct from other attributes. This text attribute appears in the attribute table of relevant items without an attribute name called text, but is referred to programmatically with the expression the text of *y*, where *y* is any item of these classes:

- Rules
- Procedures
- Methods
- Message
- Free text
- Borderless free text
- Word inserters
- Character inserters
- Character sequence inserter

For information on rules, procedures, and methods, see Rules, Inferencing, and Chaining, Procedures, and Methods.

For more information about messages and each free text, borderless free text, word inserter, character inserter, or character sequence inserters see Messages and Text Items.

The text attribute of an item always stores a value of type text. For more information about text values, see Using the Text Type.

Using Values Given by Variables and Parameters

Each variable and parameter has a last-recorded-value attribute that is handled differently from its other attributes. See Variables and Parameters for more information.

Checking for the Existence of an Attribute Value

The attributes of items can hold values, a subobject, or nothing which appears as none.

You can determine whether an item attribute has a value by using the following expressions:

- exists
- has a value
- has a current value

For more information about these expressions, see Expressions.

Using Local Names for Values

Your KB can also declare and manipulate values that are not part of any item's knowledge and that exist only when the current KB is running. For instance, you can use local names to represent values used only within one rule or procedure. See Using Local Names in Expressions.

Expiration of Variable Values

The value of each instance of a variable has an expiration time, which is the time interval after which G2 must perform data seeking to obtain a valid value. The expiration time can be never, indicating that the value is valid indefinitely.

The expiration time of a variable is determined by its validity-interval attribute. If a variable value expires, and is then required by an expression referring to that value, G2 attempts to obtain a new value to replace the expired one.

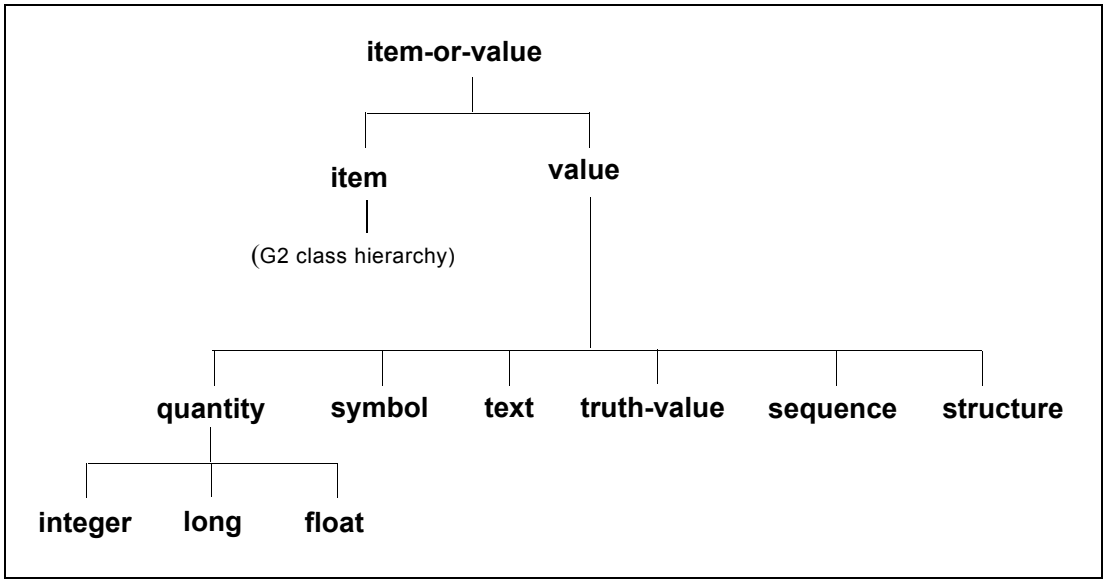
The expiration time of variable values also affects the expiration time of the expressions in which those values participate. G2 must compare the expiration time of a value used in a computation, such as in a rule or procedure, with the time required to complete the execution of a rule or procedure that uses that value. If a value expires before G2 can finish performing all portions of a rule or procedure that refers to that value, G2 must perform data seeking to obtain a new current value that replaces the expired value. How G2 data seeks for the values of variables is explained in Obtaining Values for Variables.

Distinguishing Value Types

All G2 values have a **type**, which determines the valid operations in which the value can participate. The G2 types, which are categorized into general, specific, and composite types, are:

General Types	Specific Types	Composite Types
item-or-value	integer	sequence
value	float	structure
quantity	symbol	
	text	
	truth-value	
	long	

G2 organizes these types into a type hierarchy, with the item-or-value type as the root type. The following figure summarizes the relationships among the item-or-value type and its subtypes:



This figure also shows that the item-or-value type is the parent of the item class. Conceptually, G2 items are values whose type is class item. The item class, and the rest of the system-defined classes in G2's class hierarchy, are described in Classes and Class Hierarchy.

Every attribute has a particular type, and is described in the *G2 Class Reference Manual*.

Declaring a type for a user-defined attribute in a class definition is optional. However, we recommend that you always use the most-specific type possible. The declared type of an attribute restricts the values you can store in that attribute and determines the valid operations for the attribute value.

Complex Types

As shown by some of the type specifications in the *G2 Class Reference Manual*, G2 internally uses arbitrarily complex types formed by using Boolean expressions to combine the types described in this chapter. You cannot assign a complex type to user-defined attributes, which can use only the types described in this chapter.

Declaring Types

You can declare that a piece of knowledge is of any type in these contexts:

- A class definition, for user-defined, class-specific attributes.

- A procedure or method, for return values or local names.

These items contain values of the given types:

This type...	Is used in these items...
item-or-value	g2-list and g2-array items, whose elements contain values of type item-or-value, excluding sequences and structures.
value	value-list and value-array items, whose elements contain values of type value, excluding sequences and structures.
quantity	quantitative-variable, quantitative-parameter, quantity-list, and quantity-array items.
integer	integer-variable, integer-parameter, integer-list, and integer-array items.
long	long-variable, long-parameter, long-list, and long-array items.
float	float-variable, float-parameter, float-list, and float-array items.
symbol	symbolic-variable, symbolic-parameter, symbol-list, and symbol-array items.
text	text-variable, text-parameter, text-list, and text-array items.
truth-value	logical-variable, logical-parameter, truth-value-list, and truth-value-array items.

Working with General Types

The G2 general types are:

- item-or-value
- value
- quantity

For those familiar with C, all values in G2 are implemented as pointers to data structures containing explicit type tags. All attributes and local names contain these pointers, so that G2 can always determine a specific type from the value itself.

Values declared as a general type specify that the value held in the local name or attribute will be one of the specific types that is a subtype of the general type. For example, if a local name is specified as a quantity, its value will be either an integer or a float, which are the subtypes of quantity.

Using the Item-or-Value Type

A value of type item-or-value represents a piece of information into which your KB can assign either an item or a value of a general, specific, or composite type.

Declaring a value of type item-or-value can add flexibility to some kinds of KB processing. Given a value of type item-or-value, your KB must determine the value's class or specific type before using it in a class-specific or type-specific expression.

Using the Value Type

A value of type value represents a piece of information that your KB can interpret as a number (that is, a quantity value, a float value, an integer value, or an long value), a symbol value, a text value, a truth-value value, a sequence, or a structure.

Using the Quantity Type

A value of type quantity represents a number that your KB can interpret as either type integer, long or float, depending on the processing context. This flexibility can be an advantage for some kinds of KB processing.

You can also assign a quantity value into a piece of knowledge declared with type quantity. After performing this assignment, however, for your KB to use the value that the assigned quantity now references, your KB must first use an expression to determine the specific type of the referenced value, integer or float. The KB can then use the referenced value in an expression or action.

Working with Specific Types

The G2 specific types are integer, long, float, symbol, text, and truth-value. Each value that your KB directly manipulates has a specific type. Your KB cannot create user-defined specific types.

For example, when you declare a local name in a procedure with the type integer, an action or procedure statement can assign into that local name only a value that meets the requirements for integer numbers.

Using the Integer Type

A value of type integer represents an integral number.

In G2 Standard (32-bit), G2 integer values are signed with 30-bit precision. A G2 integer value can range from -536870912 to 536870911, that is, from -2^{29} to $(2^{29} - 1)$.

In G2 Enterprise (64-bit), G2 integer values are signed with 61-bit precision. A G2 integer value can range from -1152921504606846976 to 1152921504606846975, that is, from -2^{60} to $(2^{60} - 1)$.

Note For actions that update the value of an integer value, G2 does not check for integer overflow or underflow because of the performance penalty such checking would impose.

KB saved by G2 Enterprise with G2 integer values which exceeded the value range of G2 Standard, will loose its original value when loading in G2 Standard. To prevent loosing of integer values in this case, it's recommended to use the new long type.

Using the Long Type

A value of type long represents an integral number. G2 long values are signed with full 64-bit precision. A G2 long value can range from -9223372036854775808 to 9223372036854775807, that is, from -2^{63} to $(2^{63} - 1)$.

Note For actions that update the value of an long value, G2 does not check for integer overflow or underflow because of the performance penalty such checking would impose.

The G2 long type is a native implementation which uses the underlying 64-bit arithmetic CPU instructs to do all the 64-bit computations (even in 32-bit G2 Standard). However, it takes more memory spaces (16 bytes per long value) than the integer type (4 bytes in G2 Standard and 8 bytes in G2 Enterprise), and the performance may not be as good as the integer type.

In G2 2011, the G2 long type is undocumented. And it's not a native implementation, instead, using 32-bit arithmetic instructs to simulate 64-bit computing, therefore very slow.

Using the Float Type

A value of type float represents a real number with a floating-point representation. Due to the limitations of the floating-point representation for real numbers, it is possible for a particular float value to represent an *approximation* of a real number. This only occurs for very large and very small numbers.

Float values in G2 are signed, with a 64-bit, double-precision floating-point representation. On most platforms that G2 supports, a float value can range from $\pm 1.79 \times 10^{308}$ to $\pm 2.22 \times 10^{-308}$ with approximately 16 digits of precision.

G2 restores and manipulates float values in conformance with the IEEE's *Standard for Binary Floating-Point Arithmetic* (ANSI/IEEE Standard 754-1985). For G2 to conform to this standard, the computer on which G2 runs must also support that standard.

Working with Exceptional Float Values

Arithmetic operations on float values can result in these **exceptional float values**:

Exceptional Float Value	Display of Value in G2	Causes
Negative infinity	-Inf	Negative overflow; divide a negative value of any numeric type by zero (0 or 0.0)
Positive infinity	+Inf	Positive overflow; divide a positive value of any numeric type by zero (0 or 0.0)
Not a number	NaN	Divide zero (0 or 0.0) by zero

The ANSI/IEEE Standard 754-1985 document specifies how exceptional floating point values participate in arithmetic operations. G2 conforms to this standard on platforms that support it.

For example, in a numeric expression, if an exceptional float value participates in any G2 arithmetic operation in that expression, G2 evaluates that expression as that same exceptional float value. Also, in a truth-value expression, if an exceptional float value participates in a comparison operation in that expression, G2 evaluates that expression as false.

Note In cases where on other platforms G2 would produce an exceptional float value, on the Alpha OSF platform, instead G2 signals the error "A floating point exception has occurred."

Coercing Numeric Values

Some G2 arithmetic and relational operators require G2 to *coerce* an integer value to a float value. This means that G2 automatically creates a temporary copy of a value in a different type, for use in evaluating an expression.

For example, in an expression that compares an integer value to a float value, G2 automatically coerces the integer value to a float value, then compares the two values.

Tip For the details about how G2 coerces numeric values when applying arithmetic and relational operators, see Coercion of Values Returned from Arithmetic Operators.

Using Units of Measure for Numeric Values

You can create your own set of symbols to represent units of measure, such as meters, pounds, and liters, that G2 does not provide. You define these symbols in a units-of-measure-declaration item.

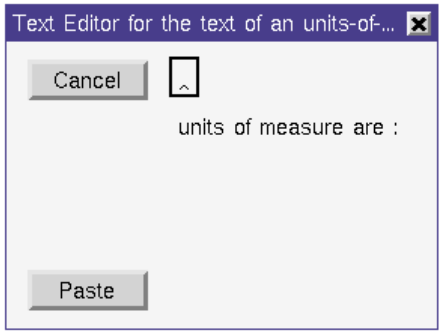
In a numeric value, such as in an attribute that stores a number or in a variable whose value is an integer or float value, you can use the symbols defined in a units of measure declaration. Doing so indicates that the numeric value represents a measurement.

A value's unit of measure symbols only affect how G2 displays that numeric value. When G2 assigns a value that uses a user-defined unit of measure into an attribute, procedure local name, or variable or parameter, that value's unit of measure is also assigned.

To create a unit of measure:

1 Choose KB Workspace > New Definition > units-of-measure-declaration.

G2 automatically invokes the Text Editor for you to declare the unit:



2 Enter the symbol of your choice, with an optional singular version of the term, using this syntax:

```
units of measure are:
    {plural-unit-of-measure-symbol
    [singular (single-unit-of-measure-symbol )]}  [,...]
```

For example, this units-of-measure declaration declares meters and centimeters:

```
units of measure are:
    meters (singular meter), centimeters (singular centimeter)
```

After you declare one or more units of measure, they appear after a numeric expression in the Text Editor prompts for:

- The initially is statement in a class definition, for attributes without a type.
- The value of any instantiated item attribute, after you enter a value.
- The data-type attribute of quantitative, integer, and float variables and parameters.
- The initial-value attribute of quantity, float, and integer variables and parameters.

Using the Symbol Type

A symbol value contains a series of characters, each of which is a member of the Unicode character set. For more information about the Unicode character set, see G2 Character Support.

Use symbol values to represent identifiers: names of items, attributes, classes, and types.

All characters in a symbol value are uppercase unless you quote them using the at sign (@) character. When creating a symbol value, the first character can consist of any Unicode character set.

Working with Characters in a Symbol Value

The Unicode character set supports alphabetic and ideographic characters from most of the world's modern and classical languages. For a discussion of Unicode, see G2 Character Support.

When creating a symbol value, the first character can consist of any Unicode character. All characters in a symbol value are uppercase unless you quote them.

Tip To quote any character, precede it with the at sign (@) character in the Text Editor.

Symbols can include lowercase characters from supported Unicode languages by quoting the characters. You can quote any Unicode character.

Each of these characters requires quoting to be included in a symbol:

```
! " # $  & ( ) * + , / : ; < = > ? @
[  ] ^ '{ | } ~ © ™ ® • ¢ £ ¥ » « ¡ ¿ ƒ
```

These characters also require quoting:

Enter this character...	By...
Tab	Pressing the Tab key one or more times. Each time you press the Tab key in the editor, G2 inserts the number of spaces designated in the number-of-spaces-to-insert-on-a-tab attribute of the Editor Parameters system table.
Space character	Pressing the space bar.
Line separator	Entering the Unicode character x2028.
Paragraph separator	Entering the Unicode character x2029.

Note For information about entering Unicode characters, see Entering Unicode Character Codes.

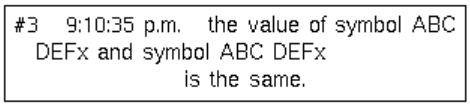
If you begin a symbol value with a period (.) or a number (0 - 9), it must also include at least one alphabetic character or quote one of its numeric characters or any of the Unicode character set symbol, punctuation, or special characters. The hyphen (-), underscore (_), period (.) , and apostrophe (') characters are exceptions, which do not require quoting.

Some examples of valid and invalid symbols are:

Valid Symbols	Invalid Symbols
@!7	.777
'123	123
-._	!!$
12@3	
my-object	
@my-object	

G2 always ignores the case of all unquoted alphabetic characters. For example, in the next procedure, G2 always executes the post action, as shown in the message displayed:

```
gds-compare-symbols()
symbol1, symbol2: symbol;
begin
     symbol1 = the symbol ABC@ def@x;
     symbol2 = the symbol abc@ DEF@x;
     if symbol1 = symbol2 then
          post "the value of symbol [symbol1] and symbol [symbol2]
               is the same."
end
```



When entering special characters in the Text Editor, first quote the character using the at sign (@), and then press Alt + i, followed by the special character you require. In this procedure example, the trademark symbol is available by entering Alt + i t.

Using the Text Type

A text value contains a series of characters, each of which must be a member of the Unicode character set. For more information about the Unicode character set, see G2 Character Support.

Use text values to contain any sequence of characters, including case-sensitive alphabetic characters. The maximum number of characters in a text value is 1000000.

Working with Characters in a Text Value

G2 allows any character from the Unicode character set in a text value. The case of characters is significant. G2 retains, displays, and prints the case of all alphabetic characters.

G2 allows quoted characters in a text value, though quoting is unnecessary for all characters in the Unicode character set, except:

- The at sign (@); enter two at sign characters (@@) to include one at sign (@) character in the text string.
- Double quotes ("); otherwise, this character delimits a literal text value.
- Left bracket ([); otherwise, remaining characters after a left bracket signify a literal value.

Specifying a literal text value is described in Evaluating Expressions. For a description of the concatenation operation on text values, see Using the Concatenation Operator.

The following procedure demonstrate-equal-texts demonstrates these facts about text values:

- G2 retains the case of alphabetic characters in a text value.
- It is redundant to quote a character other than @, ", and [in a text value.

For example:

```
demonstrate-equal-texts()

{ Notice the unnecessarily quoted character in the value of text3. Also notice
    that the text values displayed in the post action retain the case of their
    alphabetic characters. }

text1: text = "ABCabc";
text2: text = "XYZxyz";
text3: text = "[text1]+@+[text2]";
text4: text = "[text1]++[text2];

begin
    { This post action always executes. }
    if text3 = text4 then
        post "The values of text1 [text1] and text4 [text4] are equivalent."
end
```

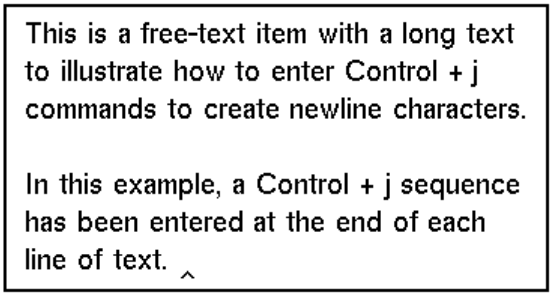
Formatting Text Values

You can include a newline character in a literal text value to format lengthy text. The way to include a newline character depends on whether you are editing the text in a non-scrolling editor, such as for messages and other free text items, or a scrolling editor, such as for procedures and methods.

To enter a newline character in a text value in a non-scrolling editor:

➔ Press Control + j anywhere within the quoted text value that you want a newline to appear.

This example illustrates the use of Control + j newline sequences in a free-text item:



Note Any newline characters that you enter to format text values are stored as Unicode line separator characters. Such newline characters do not, therefore, translate into ASCII newline character values when exporting text from G2.

To enter a newline character in a text value within a scrolling editor:

➔ Press Return anywhere within the quoted text value that you want a newline to appear.

Getting Unicode Character Codes

You can get the Unicode character code of a single character in a text by including a zero-based index in square brackets following the text. This construct returns the equivalent of the text-to-character-codes function but for a single character. For details, see Converting Character Codes to Unicode Text.

To determine the character code of a text character:

➔ *text* [*integer*]

For example, this procedure returns the Unicode character code of a character in a text:

```
get-character-code(txt: text, index: integer)
t: text;
c: integer;
begin
    t = "[txt]";
    c = t[index];
    post "[c]";
end
```

This action returns 116, which represents the character code for the letter "t", the first (0th) character in the text:

```
start get-character-code("text", 0)
```

Using the Truth-Value Type

A value of type truth-value represents a degree of certainty in the truth of a condition, comparison, or assertion. Your KB can use values of type truth-value to implement a reasoning strategy based on the principles of either boolean logic or fuzzy logic.

A value of type truth-value ranges from -1.0 true (completely false) to +1.0 true (completely true).

Tip G2 displays a truth-value of -1.0 true simply as false, and displays a truth-value of +1.0 true simply as true. In this case the displayed values true and false represent truth-values, not symbols.

In a truth-value expression that includes a relational operator, by specifying a fuzzy truth band subexpression, you can produce a fuzzy truth value, whose decimal value is greater than -1.0 true and less than +1.0 true.

For example, the following conclude action assigns a value into a truth-value attribute of an item of a user-defined class, based on the result of evaluating the expression (the volume-in-liters of tank-1 > 100) (+- 25) :

```
conclude that the truth-value-attribute of my-object =
  (the volume-in-liters of tank-1 >100) (+- 25 )
```

In this example, the subexpression (+- 25) signifies a fuzzy truth band. The degree to which the volume-in-liters of tank-1 is greater than, equal to, or less than the value 100, determines the fuzzy truth value that G2 assigns to the Truth-value-attribute of my-object.

Tip For the details about specifying fuzzy truth band expressions, see Producing Fuzzy Truth Values from Relational Operations.

Representing Time Values

G2 offers three formats for representing time: as an integer, as a float, and as a text string. Each of these has advantages and disadvantages, as described in this section. The difference between integer and float time is significant whenever time intervals greater than 17 years are required.

Time as an Integer

G2 time functions, non-subsecond time expressions, and G2's internal scheduler encode time as an integer representing a number of seconds. This technique is convenient and fast, but cannot represent an interval greater than 17 years

because the integer overflows. This restriction can cause problems in several contexts, such as:

- A G2 application runs continuously for more than 17 years.
- A simulation proceeds for more than 17 years of simulated time.
- Schedule projections extend more than 17 years into the future.

The results when integer time overflows are unpredictable. Most applications will never encounter the 17-year limit of integer time. When time values greater than 17 years may occur, use float time, as described in Time as a Float. For more information about integer time, see:

- Time functions: See Time Functions.
- Time expressions:
  - See History Expressions.
  - See Referring to the Current Time.
- Scheduling: Task Scheduling.

Time as a Float

G2 provides system procedures that encode time as a 64-bit float representing a number of seconds. The G2 expression the current subsecond [real] time also returns time as a float, as described Expressions.

Float time provides effectively unlimited capacity, but processing float values is slower than processing integer values. When time values in excess of 17 years are required, use float time rather than integer time. G2 itself cannot be changed to use float time rather than integer time internally, because doing so would cause existing applications to fail.

Information about float time appears in the following locations:

- The system procedures that manipulate float time are described in Time Information Operations in the *G2 System Procedures Reference Manual*.
- The expression the current subsecond [real] time is described in Referring to the Current Time.

Time as a String

Neither integer time nor float time provide good human readability. G2 provides several formats for representing time as a string. Each of these formats is optimized for a different purpose. The available formats are:

- **Timestamp format**: *dd mon yyyy hh:mm x.m*. This format designates a point in time. It appears in the authors attribute, and in displays of times where the format is specified to be as a timestamp.
- **Interval format**: *dd* days, *hh* hours, *mm* minutes, and *ss* seconds. This format is used in the validity attribute of a variable.
- **Calendar format**: *mm/dd/yyyy/ hh:mm:ss x.m.* This format can be used as needed in applications to represent a point in time.

System procedures that manipulate string time are described in the *G2 System Procedures Reference Manual*.

Working with Composite Types

The G2 composite types are structure and sequence. Composite types are those that are composed of one or more values of any general, specific, or composite type. G2 represents system-defined attributes whose values consist of complex data structures with structures and sequences.

A sequence is a list-like value that can contain any item or value, including other sequences and structures.

A structure consists of one or more pairs of names and values. The values of a name/value pair can consist of other structures or sequences. Use structures to represent item attributes and their values. A structure requires an even number of arguments.

For more information about working with sequences and structures, see Attribute Access Facility.

Using the Structure Type

A **structure** value type consists of a set of subattribute name and value pairs, separated by a colon (:). All name-value subattribute pairs are separated with a comma (,) in this construct:

structure ([*subattribute-name: value* [,...]])

As an example, this value is the structure returned for the history-keeping-spec attribute of a variable:

```
structure
(maximum-number-of-data-points: 10,
 minimum-interval-between-data-points: 6000)
```

Structures, which can have a virtually unlimited number of name-value subattributes (up to 523,263), including zero, are functionally similar to items. As such, you can access their attributes by:

- Using standard attribute grammar such as:

  the identity of x

- Iterating over their attribute names, using an expression such as:

  for symbol = each symbol that is an attribute name of x do...

- Add or change the value of an attribute in a structure using the function change-attribute ( ).
- Remove an attribute from a structure, using the function remove-attribute ( ).

The function structure ( ) creates and returns new structures.

Because structures consist of name and value pairs, they require an even number of arguments. The values of attributes of type structure can be any item or value, including other structures and sequences.

When the subattribute of a structure consists of an item, and that item is deleted from the KB, the attribute name remains within the structure, but has a value of none.

Structure Functions

Use these functions for working with structures:

To create a new structure with given attribute values:

➔ structure
(*attribute-name*: *item-or-value* [,...])
-> *structure*

Creates a new structure containing the given attributes associated with their corresponding values.

For example:

structure (measured-item: tank-10, temp: the temp of tank-10)

When concluding new values using the structure ( ) function, omitting one or more subattributes replaces their current value with none. For example, if the history-keeping-spec attribute of a float-variable float-var-1 is currently:

keep history with maximum number of data points =
100 and maximum age of data points = 2 hours

then the value of that attribute is expressed as:

structure(maximum-number-of-data-points: 100,
maximum-age-of-data-points: 7200)

Concluding a new value for the number of data points with an expression such as:

```
conclude that the history-keeping-spec of float-var-1 =
    structure (maximum-number-of-data-points: 50)
```

results in the value changing to this:

```
structure(maximum-number-of-data-points: 50)
```

To change one or more subattributes without changing other subattributes to the value none, use the change-attribute function or a subattribute reference to conclude a new value. For example, this expression:

```
conclude that
    the maximum-number-of-data-points of
    the history-keeping-spec of float-var-1 = 50
```

changes the value of one subattribute, without changing others.

To create a new structure with given evaluated attribute values:

➔ evaluated-structure
(*symbol-expression*, *item-or-value* [,...])
-> *structure*

Creates a new structure containing the given attributes associated with their corresponding values. The difference between this function and the structure ( ) function is that this function evaluates the expressions giving the attribute names, while structure ( ) uses the names given explicitly in the form.

To create a new structure with a changed attribute:

➔ change-attribute
(*structure*, *attribute-name*, *item-or-value*)
-> *structure*

Creates a new structure containing all of the same attributes and values in the given structure, but with the given *attribute-name* containing the given *item-or-value*.

If the attribute did not exist within the argument structure, the new attribute is added to the end of that structure. If the attribute already exists within the argument structure, the function changes its value, but keeps its position in the original attribute order of the structure.

To create a new structure with a changed evaluated attribute:

➔ change-evaluated-attribute
(*structure*, *symbol-expression*, item-*or-value*)
-> *structure*

Creates a new structure containing all of the same attributes and values in the argument structure, but with the given *symbol-expression* containing the given *item-or-value*.

This is the same operation as change-attribute(), except for the attribute name being given by an evaluated expression instead of using the name given explicitly in the form.

To create a new structure with a removed attribute:

➔ remove-attribute
(*structure*, *attribute-name*)
-> *structure*

Creates a new structure containing all of the same attributes and values in the argument structure, but with the named attribute removed. If the named attribute was not in the argument, the function returns an exact copy of the argument structure.

To create a new structure with a removed evaluated attribute:

➔ remove-evaluated-attribute
(*structure*, *symbol-expression*)
-> *structure*

Creates a new structure containing all of the same attributes and values in the argument structure, but with the named attribute removed. If the named attribute was not in the argument structure, the function returns an exact copy of the argument structure.

Structure Expressions

Use the following expressions to access structures.

To return an attribute value:

➔ the *attribute-name* [*local-name*] of *structure*

Returns the value associated with the attribute name within the given structure. If no such attribute exists within the structure, or the attribute contains none, G2 signals an error.

Note If you have explicitly defined an attribute name to be lowercase by using quote characters (@), G2 signals an error of you omit the quote characters when accessing the attribute name.

To return an attribute value named by a symbolic expression:

➔ the {*class-name* | *type* } that is an attribute of *structure* named by *symbolic-expression*

Returns the value associated with the attribute named by the *symbolic-expression*. If no such attribute exists within the structure, or the attribute contains none, G2 signals an error.

To return the attribute names within a structure:

➔ each symbol [*local-name*] that is an attribute name of *structure*

Returns the symbols that name attributes within the structure.

This code fragment contains an example of each expression:

```
identification: structure =
    structure(corporation: the symbol acme, id: "456GL900")
S: symbol;
    ... ;
{ Post the value of the corporation attribute.}
post "[the corporation of identification]";

{ Post the value of the id attribute. }
conclude that id-symbolic-parameter = the symbol id;
post "[the text that is an attribute of identification named by
    id-symbolic-parameter]";

{ Post each attribute name. }
for S = each symbol that is an attribute NAME of identification do post "[S]" end;
    ...;
```

Using the Sequence Type

A **sequence** value type is a list-like entity that can contain any item or value, including other sequences and structures. Sequences can have a virtually unlimited number of elements (up to 1,046,526), including zero. Each sequence element is separated with a comma (,) in this construct:

sequence ([*item-or-value* [,...]])

The next example shows the sequence representing a portion of the item-configuration attribute value.

```
sequence (the symbol developer, the symbol user)
```

Sequences are functionally similar to lists. As such, you can access their elements by:

- Iterating over each element within a sequence.
- Adding to the beginning or end of a sequence using the functions insert-at-beginning ( ) and insert-at-end ( ).
- Inserting after a particular item or value in a sequence, using the insert-after ( ) function.
- Inserting before or after an element at a particular index in the sequence using the functions insert-before-element ( ), insert-after-element ( ).
- Using an element index ([0]) to address an element directly.

When referencing and using sequences, remember that, unlike lists and array items, sequences are values. While you can use sequences in some list and array expressions, you pass sequence values as a copy, rather than as a reference, and change their values using the sequence functions.

Note Deleting an item contained in a sequence changes the element value to none, but does not decrease the number of elements.

Sequence Functions

Use these functions to create and manipulate sequences.

To return a new sequence containing the given elements:

➔ sequence
(*item-or-value* [,...])
-> *sequence*

Returns a new sequence containing the given elements. Sequences may contain from zero to 1,046,526 elements.

For example:

```
get-debug (P:class procedure) = (sequence)
SEQ: sequence;

begin
    SEQ = sequence (the tracing-and-breakpoints-of P);
    return SEQ
end
```

To return a new sequence one element shorter than the given sequence:

➔ remove
(*sequence*, *integer*)
-> *sequence*

Returns a new sequence one element shorter than the given sequence, where the element at the given index has been removed. The first element is at index 0. If there is no element at the given index, an error is signalled.

To return a new sequence whose first element is the given item-or-value:

➔ insert-at-beginning
(*sequence*, *item-or-value*)
-> *sequence*

Returns a new sequence whose first element is the given *item-or-value*, and whose remaining elements are all elements in the given sequence.

To return a new sequence with specific elements inserted:

➔ insert-at-end
(*sequence*, *item-or-value*)
-> *sequence*

Returns a new sequence whose elements are those in the given sequence, but which also contains an additional last element which is the given *item-or-value*. This operation is generally faster than insert-at-beginning for incrementally collecting large sequences.

Sequences are stored as data arrays (not items), potentially with some empty elements at the end of the array as the sizes of data structures are rounded up to allocated sizes. In many cases, elements added to the end of sequences may be filled into these empty locations without having to shift the previous elements in the sequence.

To return a new sequence with inserted arguments:

➔ insert-after
(*sequence*, *item-or-value*, *item-or-value*)
-> *sequence*

Returns a new sequence with the same elements as the argument sequence, but with the second argument inserted after the first occurrence of the third argument within the sequence.

To return a new sequence with inserted elements before a given index:

➔ insert-before-element
(*sequence*, *integer*, *item-or-value*)
-> *sequence*

Returns a new sequence with the same elements as the argument sequence, but with the third argument inserted into the sequence at the index given as the second argument.

The allowable range for the index argument of a sequence is:

-1 to (*number of elements* - 1)

This operation can insert the new *item-or-value* as the new first element, new last element, or at any location within the sequence.

If an index is given outside of this range, an error is signalled.

To return a new sequence with inserted elements after a given index:

➔ insert-after-element
(*sequence*, *integer*, *item-or-value*)
-> *sequence*

Returns a new sequence with the same elements as the argument sequence, but with the third argument inserted into the sequence immediately after the index given as the second argument.

The allowable range for the index argument of a sequence is:

-1 to (*number of elements* - 1)

This operation can insert the new *item-or-value* as the new first element, new last element, or at any location within the sequence. If an index is given outside of this range, an error is signalled.

To change a single element of a sequence:

➔ change-element
(*sequence*, *integer*, *item-or-value*)
-> *sequence*

Returns a new sequence with the same elements as the argument sequence, but with the value at the index location changed to the given *item-or-value*. The first element is at index 0, and it is an error if the index is larger than the current size of the given sequence.

To concatenate two or more sequences:

➔ concatenate
(*sequence*, *sequence* [,...])
-> *sequence*

Returns a new sequence containing the combined elements of the argument sequences, with all elements of those sequences concatenated in order to form the new sequence.

To get a portion of a sequence:

➔ portion
(*sequence*, *integer*, *integer*)
-> *sequence*

Returns a new sequence containing a portion of the elements of the argument sequence. The first integer argument is the index at which to start copying, and the second integer argument is the number of elements to return in the new sequence.

Sequence Expressions

Use these expressions to access sequences:

To obtain a particular element in a sequence:

➔ {*sequence*} [*integer*]

Returns the nth *item-or-value* in the given sequence, where *integer* is a zero-based index.

To iterate over elements in a sequence:

➔ the {*class-name* | *type*} [*local-name*] in {*sequence*}
-> [*item* | *integer* | *float* | *symbol* | *text* | *truth-value* | *structure* | *sequence*]

Returns each element of the given type found within the sequence.

To return an element of a particular type from a sequence:

➔ the {first | second | next to last | last {*class-name* | *type*} in {*sequence*}

Returns the element of the given type at the described position within the sequence.

To determine whether an item-or-value is a member of a sequence:

➔ {*item-or-value*} is [not] a member of {*sequence*}

Returns whether or not the given *item-or-value* is a member of the sequence.

When testing for membership in a sequence, G2 ignores the alphabetic case when comparing two text values and ignores the type when comparing two quantity values. For example:

- The text string "Text" is a member of the sequence that contains "text".
- The float 2.0 is a member of the sequence that contains the integer 2.

To determine the number of elements in a sequence:

➔ the number of elements in {*sequence*}

Returns the number of elements in the given sequence.

Using Structures and Sequences in User-Defined Classes

You can use structures and sequences as user-defined attribute values in class definitions. Unlike the general and specific G2 value types (quantity, integer, float, and so on), both structures and sequences can consist of multiple values:

- Sequences can contain values of items and all value types, including other sequences and structures.
- Structures can have values of items and all value types, including other structures and sequences.

While structures and sequences offer similar functionality to lists and other items, they consume considerably less memory. If your class-specific attributes do not require the full capabilities that items provide, we recommend that you use:

- Structures to represent items.
- Sequences to provide list-like functionality.

Comparing Structures and Items

The fundamental properties of structures and items are:

Property	Structures	Items
Has iconic representation		✓
Has methods		✓
Has inheritance		✓
Can save as permanent knowledge	✓	✓
Consists of attributes and values	✓	✓
Can conclude values into attributes		✓
Has fixed set of attributes defined by a class definition		✓
Has arbitrary set of attributes that can be added to and removed from per instance	✓	
Must be created and deleted explicitly		✓
Requires minimum memory	✓	
Memory and existence are managed automatically	✓	

Comparing Sequences and Lists

The fundamental properties of sequences and lists are:

Property	Sequences	Lists
Has iconic representation		✓
Requires minimum memory	✓	
Can save elements as permanent KB knowledge	✓	✓
Can have elements of structures and sequences	✓	
Programmatically manipulate elements	✓	✓

10

G2 Items

Presents the characteristics that are common to all G2 items.

Introduction

Items are the fundamental data structures within G2 that you use to represent knowledge. You use items to collect and organize knowledge about real objects, processes, and relationships. You use G2 to collect and organize a set of items in a knowledge base (KB). The items in a KB represent a set of application knowledge.

Each item represents knowledge that has a distinct identity, that persists, and which you can reference directly or indirectly. Each item also represents a set of knowledge that has a particular pattern or template, based on its class. G2's object-oriented support for defining items enables you to design custom classes and to create as many items of each class as required.

As you develop a KB, you work with items interactively by creating them, naming them, moving and transferring them upon workspaces, and so on. When G2 runs the current KB, the KB's own processing works with items by reasoning about them programmatically in actions, rules, procedures, functions, and formulas.

Note To perform an operation *programmatically* means that you perform it by executing a G2 executable item, such as an action button, rule, procedure, method, and so on. To perform an operation programmatically, the current KB must be running.

Items play the role of objects in other object-oriented programming languages. For historical reasons, G2 uses the term *item* rather than *object*.

Logical Components of Items

Through its class inheritance, each item contains information that enables it to represent various kinds of knowledge. Internally, every item consists of several logical components, which may be accessible interactively, programmatically, or both:

Logical Component	Description
table attributes	The attributes of an item that are displayed in its attribute table.
hidden attributes	The internal attributes of an item that are displayed on its table of hidden attributes. These attributes have been made accessible through the attribute-access facility.
status	Information about whether an item is one of several pre-defined states: ok, incomplete, or bad. The status of an item also includes information such as whether the item is permanent or transient, enabled or disabled, activated or deactivated.

Logical Component	Description
position	The workspace x and y coordinates of an item upon a workspace. You can return the integer value representing an item's position, using the expressions: the item-x-position of *item* the item-y-position of *item*
size	The width and height of the icon of an item in workspace units. You can return the integer value representing an item's width and height, using the expressions: the item-width of *item* the item-height of *item*
representation	The color or color-pattern of an item. For example, you can interactively or programmatically change the background-color of a workspace. Similarly, you can change the color of the named regions of an item's icon, or for textual items, such as messages, the text color or size.

The logical components of items are further described in Understanding the Knowledge Contained in Items and Item Representation.

You work with items interactively using the G2 developer's environment. By default, when you click the mouse on an item, it displays its menu. An item menu presents operations that you can apply to that item.

You can display the values stored in the attributes of an item by displaying the item table. Each table shows the name and class of the item, its list of attributes, and the current value of each attribute. You can also display the current value of a particular attribute by creating an attribute display, which appears next to the item itself upon a workspace.

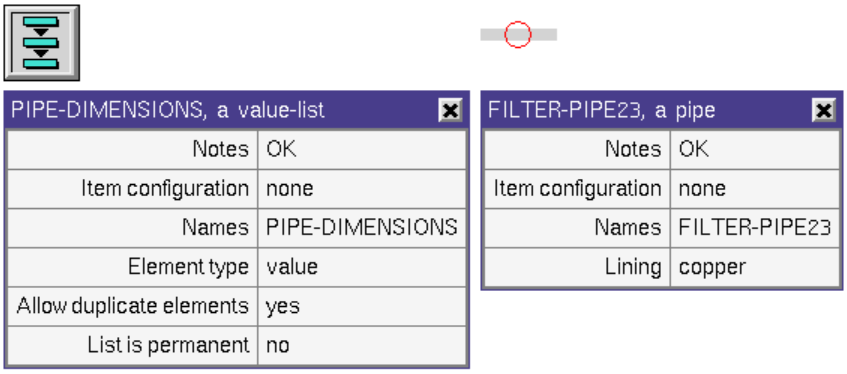
Understanding Item Inheritance

Each item is an instance of a class. An item's class defines the template for the knowledge it can contain. Each class is associated with at least one parent, or superior, class. A class can also have subclasses, whose definitions are based on the definitions of their parent classes. For further information see Classes and Class Hierarchy.

An item's class defines its set of **attributes**. Each attribute can contain a value, or piece of knowledge. Most of the knowledge that your KB manipulates resides in the attributes of the KB's items.

All items are instances of some subclass of the item class, which is the root class in the system-defined class hierarchy. The item class defines three attributes, notes, names, and item-configuration, which all subclasses of the item class inherit. By definition, each item in your KB has these attributes.

The next figure shows a workspace that contains one value-list and an instance of a user-defined class. Both the value-list and the pipe item inherit their top-three attributes from item class:



There are items of some system-defined classes that do not use the names attribute, but rather define a class-specific attribute to contain the item's identifier.

For instance, because a procedure can be invoked, as well as referenced for its attribute values, its name is based on the declaration found in its text. Similarly, because a relation definition establishes two relationships, a relation and an inverse relation, it uses the class-specific relation-name attribute instead of the names attribute.

Understanding the Knowledge Contained in Items

Each item represents a set of knowledge whose template is based on its class. An item's set of attributes represents such a template. However, each item also contains other information that G2 maintains, such as its status and relationships with other items.

Identifying the Knowledge in Attributes

The class of an item determines its set of attributes. An item's attributes contain knowledge in a form that is easy to work with. Attributes are described in detail in Attributes and Tables.

Identifying the Knowledge Not Stored in Attributes

Items of some classes can contain one or more values that are distinct from the items' attributes. For instance, a variable or parameter can contain a value, and can be defined to also contain history datapoint values. Also, lists and arrays can contain values and references to other items in their elements. See the appropriate chapter for more information about the knowledge that these items can contain.

Further, items also include a set of hidden attributes, which are those that do not appear in an attribute table, and include:

- The item name box.
- Attribute display.
- Relationships.
- Containing module.

For more information about hidden attributes, see Attribute Access Facility. Obtaining the relationships in which an item is participating is described in Referring to the Relationships of an Item.

Identifying the Status Knowledge of Items

Items also contain several kinds of **status** knowledge:

- Permanent/transient: Whether the item is retained in the current KB after you reset or reset it.
- Active/inactive: Whether other items in the KB can reference the item. This is determined by whether the item's parent workspace has been activated.

- Enabled/disabled: Whether the item can be activated. This is determined by whether you have interactively selected enable or disable from the item's menu.
- Participation: Whether the item's attributes contain enough information, or the right information, to participate in processing.

G2 automatically maintains each item's status knowledge.

Permanent/Transient Status

At a given moment, each item in the current KB is either permanent or transient. The permanent/transient status of items is user-settable, but only programmatically.

A **permanent item** continues to exist in the current KB after the KB is reset or restarted. When you save the current KB to a file, only the KB's permanent items are stored in the KB file.

A **transient item** does not continue to exist in the current KB after the KB is reset or restarted. Likewise, a KB that has been loaded but not yet started contains no transient items. When you save the current KB to a file, the KB's transient items are *not* stored in the KB file.

After being created interactively, an item is permanent. For instance, any item that you create by selecting from the KB Workspace > New Object menu is a permanent item.

When you create an item programmatically, using the create action, that item is transient.

To make an item permanent:

➔ make permanent *item*

To make an item transient:

➔ make transient *item*

For instance, this rule creates a new transaction message and makes it permanent:

```
for any transaction T
    if the status of T is not message-sent
        then in order
            create a transaction-message TM and
            conclude that the status of TM is unsent
                    and make TM permanent
```

Changing the permanent/transient status of an item causes G2 to propagate the new status to all items below it in the workspace hierarchy.

Note You cannot make a permanent item the subordinate item of a transient item. For instance, you cannot transfer a permanent item to a transient workspace, and you cannot make a permanent workspace the subworkspace of a transient item.

G2 provides the g2-system-predicate system procedure to obtain any item's permanent/transient status:

To determine if an item is permanent or transient programmatically:

➔ g2-system-predicate
(*item-to-check*: item-or-value; *predicate*: symbol)
-> *permanent-transient-showing*

Returns a truth-value indicating the predicate you pass to the procedure, which can be permanent, transient, or showing.

Active/Inactive Status

The active or inactive status of an item indicates whether G2 has activated it. When an item is inactive, it cannot be referenced by other items.

You can only set the active/inactive status of activatable subworkspaces. You perform this action programmatically.

To activate an activatable subworkspace programmatically:

➔ Use the activate and deactivate actions.

When you start or restart the current KB, G2 automatically activates all enabled top-level workspaces, then automatically propagates those workspaces' active/inactive status to all items below them in the KB's workspace hierarchy. All enabled items on active workspaces are activated.

Note When you deactivate the subworkspace of an item, G2 behaves as though the items upon the subworkspace do not exist. All items upon the subworkspace are no longer active. The subworkspace itself, however, can still be referenced and is included in existence checks such as the count of each kb-workspace.

For more information about how G2 activates workspaces, see Activating and Deactivating Workspaces.

In addition, G2 cannot activate these items:

- Disabled items.
- Items whose status is bad.
- Items directly or indirectly subordinate to a non-activatable item (see Identifying the Superior and Subordinate Relationships among Items).

Only an active variable can have a current value and a history. Only an active list or array can contain values in its elements.

Referencing Inactive Definitions

If a class definition is not active, the class that it defines continues to exist. Likewise, relation instances continue to exist even when the relation definitions on which they are based become inactive.

Enabled/Disabled Status

The enabled/disabled status of an item refers to whether the item can be activated. By default, when you create an item interactively, it is enabled and can thus be activated.

You can disable an item interactively or programmatically. Once disabled, an item is effectively deactivated, and its activation status is inactive. Changing the enabled/disabled status of an item propagates to the items below it in the KB workspace hierarchy. Enabling or disabling a workspace affects the items that reside upon it, causing them all to become deactivated. Enabling one or more items that reside upon a disabled workspace has no effect until the workspace status is active and enabled.

To change the enabled/disabled status of an item interactively:

➔ Select enable or disable from the item's menu.

To enable all disabled items in the current KB:

➔ Select Main Menu > Run Options > Enable All Items.

To change the enabled/disabled status of an item programmatically:

➔ g2-system-command
(*command*: symbol, *win*: class g2-window, *item*: class item,
attribute: symbol)

where:

command is enable or disable.

For details, see the *G2 System Procedures Reference Manual*.

You can change an item's enabled/disabled status at any time, regardless of the KB's run-state. Changing this status does not affect the item's other status values.

When you enable an item, G2 immediately activates the item, unless there is another item above it in the workspace hierarchy that is not activated. If the enabled item is an activatable subworkspace, it is activated only when it is the target of an activate action.

When you disable an item, G2 cannot activate it, even if the item meets all other criteria for activation.

When you save the current KB, G2 also saves the knowledge of which items are disabled. Thus, items will continue to be disabled after you next load that KB.

Participation Status

The status of an item can be ok, incomplete, or bad and reflects whether the item knowledge is valid for KB participation. G2 changes the participation status of an item appropriately as the knowledge an item contains is updated.

The ok, incomplete, or bad status of each item appears in its notes attribute. The next table summarizes the meaning of each setting:

Item Status	Description
ok	All attributes have valid values, and a sufficient number of attributes have values to permit the item to participate in the KB's processing.
incomplete	At least one attribute, whose setting is required for the item to participate in KB processing, requires a different value.
bad	At least one attribute does not have a valid value.

For example, after you create a new class definition, G2 initializes the status of the item to incomplete until you specify new values for the item's class-name and direct-superior-classes attributes.

The notes attribute of an item can also contain other useful information. For example, the status of an item may be ok, but if the item resides upon a disabled workspace, it cannot participate in KB processing. Such a status is displayed in the item notes attribute with a message such as:

```
OK, but some superior item is either DISABLED or not OK.
```

Because G2 reports the participation status of an item in the notes attribute, you can reference this status in expressions. For instance, when debugging your KB, use the Inspect facility to construct a command like this:

```
show on a workspace every acid-bath-tank whose item-status is incomplete

    or

show on a workspace every help-organizer with notes
```

These commands display items whose notes attribute does not contain the value ok.

The notes attribute is a composite attribute, as described in Attribute Access Facility. You can refer directly to the status information of active items that

the notes attribute contains by referring to the item-status of an item, and to the actual notes of an item using the item-notes.

To refer to the item-status of an item:

➔ the item-status of *item*
-> {OK | INCOMPLETE | BAD}

To refer to the item-notes of an item:

➔ the item-notes of *item*
-> {none | sequence ([text [,...]] }

Identifying the Superior and Subordinate Relationships among Items

An item's knowledge includes whether it has a superior or subordinate relationship to other items. G2 considers information about the following relationships to be part of an item's knowledge:

- The relationship between a KB workspace and the items upon it.
- The relationship between an item and its subworkspace.
- The relationship between an item and the object that is contained in an attribute; this includes an attribute that is an instance of an object or given by a variable or parameter.

Tip You can use expressions to refer to the item that is superior or subordinate to another item. See Referring to the Superior Item.

G2 propagates knowledge from item to item along the lines of the superior and subordinate relationships, including:

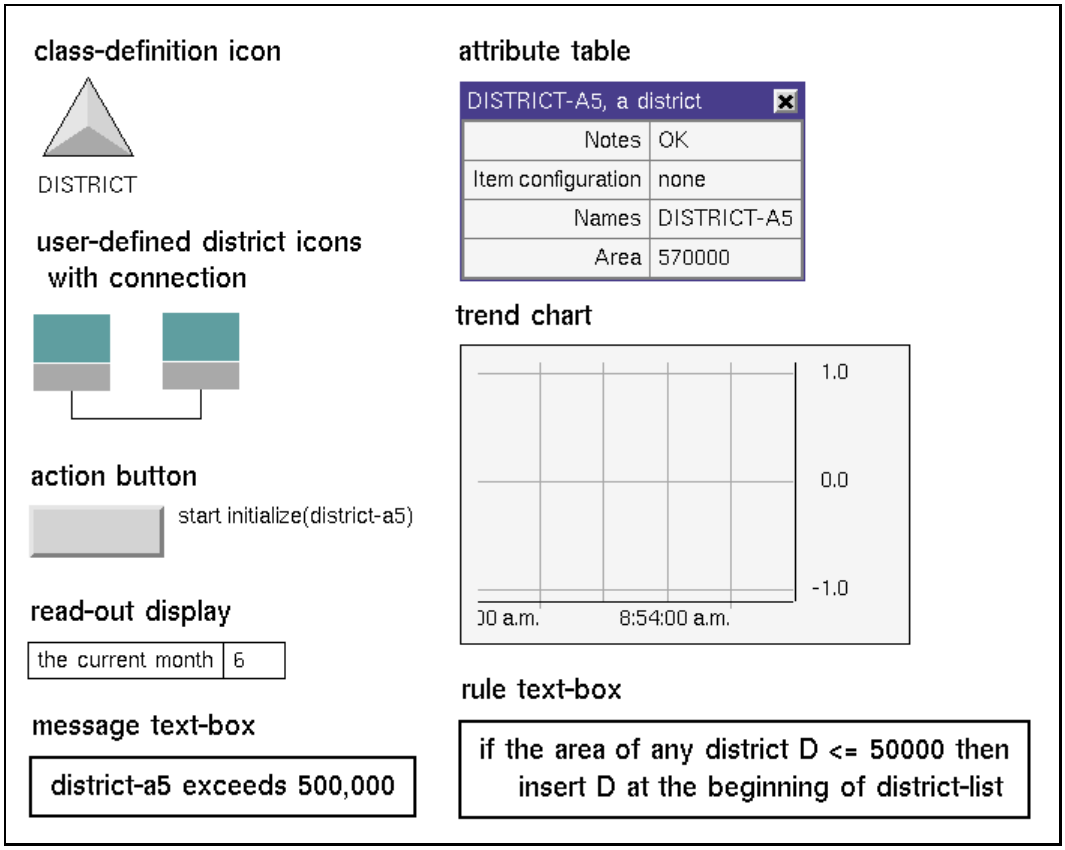
- The active/inactive, permanent/transient, and enabled/disabled status of items.
- The item configurations.

Item Representation

The visible portion of an item's knowledge is called its **representation**. An item's representation is determined by the **representation style** of its class, for example, its icon, text box, workspace, table, chart, and so on.

Note The representation of an item (other than connections) always occupies a rectangular region on the screen, even if the visible portion of the representation is not rectangular.

The following figure displays some item-representation styles:



Identifying the G2 Color Palette

G2 supports a large set of colors. G2 displays its color palette when you select color on the menus of items. You can assign any supported color to any color attribute of an item, or to any region of the icon of a system-defined or user-defined class.

The G2 color palette provides these colors:

antique white	aquamarine	azure
beige	black	blue
brown	cadet blue	coral
cyan	dark gray	dark slate blue
dim gray	extra light gray	floral white
forest green	gold	goldenrod
gray	green	green yellow
indian red	ivory	khaki
lavender	light blue	light cyan
light goldenrod	light goldenrod yellow	light gray
light pink	light steel blue	light yellow
lime green	linen	magenta
maroon	medium aquamarine	medium blue
medium goldenrod	medium orchid	orange
pale goldenrod	pale green	pale turquoise
pink	plum	powder blue
purple	red	salmon
sienna	sky blue	slate blue
smoke	tan	thistle
turquoise	violet	violet red
wheat	white	yellow

The G2 color palette also includes the metacolors foreground, background, and transparent. Each **metacolor** assigns a color value for an item's color attribute by referring to a color attribute of another item:

- A metacolor of foreground means that the actual color is determined by the color value of the foreground color attribute of the item's parent workspace.
- A metacolor of background means that the actual color is determined by the color value of the background color attribute of the item's parent workspace.
- The metacolor transparent means exactly that: whatever you assign to this metacolor becomes transparent. Any item beneath a transparent item or icon becomes visible.

  You can select and drag a transparent item. Any visible knowledge that is behind the transparent item, or other entity with color, is visible.

  You cannot set the background-color color attribute of a workspace to transparent.

Identifying the Color Attributes of Items

Each representation style has a corresponding set of color attributes. A **color attribute** is a component of an item's representation that can appear in a distinct color.

Each item representation presents a set of color attributes:

- The *icon* representation has the icon-color color attribute.
- The *text box* representation has the text-color, border-color, and background-color color attributes.
- The *workspace* representation has the foreground-color and background-color color attributes.
- The *connection* representation has the stripe-color color attribute.

Other item representations do not have settable color attributes.

Note The icon-color color attribute of an item is distinct from the color regions defined for its icon. For more information about icon color regions, see Composition of an Icon.

The settings of an item's color attributes are part of its knowledge. You can set the color attributes of items interactively or programmatically.

To set a color region of an item interactively:

➔ Select the color choice on the item's menu as described in Changing the Color of an Item.

Actions That Affect Item Appearance

G2 provides the following actions that change an item color or pattern:

To change a color attribute of an item:

➔ change the *color-attribute-name* of *item* to
{*color-name* | *symbolic-expression*}

To change a color pattern of an item:

➔ change the color-pattern of *item* so that
{*color-attribute-name* is *color-name*} [, ...]

For items with an iconic representation, you can programmatically set the icon-color region as well as any other user-defined icon color-region.

To change an icon region of an item:

➔ change the *region-name* icon-color of *item* to *color-name*

You can provide a symbol of the form RGB*rrggbb* as a valid color name, where *rr*, *gg*, *bb*, are the 8-bit hex values for red, green, and blue. For details, see Other Literal Terms.

Locating Items upon a Workspace

The location of each item upon its parent workspace is part of its knowledge. Note that some items do not reside upon a workspace, yet they are still part of the KB's knowledge.

The location of a workspace within a G2 window is part of its knowledge.

Note G2 maintains information about whether a workspace is being displayed, and at what scale, on a per-window basis; it does not maintain this information in the workspace item. For more information about the relationships among workspaces and the windows of G2 and Telewindows processes, see G2-Windows.

Layering Items upon the Same Workspace

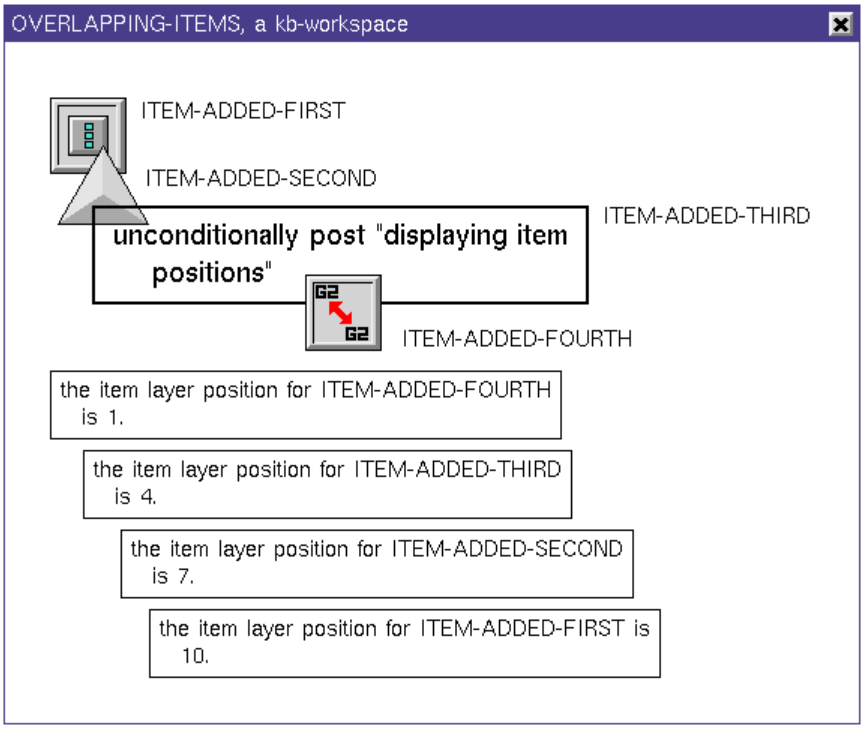
Item layering refers to how G2 draws the representations of items in a top-to-bottom manner upon a workspace. The layering of an item is part of its knowledge.

Note G2 includes two drawing modes, Paint and XOR. The XOR drawing mode is a superseded capability. Your KB should use only the Paint drawing mode, described in Drawing Parameters. For further information, see Appendix F, Superseded Practices.

When Paint drawing mode is in effect and the representations of two items intersect, G2 displays those items so that they overlap. Each item's **item layer position** determines which item appears on top. Each item upon a workspace has a unique item layer position, which is an integer value of zero or higher that G2 sets and maintains. An item whose item layer position is zero appears on top of all other items upon that workspace.

In the next example, at the bottom of the overlapping-items workspace, the messages created by the execution of report-item-layer-positions procedure report the item layer positions of the other four items on the workspace. The item layer positions correspond to the overlapping appearances of the items' representations on the workspace.

```
report-item-layer-positions()
ITEM: class item;
M: class message;
SYM: symbol;
POS: integer;
begin
    for ITEM = each item upon overlapping-items
        do
            SYM = call g2-name-for-item(ITEM);
            POS= call g2-get-item-layer-position(ITEM);
            create a message M;


            transfer M to overlapping-items at
                (-325 + (20 * the count of each message upon
                    overlapping-items),
                100 - (50 * the count of each message upon
                    overlapping-items));
            change the text of M to "the item layer position for [SYM] is [POS]."
        end
end
```



In general, when you add or transfer an item to a workspace, that item appears on top of all other items already on that workspace. The first item placed upon a new workspace has an item layer position of zero. This is true whether you create the item on that workspace or transfer the item from another workspace. As the set of items on a workspace changes, G2 automatically adjusts the item layer position values of the items that remain in the workspace.

Note Other entities displayed on a workspace, such as its name box or an attribute table, also have their own item layer positions. For this reason, at any one point in time, the item layer positions of the items on your workspace might not include the value zero or be consecutive.

Your application should *not* rely on the absolute value of any item's item layer position. Rather, your application should rely on the relative differences among the layer positions of items.

Distinguishing Permanent, Transient, and Current Knowledge

After you reset G2, the current KB contains only one version of each item's knowledge: its **permanent knowledge**. An item's permanent knowledge is the set of attribute values, status values, and item relationships that are in effect when G2

is reset. When you interactively create an item or change the value of an attribute when G2 is reset, G2 adds that item or value to the permanent knowledge of the current KB. G2 saves only permanent knowledge to a KB file, unless you direct G2 to save a snapshot file of your KB.

After you start the current KB, you can add both transient knowledge and permanent knowledge to the current KB. **Transient knowledge** is removed from the current KB when G2 is reset. The transient and permanent knowledge that exists in the current KB when G2 is running or paused is known collectively as **current knowledge**. G2 uses only the current knowledge when it is running your KB.

When G2 is running, you create transient knowledge by creating transient items, relationships, and array and list elements; and by changing attribute values using the change and change the text of actions.

Here are some actions that produce transient knowledge:

```
change the text of the length of cable45 to "15.3"

create a generator

change the name of the generator upon this workspace
    to the symbol test-generator-9

conclude that the list-is-permanent of accounting-list is false;
insert 5 at the beginning of accounting-list
```

You create permanent knowledge by creating permanent items, relationships, and array and list elements; and by changing attribute values interactively, by executing the conclude action, and by executing the change and change the text of actions followed by a make *item* permanent action.

Here are some actions that produce permanent knowledge:

```
conclude that the length of cable45 = 15.3

change the text of the length of cable45 to "15.3";
make cable45 permanent

create a generator G;
make G permanent

conclude that the names of the generator upon this workspace
    = the symbol bozo
```

How Using Change Actions Effects the Current Knowledge of the KB

In a running G2 session, the first time you change an attribute's value using the change or change the text of action, G2 first copies its permanent value to an internal attribute and then applies the change to the attribute. The new attribute value is transient because the saved permanent value will be reinstated when G2 is reset.

Note Once you have made a transient change to an attribute value, G2 will reinstate the original permanent value even if you execute subsequent permanent change actions on the attribute within the running G2 session.

In the following example, the value of the length of cable45 will revert to 15.3 upon a even though a transient change action has been followed by a permanent conclude action within a running G2 session:

```
conclude that the length of cable45 = 15.3;
change the text of the length of cable45 to "0";
conclude that the length of cable45 = 2000
```

An Example Using Permanent and Transient Knowledge

To demonstrate how an item's permanent and transient knowledge differ:

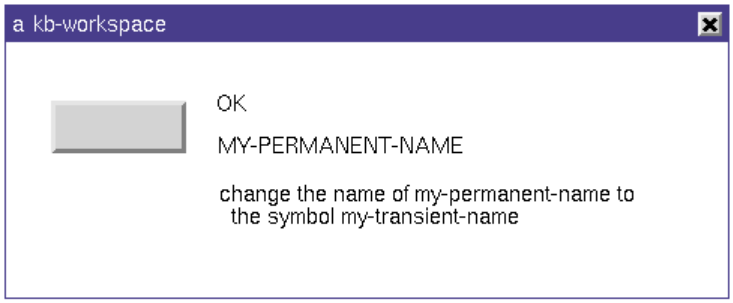
1. Reset the current KB.
2. Create a new workspace by selecting Main Menu > New Workspace.
3. Create a new action button and place it upon a workspace by selecting KB Workspace > New Button > action-button.

   Because you created this new button interactively, you added a permanent item to the current KB. The default value of the new button's attribute, its default color, and its subordinate relationship to its parent workspace are pieces of the new button's permanent knowledge.
4. Open the new button's attribute table by selecting table from its menu.
5. Edit the name of the button to be my-permanent-name.

   Editing the button's name interactively updates the button's permanent knowledge.
6. Edit the action attribute of the button to:

   ```
   change the name of my-permanent-name
       to the symbol my-transient-name
   ```

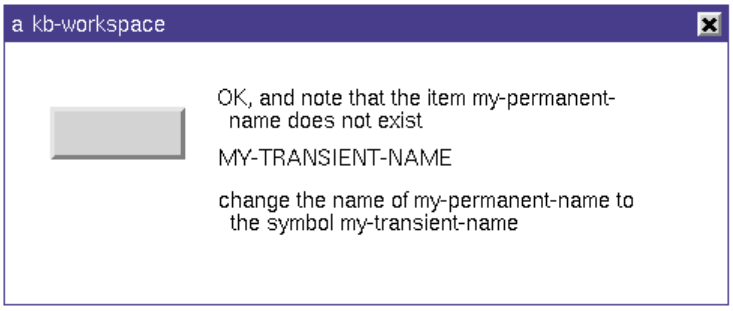
   With attribute displays, your workspace should now look similar to this:



7 Start the current KB by selecting Main Menu > Start.

8 Press the new action button.

This causes G2 to perform the button's action, which updates the display of the value of its names attribute in the table and adds a note to the action button indicating that the item my-permanent-name does not exist:



Executing this action updates the button's current knowledge with transient data, but not its permanent knowledge. By pressing the button, you invoke a change action, which performs a transient programmatic change to the item's knowledge.

9 To confirm this, reset the KB again by selecting Main Menu > Reset.

Notice that resetting the current KB causes G2 to update the display of the button's names attribute in the table and to remove the note about the item not existing. It again shows the value my-permanent-name, part of the button's permanent knowledge.

Working with Items Interactively

When G2 is running, part of its memory contains all the items in the current KB. By default, when you start G2, the current KB is empty. You use the developer's environment to add items to the current KB. After you add some number of items, you save the current KB into a KB file. Working with the current KB and with KB files is described in Knowledge Bases.

To place items on a workspace interactively:

1 Create a new workspace by selecting Main Menu > New Workspace.

G2 creates a new empty workspace.

2 To add an item to this workspace, click on the workspace background and select one of the menu choices on the KB Workspace menu that begins with the word New.

For example, to create a variable:

a Select the New Object menu choice.

b Select g2-variable from the choose a class submenu.

c Choose logical-variable from the choose a class submenu.

G2 automatically creates a new logical variable item and attaches its icon to the mouse pointer.

3 Position the mouse and click to place the icon on the workspace.

The new item now resides upon that workspace.

When you create an item interactively, you see it appear on the screen. Items have different kinds of appearances, such as icons or text boxes. Each kind of item appearance is called its representation. For a description of the kinds of item representations in G2, see Item Representation.

To learn more about working with workspaces, see Workspaces.

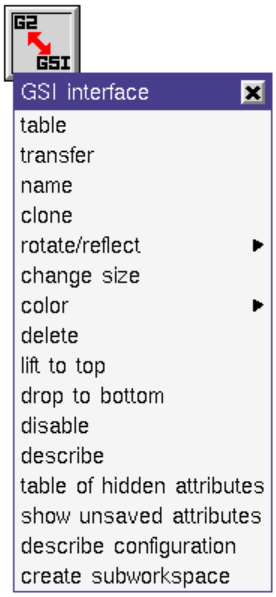
Using Item Menus

You perform an operation on an item interactively by selecting a choice from its menu.

To open an item's menu:

➔ Click the mouse on the item.

As the next figure shows, clicking the mouse on an item causes G2 to display the item's menu over or near the item. The title bar of an item's menu shows the item's class.



To dismiss an item's menu:

➔ Click the mouse on the title bar of the menu.

G2 automatically positions and scales a menu so that it is entirely visible within the G2 process's window.

Common Item Menu Choices

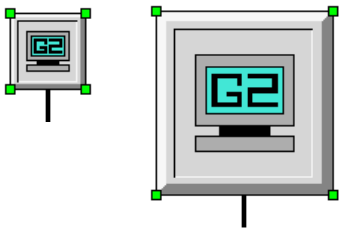
This table lists several interactive operations that are common to most classes of items:

Menu Choice	Description
change size	Open a workspace that lets you resize the item. **Note:** This menu choice is only available when the show-selection-handles attribute in the Drawing Parameters system table is false.
clone	Create a new copy of the item.
color	Change a color setting.
create subworkspace	Create a new workspace that is subordinate to this item.

Menu Choice	Description
delete	Remove this item from the current KB.
describe	Display the Describe workspace for this item.
describe configuration	Display the inheritance of configurations for this item.
enable disable	Allow or disallow this item to participate in the KB's processing.
lift to top drop to bottom	Display the item so that it is on top of all other items upon this workspace; display the item so that it is beneath all other items upon this workspace.
name	Edit the name of this item.
rotate/reflect	Rotate the item's representation in increments of 90 degrees; display the item's representation with mirrored appearance.
show unsaved attributes	Display the table for the item with permanently changed attributes highlighted.
table	Display the item's attribute table.
table of hidden attributes	Display the item's hidden attributes table.
transfer	Allow you to drag the item to another workspace.

Changing the Size of an Item

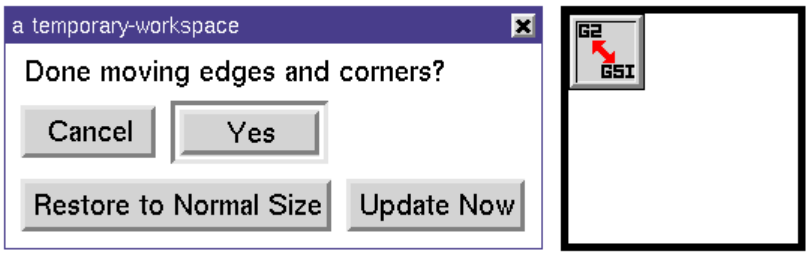
By default, items have selection handles when you select them. To resize an item, drag the selection handles. For example:



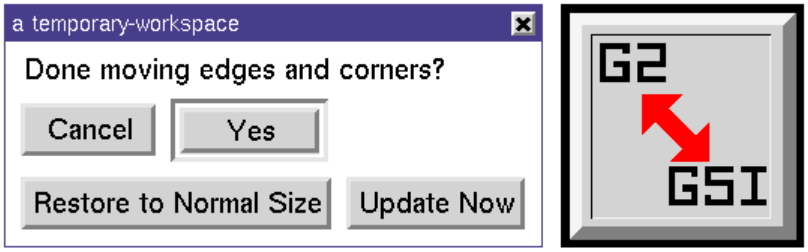
The show-selection-handles attribute of the Drawing Parameters system table determines whether selection handles appear on items. By default, show-selection-handles is true, which shows selection handles. When show-selection-handles is false, selection handles do not appear; instead, the change size menu choice appears in the item menu for changing the size.

Selecting the change size menu choice opens a dialog that you use to change the size of this item's representation. G2 encloses the item in a rectangle with a thick border. To change the size of an item, move any edge or corner of the rectangle, then press the Update Now button.

For example, this figure shows how you can enlarge an icon:



(1)
Edges and corners have been moved.



(2)
The icon image has been updated.

When you finish changing the size, either:

- Click the Cancel button to revert the item's representation to its previous size.
- Click the Yes button to retain the change you made.

Note Every G2 item has a maximum size limit, beyond which the change size option has no effect.

Cloning an Item

Cloning an item means to make a copy of it and all its knowledge. You can clone items interactively or programmatically.

To clone an item interactively:

1 Select the clone choice on its menu.

   This causes G2 to create a copy of the selected item and to attach the new item to the mouse pointer.

2 Move the mouse to a location upon this workspace or upon any other visible kb-workspace, then click the mouse.

   This causes G2 to place the new item at that location upon the workspace.

Interactively, you can clone either permanent or transient items. Cloning a permanent item interactively results in a permanent cloned item; cloning a transient item interactively results in a transient cloned item.

To clone an item programmatically:

➔ Execute the create by cloning action.

Unlike when cloning an item interactively, executing the create by cloning action always causes G2 to create a transient cloned item that does not reside upon any workspace. For the new item to become visible, your KB must also execute the transfer action, to place the new item upon a kb-workspace. You can also optionally execute actions that make the cloned item permanent.

For more information about the create by cloning action, see Creating an Item by Cloning Another.

Cloning Specific Knowledge

In most cases, a cloned item's knowledge includes all the knowledge of its source item, if it is valid for a new copy of that knowledge to exist in the current KB. For instance, a cloned item has the same attribute displays that are defined for its source item. However, an item that is cloned from another item that is the value of an attribute cannot also be the value of the same attribute.

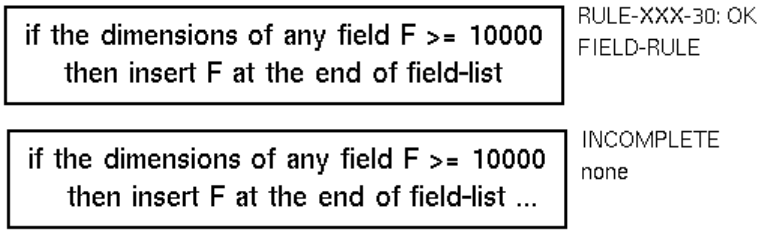
You can clone a KB-workspace. This results in a new workspace item, cloned items upon the new workspace that are copies of source items upon the source workspace, cloned subworkspaces, if any, of the cloned items, and so on. However, if you clone a workspace that is the subworkspace of another item, the clone workspace is not also the subworkspace of that item.

A cloned item does not automatically participate in the relations already established between its source item and other items. A cloned item does *not* have a value in its names attribute. This reflects the philosophy that, in general, it is less desirable to have more than one item with the same name, especially among items of the same class.

After cloning an executable item (that is, a rule, procedure, method, and so on), G2 initializes the item so that its OK/incomplete/bad status is *incomplete*. This means that the item must be edited or explicitly recompiled before G2 allows it to

be invoked. For a cloned rule, G2 also appends ellipses (...) to the rule's text, to indicate visually that the rule's status does not allow it to be invoked.

The next figure shows a named rule and a rule that has been cloned from the named rule. The cloned rule's text has ellipses indicating it is not invokable, and its attribute displays show that it is incomplete and that it has no value in its names attribute:



Changing the Text Alignment of an Item

The align text choice for items with a text box representation allows you to align the text left, right, or center.

Changing the Color of an Item

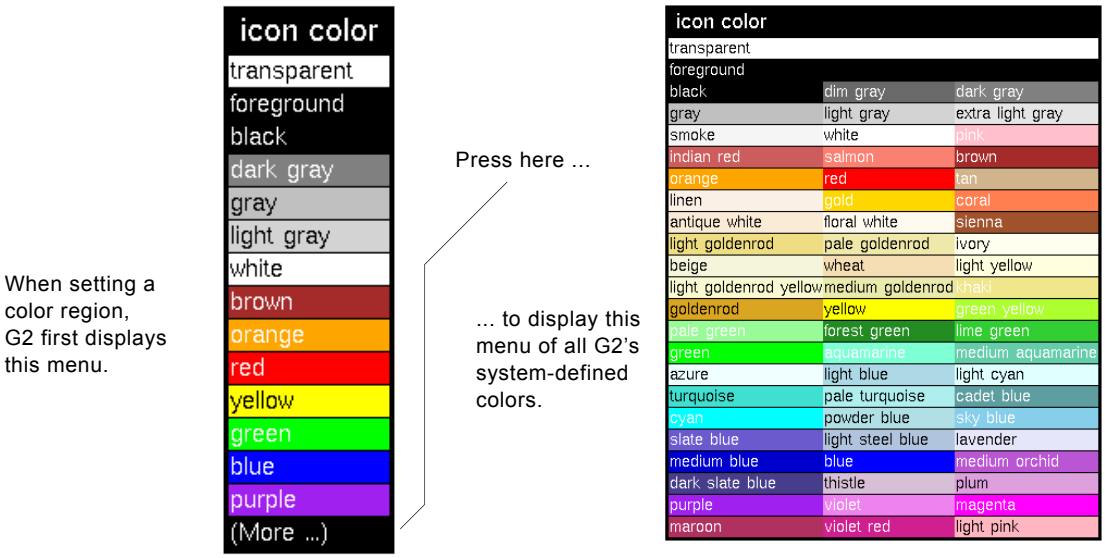
The color menu choice allows you to assign color settings interactively to the color attributes of items.

The color attributes that are settable using the color menu choice depend on the item's representation style, as follows:

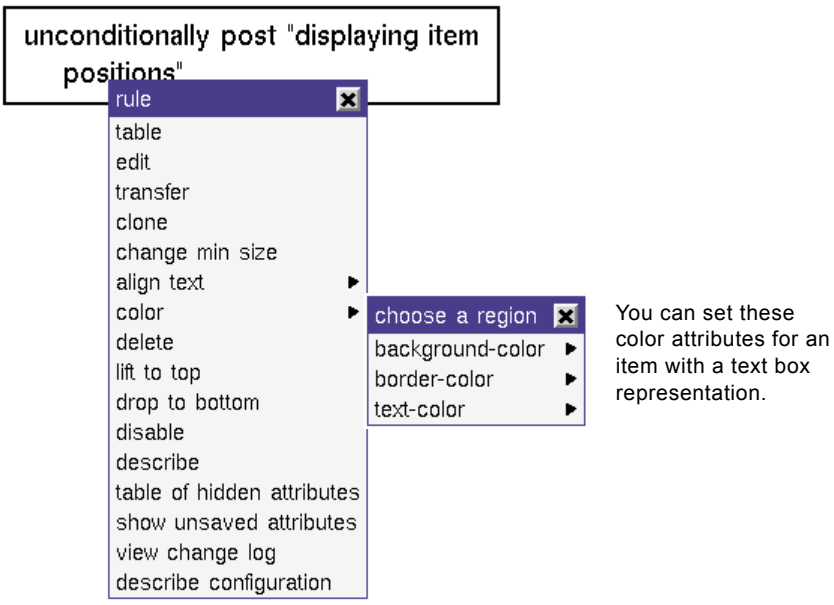
- For items with an icon representation style, you can set the icon-color color attribute.
- For items with a text box representation style, you can set the text-color, border-color, and background-color color attributes.
- For workspaces, you can set the foreground-color and background-color color attributes.
- For all other items, you cannot set a color attribute.

For information about how a set of color attributes corresponds to each item representation style, see Item Representation.

The next figure shows how G2 displays the color palette after you select the color menu choice followed by the icon-color region:



For example, the color choice in the menu for a rule allows you to set the three color attributes for items with a text box representation style:



Deleting an Item

The delete menu choice for an item causes G2 to remove that item from the current KB. Use this menu choice to delete an item, regardless of whether the KB is running, paused, or reset, and regardless of the settings of the item status.

Use the Delete Workspace choice on a workspace's menu to delete the workspace. Remember that deleting a workspace also deletes all items upon the workspace, as well as all items below those items in the KB's workspace hierarchy.

Deleting an item, either interactively or programmatically, can also affect other items:

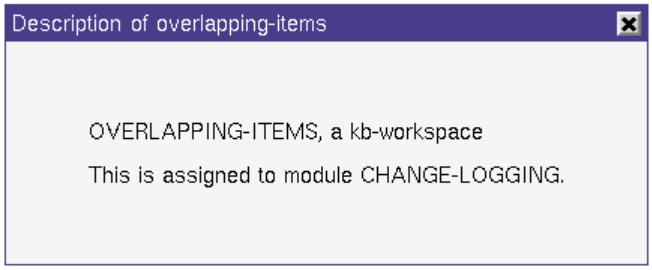
- Deleting an item that participates in a relation also causes G2 to break that relation.
- Deleting an item that is also referenced in an element of a list or array, causes the next reference to that element to produce a *no value* condition.
- Deleting an item that is attached to a connection also causes G2 to delete that connection. If you delete an item attached to a connection that, in turn, attaches to another connection via a junction, G2 deletes only the connection between the deleted item and the junction.

Whether done interactively or programmatically, deleting a permanent item (and any additional changes that this causes for other permanent items) cannot be undone by resetting the current KB. Deleting a permanent item changes the current KB's permanent knowledge. For a description of the KB's permanent knowledge, see Distinguishing Permanent, Transient, and Current Knowledge.

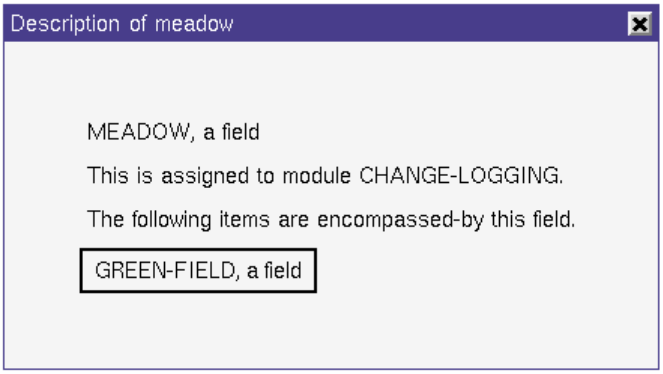
Describing an Item

The describe menu choice for an item directs G2 to display a Describe workspace, which presents information about an item's knowledge. The exact information that the workspace displays depends on the class of the selected item.

For many items, the Describe workspace simply displays the first name listed in the item's names attribute and its module assignment as for this workspace:



For this instance of a user-defined class, G2 also displays a description:



The Describe workspace has its own menu. The text items that appear in a Describe workspace are also selectable and have their own menus.

Tip See the various chapters in this guide to obtain more information about what appears in the Describe workspace for items of a particular system-defined class.

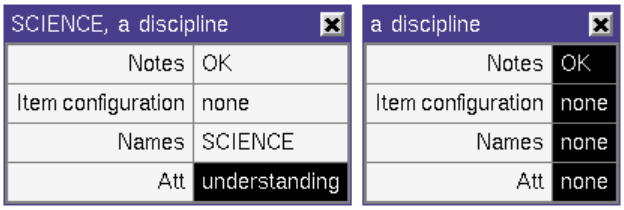
Describing the Configuration of an Item

Selecting the describe configuration choice on an item's menu displays a table that shows the hierarchy of configurations that apply to the item. Configurations and the configurations hierarchy are described in Configurations.

Showing Unsaved Attributes

Selecting the show unsaved attributes choice on an item's menu displays the table for the item with permanently changed attributes highlighted. All attributes are highlighted for a new item. An item that has no unsaved permanent attribute changes has no show unsaved attributes menu option.

The following example shows a an item with one unsaved attribute value highlighted and a newly created item with all attribute values highlighted:



Lifting to the Top and Dropping to the Bottom

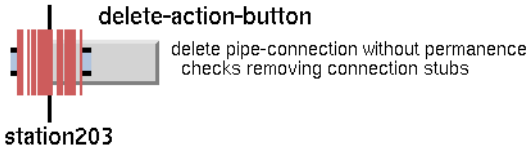
Use the lift to top and drop to bottom menu choices to change an item's layer position, adjusting its relative position, top to bottom, among the items in its workspace.

Selecting lift to top causes the selected item to appear on top of any other item on its workspace. Selecting drop to bottom causes the selected item to appear beneath any other item on its workspace.

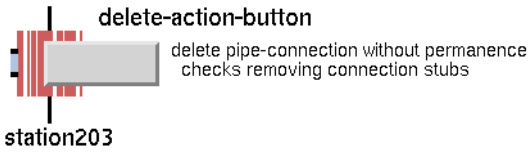
Lift to top and drop to bottom work by revising the layering of items on a workspace. Lift to top makes the selected item's layer position less than the layer positions of all other items on that workspace. Conversely, drop to bottom makes the selected item's layer position greater than the layer positions of all other items on that workspace.

For two items whose representations overlap, the following diagrams shows the result of selecting lift to top and drop to bottom:

In this example, station203 is lifted to the top:



Here, station203 is dropped to the bottom:



Naming an Item

To name an item, select the name menu choice and edit the names attribute. Selecting name causes G2 to open a Text Editor workspace. When you finish editing the name, press Return, or click the mouse on the End button.

If you name an item, try to make the name unique. Though G2 allows duplicate item names, their use is not recommended as a development practice. When duplicate names exist, and more than one of the items satisfies a reference, G2 chooses one of the items at random. The choice may not be the same from one reference to the next, which can cause unpredictable results.

For some items, after you edit the item's name, G2 displays the item's name in a **name box** below the item:

After a name box appears, you can drag it to a new position near the item or anywhere else upon the workspace. You can quickly open the Text Editor and edit the item's name by clicking on the text of the name box.

For debugging purposes, G2 creates a unique, internal name for an item, as needed, for example, when you describe the item. Occasionally, the G2 internal name appears in the notes for the item indicating that its status is OK, for example, METHOD-XXX-BAR::MY-METHOD-13: OK. You can safely ignore these messages. G2 does *not* include them when you use the show on a workspace every item with notes command.

To open the menu for a item's name box:

➔ Click near the corner of the displayed item name.

To edit the name of an item shown in a name box:

➔ Double-click the name box.

or

➔ Open the menu for the name box and select edit from the menu.

To hide the display of a name box:

➔ Open the menu for the name box and select hide name from the menu.

Tip An item's name box is not the same as an attribute display. See Adding Attribute Displays to Attribute Tables.

Showing and Hiding an Item Name Box Programmatically

You can access and manipulate an item's name box programmatically using the attribute access facility. The name-box of *item* is represented by a structure. Here is an example:

```
structure
    (color:
        structure(background-color: the symbol transparent,
            text-color: the symbol foreground),
    position:
        structure(x-offset 238, y-offset: 111))
```

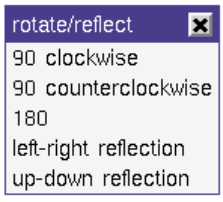
The next procedure shows one way to hide and show the name box of any variable or parameter. This procedure:

- Accepts a single variable or parameter as an argument.
- Creates a new structure of the existing values for the name box, using the structure( ) function.
- Concludes that the name-box of the variable has no value.
- Waits for 5 seconds (used for testing purposes).
- Concludes that the name-box has its original values.

```
show-hide-name-box(art-object: class art)
name-box-value: structure;
begin
    name-box-value = the name-box of art-object;
    conclude that the name-box of art-object has no value;
    wait for 5 seconds;
    conclude that the name-box of art-object = name-box-value
end
```

Rotating and Reflecting an Item

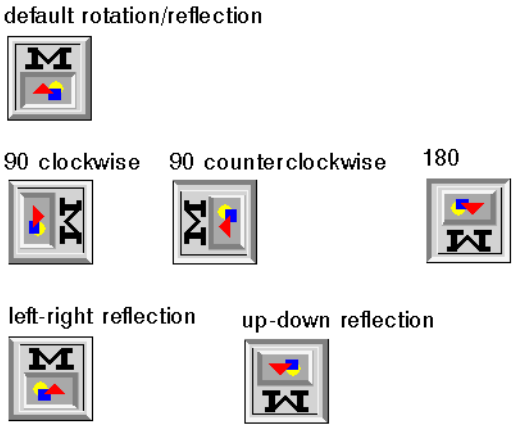
The rotate/reflect menu choice presents this submenu:



Selecting a degree of rotation directs G2 to turn the item's representation on its center, by either 90 degrees or 180 degrees.

Selecting either left-right or up-down reflection directs G2 to display a mirrored version of the item's representation.

This figure shows the effect of rotating and reflecting an item with an icon representation:



Displaying the Tables for an Item

Selecting the table menu choice directs G2 to display the attribute table for this item, and selecting the table of hidden attributes menu choice displays the hidden attributes.

Interacting with attribute tables is explained in Using Attribute-Tables and Hidden-Attributes-Tables.

Transferring Items to Another Workspace

Selecting the transfer menu choice allows you to use the mouse pointer to drag the selected item to any KB workspace that is displayed in the G2 window. If necessary, G2 enlarges the workspace so that it encloses the item's representation within the workspace's current margins.

Item Expressions

These expressions refer to an item or, for the designated generic reference expressions, a set of items.

Referring by Item Name

To refer to an item by name:

➔ *item-name*
-> *item*

In most contexts, specifying a symbol that names an item produces that item, for use by an action or in an expression.

Referring through a Symbolic Expression

To refer to an item by a symbolic expression:

➔ the *class-name* [*local-name*] named by *symbolic-expression*
-> *item*

With the the quantifier, this generic reference expression produces one of the items of the specified class whose item name is produced from *symbolic-expression.* This expression *indirectly* references an item.

With the any quantifier, this expression produces the set of items of the specified class whose item name is produced from *symbolic-expression.*

Referring by Variable or Parameter Name

To refer to an item by a variable or a parameter name:

➔ {*g2-variable* | *g2-parameter*}
-> {*item* | *value* }

In some contexts, specifying a symbol that names a variable or parameter produces its current value, if one exists.

Referring by Workspace Location

To refer to an item by its location upon a workspace:

➔ the *class-name* [*local-name*] upon *kb-workspace*
-> *item*

With the the quantifier, this generic reference expression produces the one and only item of the specified class that G2 finds upon the specified workspace. With

the any quantifier, this expression produces the set of items of the specified class upon the specified workspace.

Referring by Identity

To reference an item by identity:

➔ *item* is [not] the same object as *item*
-> *truth-value*

This expression produces a truth-value that indicates whether an item referenced in one manner is the same item as another item referenced in a different manner. For example:

```
for any tank T that is downstream-of water-outflow-2
    if tank-1 is the same object as T then
        inform the operator that
          "[the public-name of tank-1] is downstream of [the public-name of
          water-outflow-2]."
```

This generic if rule checks whether tank-1 is among the items that participate in a downstream-of relation with the item water-outflow-2.

Referring by Association with an Event or Location

These expressions refer to the item that is associated with an event or to the workspace that contains the item in which the expression is evaluated.

To reference an item through its associated window:

➔ this window
-> *g2-window*

This expression produces the G2 window that receives the user gesture associated with this expression. Specify this expression only in the action attribute of an action button or in the action attribute of a user menu choice.

For example, this action affects the workspace that contains an action button that, when pressed, causes this action's this window expression to be evaluated:

```
hide this workspace on this window
```

To reference an item through its workspace:

➔ this workspace
-> *kb-workspace*

This expression produces the parent workspace of the item within which this expression is evaluated.

Note If the this workspace expression is evaluated as the result of selecting a user menu choice, it produces the workspace that contains the user menu choice, *not* the workspace that contains the selected item.

Referring by Item Evaluation

Several expressions refer to the item associated with the executable item in which the expression is evaluated.

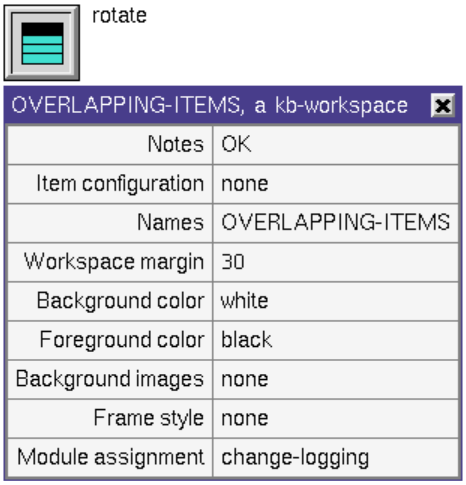
To reference the item from whose menu a user menu choice is selected:

➔ the item
-> *item*

Specify this expression only in the action attribute of a user menu choice, to refer to the item from whose menu this user menu choice is selected.

The the item expression is valid regardless of the class named in the user menu choice's applicable-class attribute.

For example, the next figure shows a user menu choice whose action attribute specifies the action rotate the item by 90 degrees:



Referring to Other Item Knowledge

These expressions refer to other knowledge contained in or associated with an item, but not found in the item's attributes.

Referring to the Name and Class

To refer to the name of an item:

➔ the name [*local-name*] of item
-> *symbol*

With the the quantifier, this generic reference expression produces the first symbol in the names attribute of the specified item. With the any quantifier, this expression produces the set of symbols that name the specified item. For example:

```
if the name of the custom-object O nearest to help-button is custom
    then conclude that the status of O is OK
```

To refer to the class of an item:

➔ the class of *item*
-> *symbol*

This expression produces a symbol that names the class of the specified item. For example:

```
if the class of the custom-object O nearest to help-button is my-object
    then conclude that the status of O is ok
```

To refer to an item that is an instance of a particular class:

➔ *item* is an instance of the class named by (*symbolic-expression*)
-> *truth-value*

This expression produces a truth-value that indicates whether the specified item is an instance of the class (or any of its subclasses) named in the specified symbolic expression. For example:

```
if the custom-object O nearest to help-button is an instance of the class
  named by (the equipment-class-identifier of my-object) then
    conclude that the status of O is ok
```

Referring to the Superior Item

To refer to the item superior to a workspace:

➔ the *class-name* [*local-name*] superior to *kb-workspace*
-> *item*

This expression produces the superior item of the specified class of a workspace that is also a subworkspace. For example:

```
move the item superior to this workspace by (100,100)
```

If you specify a top-level workspace as *kb-workspace*, G2 signals an error. To prevent this, use this expression with the exists expression, as follows:

```
for any kb-workspace W
    if the custom-object O superior to W exists and the name of W is help
        then conclude that the purpose of O is help-schematic
```

This generic if rule identifies each workspace that has a superior item and, for each that does and whose name also is help, sets the purpose attribute of its superior item to the symbol help-schematic.

To refer to the item superior to an object contained in an attribute:

➔ the *class-name* [*local-name*] superior to *item*
-> *item*

For an object that is the attribute value of an item of a user-defined class, this expression produces the item that is its superior item. For example:

```
rotate the item superior to my-custom-object
```

If you specify an object that has no superior item, G2 signals an error. To prevent this, use this expression with the exists expression, as follows:

```
for any custom-object O1
    if the item O2 superior to O1 exists and the name of O2 is help
        then conclude that the status of O2 is ok
```

This generic if rule identifies each custom-object that has superior item and, for each that does and whose name also is help, sets its status attribute to the symbol ok.

Referring to the Workspaces Associated with an Item

Expressions that return the workspace of an item are described in Expressions That Refer to KB Workspaces.

Referring to the Relationships of an Item

Each item contains information about the relationships in which it is participating. Such knowledge is accessible programmatically and on the item's hidden attributes table.

To refer to the relationships of an item:

➔ the relationships of *item-of-interest*
-> *relationships*

Argument	Description
item-of-interest	The item whose relationships you wish to obtain.

Return Value	Description
relationships	A sequence of structures of one or more relation names and the items to which it applies.

Each structure in *relationships* contains these subattributes:

Subattribute	Type	Description
relation-name-reference	symbol	The name of the relation, which can be either the relation-name or the inverse-of-relation of the relation.
relation-is-inverted	truth-value	A value of true indicates that the item participates inversely in a non-symmetric relationship; otherwise the value of this subattribute is false. It is not necessary to include this subattribute when concluding relationships.
related-items	sequence	A sequence of items with the relation-name-reference relationship to the *item-of-interest*.

As an example, in a KB that has two relation definitions, married-to and a-son-of, you could create relations between items with the following conclude actions:

- conclude that bill is married-to edna
- conclude that george is a-son-of edna

Such actions result in the item edna having two relationships with two separate items, bill and george. A reference to:

```
the relationships of edna
```

returns a sequence of structures as follows:

```
sequence
    structure(relation-name-reference: the symbol married-to,
                relation-is-inverted: false,
                related-items: sequence(BILL),
    structure(relation-name-reference: the symbol a-son-of,
                relation-is-inverted: true,
                related-items: sequence(GEORGE)))
```

If an item is not participating in any relationships, G2 returns the empty sequence: sequence().

Referring to the Size of an Item

Note The default borders of a kb-workspace, as well as borders created by using a frame-style-definition are *not* included in the item-width and item-height of a kb-workspace.

To refer to the height of an item:

➔ the item-height of *item*
-> *integer*

This expression produces the height in workspace units of the specified item's representation. You can apply this expression to items with an iconic representation as well as to items with other representations. For a description of the representations of items, see Item Representation.

An item's visible icon might not be as high as the rectangular space provided for it; therefore, the height in workspace units produced by this expression might be greater than the visible icon's height. For example:

```
conclude that the height of my-object = the item-height of my-object
```

To refer to the width of an item:

➔ the item-width of *item*
-> *integer*

This expression produces the width in workspace units of the specified item's representation. You can apply this expression to items with an iconic representation as well as to items with other representations. For a description of the representations of items, see Item Representation.

An item's visible icon might not be as wide as the rectangular space provided for it; therefore, the width in workspace units produced by this expression might be greater than the visible icon's width. For example:

```
conclude that the width of my-object = the item-width of my-object
```

Referring to Degrees of Rotation

To refer to the degrees of rotation of an item's representation:

➔ the icon-heading of *item*
-> *integer*

This expression produces the number of degrees of rotation (0, 90, 180, or 270) for the specified item. For example:

```
conclude that the angle-of-rotation of my-object = the icon-heading of my-object
```

Not all items are rotatable. See the description of the rotate action in rotate.

Referring to the Position of an Item

To refer to the item-x-position of an item:

➔ the item-x-position of *item*
-> *integer*

This expression produces the horizontal offset in workspace units between the center of the specified item's representation and its parent workspace's origin. A positive integer indicates a position to the right of the origin, and a negative integer indicates a position to the left of the origin. For example:

```
conclude that the horizontal-location of my-object = the item-x-position of my-object
```

To refer to the item-y-position of an item:

➔ the item-y-position of *item*
-> *integer*

This expression produces the vertical offset in workspace units between the center of the specified item's representation and its parent workspace's origin. A positive integer indicates a position above the origin, and a negative integer indicates a position below the origin. For example:

```
conclude that the vertical-location of my-object = the item-y-position of my-object
```

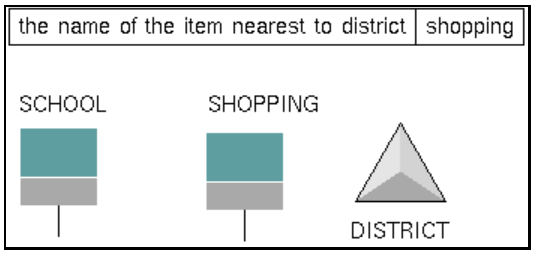
To refer to the item nearest to another item:

➔ the *class-name* [*local-name*] nearest to *item*
-> *item*

With the the quantifier or the any quantifier, this generic reference expression produces the one item of the specified class (or any of its subclasses) that is on the

same workspace as, and the fewest workspace units from, the specified item. The two specified items need not be connected.

G2 calculates the proximity of a pair of items as the distance in workspace units between the *centers* of their respective representations. In the following example, the readout table indicates that shopping is nearest to district.

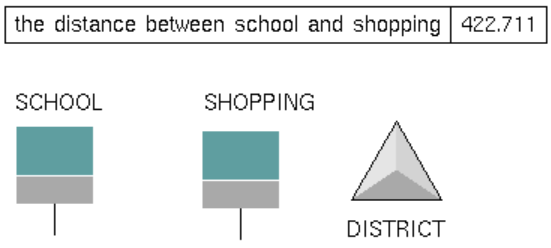


To refer to the distance between items:

➔ the distance between *item* and {*item* | the nearest *class-name*}
-> *integer*

This expression produces the number of workspace units of distance between the centers of the representations of the two specified items. The two items must be upon the same KB workspace, else G2 signals an error. The two items need not be connected.

The next figure shows two readout tables that use this expression to display the distance between items X and Y and between items X and Z.



The Item Class

The item class is the root class of the G2 class hierarchy. For more information about G2's system-defined class hierarchy, see Classes and Class Hierarchy.

The system-defined attributes that G2 provides for all items in a knowledge base are described in the following table. To avoid redundancy, they are omitted from all other attribute descriptions in this manual that pertain to items:

Attribute	Description
notes	Displays the current value of the item participation status: OK/incomplete/bad, active/inactive, and enabled/disabled. Use this attribute's value to identify a discrepancy in the knowledge that an item contains. If there are no problems with the item, G2 displays ok. If there are problems with the value of any item attribute, G2 displays bad. If there are attributes that require values but do not yet have them, G2 displays incomplete.
Allowable values:	ok bad incomplete
Default value:	Varies, depending on the class of the item.
Notes:	You cannot edit this attribute, but you can refer to it programmatically in an expression, and by using the phrase with notes in a filter expression within the Inspect facility. For information on the status of items, see Participation Status.

Attribute	Description
names	Contains one or more names for an item. G2 does not require that this attribute have a value. For example, a short name can identify the item in a schematic, while a long name refer to the item in a rule or other statement.
Allowable values:	Any unreserved symbol
Default value:	none
Notes:	Items of some system-defined classes do not offer a names attribute. See Understanding Item Inheritance. If two connection posts have the same name, G2 considers them to be connected.
item-configuration	Contains one or more configuration statements that apply to this item and to all items below it in the workspace hierarchy.
Allowable values:	See Declaring Configurations for Items.
Default value:	none

System Procedures for Working with Item Groups

Several system procedures let you work with groups of items. The system procedures are the programmatic equivalent of interactively using the operate on area menu choice available on the KB Workspace menu.

The following system procedures allow you to get a group of items and to move, transfer, clone, align, and distribute the items as a group. System procedures also exist for moving, transferring, and cloning a group of items to the mouse.

For more information on all these system procedures, see Move and Transfer Operations in the *G2 System Procedures Reference Manual*.

To create a list of a group of items:

➔ g2-get-items-in-area
(*workspace*: class kb-workspace, *left*: integer, *top*: integer,
right: integer, *bottom*: integer, *items-in-area*: class item-list)

Selects a group of items on a workspace, and appends all but connections and stubs to the existing list *items-in-area*. Connections and stubs are not included because G2 can obtain them automatically given the included items. Other group-movement system procedures require such a list of items as their first argument.

To move a group of items:

➔ g2-move-items
(*item-list*: class item-list, *delta-x*: integer, *delta-y*: integer)

Moves one or more items, including any connections between them and any stubs, to a new location on the current workspace. Connections to items not moved redraw as needed to maintain their attachments to the moved items.

To transfer a group of items from one workspace to another:

➔ g2-transfer-items
(*item-list*: class item-list, *destination*: class kb-workspace,
delta-x: integer, *delta-y*: integer)

Transfers one or more items, including any connections between them and any stubs, from one workspace to another. No connection can exist between an item that is transferred and one that is not. If any such connection exists, G2 signals an error and leaves the workspaces unchanged.

To clone and transfer a group of items from one workspace to another:

➔ g2-clone-and-transfer-items
(*item-list*: class item-list, *destination*: class kb-workspace,
delta-x: integer, *delta-y*: integer)
-> *transferred-items*: class item-list

Clones a group of items, including any connections between them and any stubs, and transfers the cloned items to another workspace. Connections to items not cloned are not transferred.

To align a group of items:

➔ g2-align-items
(*items*: class item-list, *operation*: symbol)

Aligns a group of items. The options for *operation* are: left, right, top, bottom, left/right-center, and top/bottom-center. To specify the slash character in a symbol (/), you must use the @ escape character, for example, left@/right-center.

To distribute a group of items:

➔ g2-distribute-items
(*items*: class item-list, *operation*: symbol)

Uniformly distributes a set of items. The options for *operation* are: horizontally and vertically. At least three items are required. The outermost two items are unchanged, and the remaining inner items are positioned between the outermost items such that the space between any two items is constant.

11

Attributes and Tables

Shows you how to use item attributes and the attribute tables that display them.

Introduction

The most significant part of any item's knowledge is typically contained in its attributes. You display and edit an item's attributes by displaying the attribute tables for the item.

Attribute Contents

Attributes can contain **values**, such as integers, floats, symbols, text strings, truth-values, sequences, and structures. For information on the values that attributes can contain, see Values and Types.

Attributes can also contain instances of user-defined and system-defined classes of objects, for example, instances of variable and parameter classes. An object contained in an attribute of an item is called a **subobject**. For information on accessing the attributes of objects contained in an attribute, see Displaying the Subtable for an Attribute That Contains an Object.

Some attributes contain the text of a text-based item, such as a rule or procedure. These are called **text attributes**. You work with the text attribute of an item just as you would work with other named attributes.

Other item attributes contain actions, statements, or G2 expressions, such as the action attribute of an action button, the text attributes of rules and procedures, and the item-configuration attribute of items. These are called **compiled attributes**. Items that have compiled attributes might have dependencies upon other items that affect their compilation status, as described in Using Compilation Configurations.

Distinguishing System- and User-Defined Attributes

An item's attributes are either *system-defined* or *user-defined*. The G2 system-defined classes define system-defined attributes for items. You cannot change the definitions of any system-defined attributes.

When you create user-defined classes, you can also define user-defined attributes for those classes. Definitions describes how to create user-defined classes and user-defined attributes.

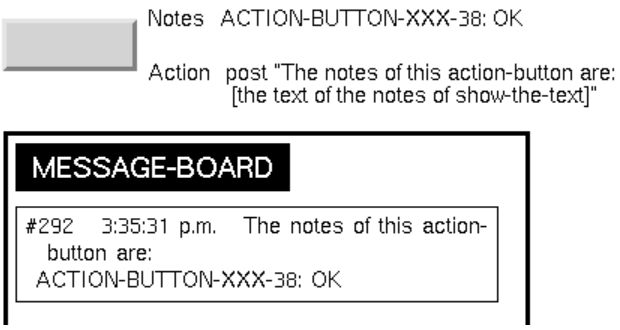
You can access most system-defined attributes of items. Some system-defined attributes, such as notes, are text-readable only. You can refer to the text of such attributes, but not to their values directly. Accessing system-defined attributes is described in Attribute Access Facility.

Accessing Text-Readable Only System-Defined Attributes

To access programmatically the value of a text-readable only system-defined attribute:

➔ Form an expression that refers to the text of that attribute.

For example, the next figure shows an action button. Pressing this action button displays the text of its own notes attribute in a message that G2 posts to the Message Board workspace:



After you create a user-defined class with user-defined attributes, you can interactively edit any user-defined and writable system-defined attribute in each instance of that class. Your KB can also programmatically conclude the value of any user-defined and writable system-defined attribute.

Suppressing Interactive Editing of Editable Attributes

To suppress interactive editing of editable attributes:

➔ Define configurations that direct G2 not to display the edit choice on menus or to respond to mouse clicks that, by default, open the Text Editor.

Declaring configurations is the subject of Configurations.

Using Attribute-Tables and Hidden-Attributes-Tables

An item has two attribute tables, both of which are accessible from the item's menu:

- An item's table menu choice displays the **attribute-table** which displays the user- and system-defined attributes of the item.
- An item's table of hidden attributes menu choice displays the **hidden-attributes-table** which displays the virtual attributes of the item. Virtual attributes give you system information about the item such as its workspace coordinates, its relations, and whether the item is permanent or transient. You

access the value of a virtual attribute and conclude a new value into a writable virtual attribute programmatically and interactively just as you do for other attributes. All user-accessible system-defined attributes are documented in the *G2 Class Reference Manual*.

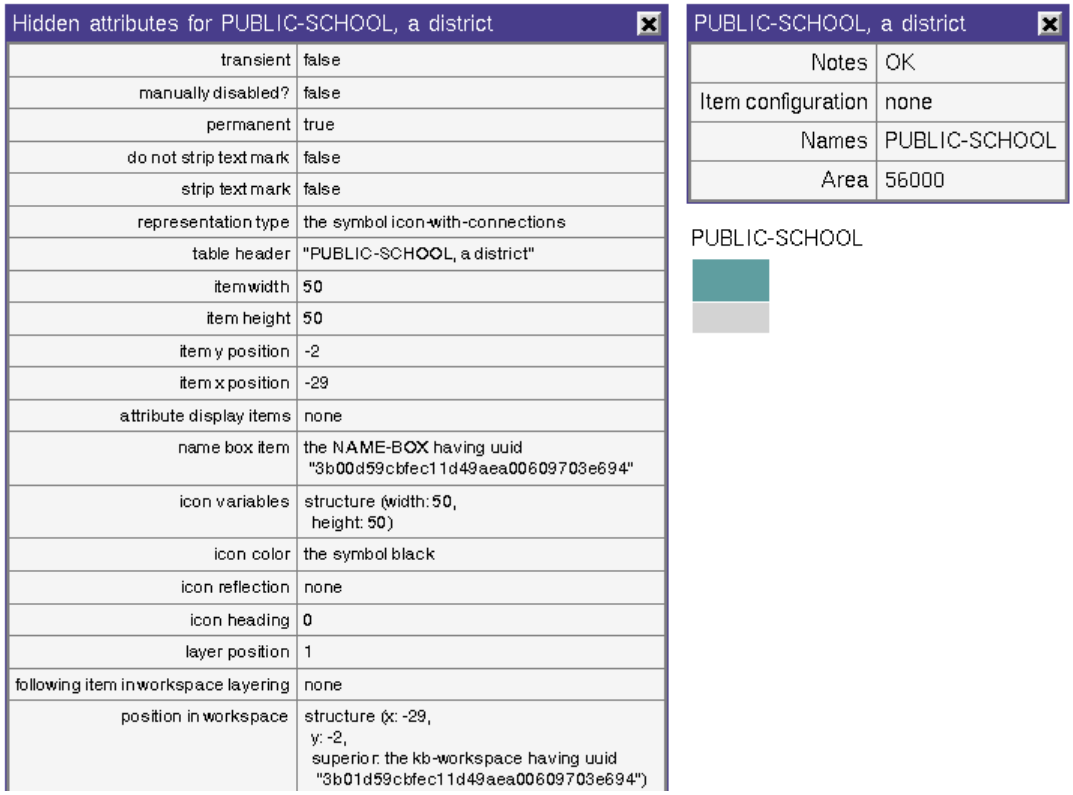
Displaying an Attribute Table for an Item

To display an attribute table for an item:

1. Click the item to display its menu.
2. Choose either the table or the table of hidden attributes menu choice.

Note The ability to display an attribute table assumes that the item has not been configured to behave otherwise. Configurations are described in Configurations.

This figure shows the attributes table and a partial hidden-attributes tables for an instance of a user-defined object class with a single class-specific attribute, area:



Hidden attributes for PUBLIC-SCHOOL, a district

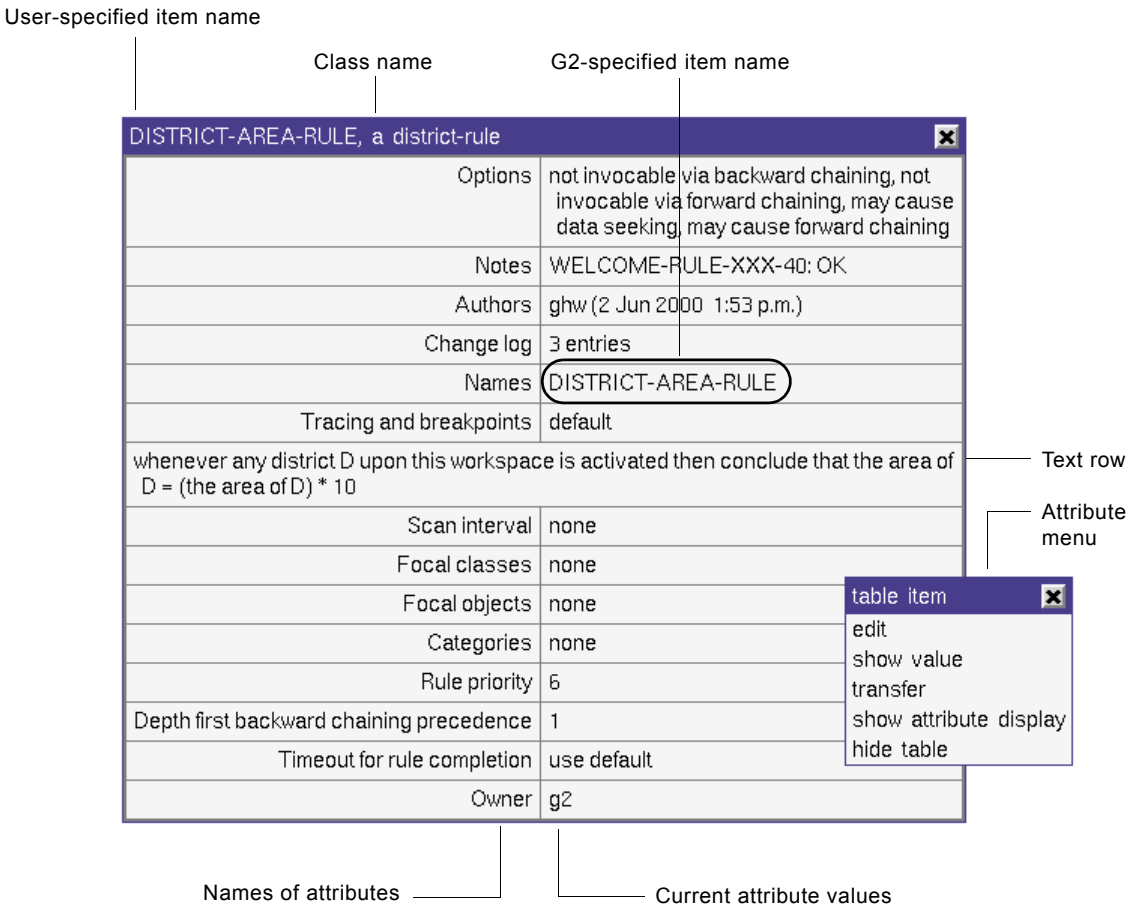
transient	false
manually disabled?	false
permanent	true
do not strip text mark	false
strip text mark	false
representation type	the symbol icon-with-connections
table header	"PUBLIC-SCHOOL, a district"
item width	50
item height	50
item y position	-2
item x position	-29
attribute display items	none
name box item	the NAME-BOX having uuid "3b00d59cbfec11d49aea00609703e694"
icon variables	structure (width: 50, height: 50)
icon color	the symbol black
icon reflection	none
icon heading	0
layer position	1
following item in workspace layering	none
position in workspace	structure (x: -29, y: -2, superior: the kb-workspace having uuid "3b01d59cbfec11d49aea00609703e694")

PUBLIC-SCHOOL, a district

Notes	OK
Item configuration	none
Names	PUBLIC-SCHOOL
Area	56000

PUBLIC-SCHOOL

This figure shows the organization of an attribute table for an instance of a user-defined rule class:



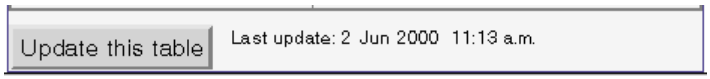
Updating Attribute Tables

You can open more than one attribute table and more than one hidden-attributes table for an item. For all open non-hidden attribute tables, G2 continuously and automatically updates the tables when the values of the displayed attributes change. For this reason, displaying an attribute table is a convenient way to monitor an item's attributes as you develop your KB. However, G2 does not update a hidden-attributes-table unless you explicitly request an update.

To update an item's hidden-attribute-table:

➔ Click the Update this table button at the bottom of the table of hidden attributes.

To the right of this button, G2 displays the time of the last update:



In an attribute table, G2 indicates that a value has expired by displaying the characters ***. Since only the value of a variable can expire, an expired value can occur in:

- A variable's attribute table, when the current value expires.
- The attribute of a user-defined class that is given by a variable.

Display-Precision on Attribute Tables

G2 displays each float value with no more than three digits of precision to the right of the decimal point. Thus, G2 displays the float value 0.333456 as 0.333. If you reenter a value with only three digits of precision, other significant digits of the original number are lost.

To display the float value of an attribute with greater precision, use a readout table whose expression-to-display attribute specifies a formatting expression, such as dd.ddddd.

Positioning Attribute Tables

By default, an attribute table does not reside upon any workspace. However, you can use the mouse to drag an attribute table anywhere within the G2 window. Attribute tables also respond to G2's system-defined keystroke commands for moving workspaces, such as Control + u and Control + d, to move it up and down, respectively. You can also transfer an attribute table to a KB workspace, as described in the next section.

Attribute tables persist after you reset or restart the current KB. An attribute table is not an item, so G2 does not save it when you save the current KB into a KB file.

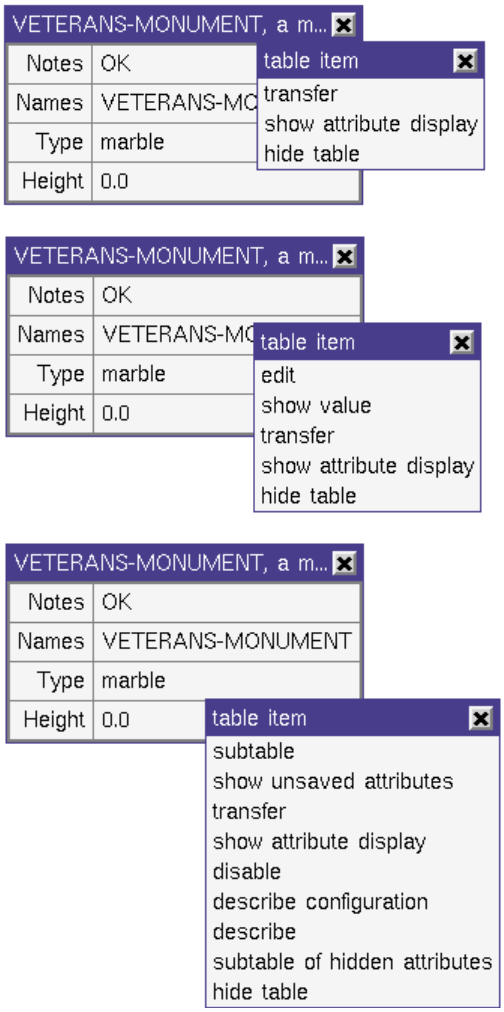
Using Attribute Menus on an Attribute Table

The attribute tables for an item have attribute submenus for most attributes. The menu choices allow you to edit the attribute value, create an attribute display, make a subtable, and perform other functions, depending on the attribute type. For some attributes, clicking on the value of an attribute in a table immediately opens the Text Editor.

To display a table menu:

➔ Click the mouse on any row in the table.

The choices shown on an attribute menu depend upon whether you clicked in a row that displays an editable value attribute, an editable subobject attribute, or a uneditable attribute, as this figure shows:



To edit an attribute value:

➔ Double-click the attribute value.

or

➔ Select the edit menu choice.

This opens the Text Editor. You use the Text Editor to change the value of any editable attribute. For information on interacting with the Text Editor, see The Text Editor.

Certain attribute values are yes or no, in which case you can toggle the value of the attribute.

To toggle the value of a yes/no attribute:

➔ Double-click the attribute value.

or

➔ Select the change to "yes" or change to "no" menu choice, depending on the current value.

Transferring an Attribute Table

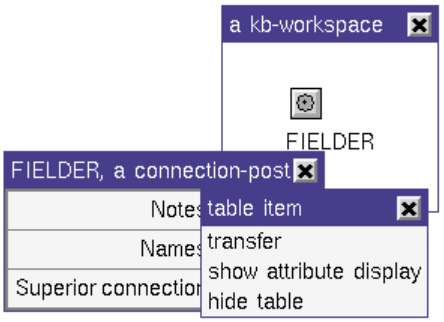
To transfer an attribute table means to place it upon a KB workspace.

A transferred attribute table has a special status within your KB:

- G2 continues to automatically update a transferred non-hidden attribute-table with the current values of the displayed attributes.
- The permanent/transient status of a transferred attribute table remains transient.
- G2 does not save a transferred attribute table into a KB file.

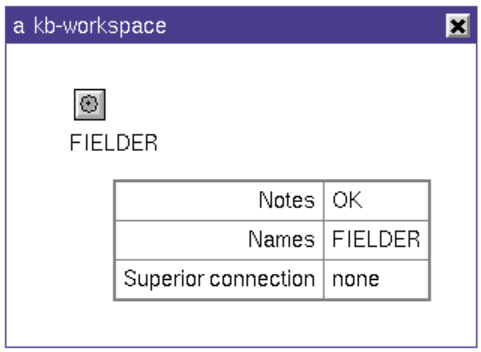
To transfer an entire attribute table to a workspace:

1 Click in one of the cells of the attribute table to display the table-item menu:



2 Choose transfer to attach an outline of the table to the mouse pointer.

3 Drag the attribute table to any displayed workspace, including the workspace on which you are currently displaying the table:



4 Click the mouse to place the table on the workspace. The workspace borders move outward as necessary to include the table. The transferred attribute-table changes to a white-background table without a title bar.

Note You cannot programmatically transfer a table, and you cannot programmatically move a table that is on a workspace.

Displaying the Subtable for an Attribute That Contains an Object

A user-defined attribute of a user-defined item class can contain as its value any instance of the object class.

The object contained in the attribute of an item is called a **subobject**. A subobject has its own table associated with it, called a **subtable**. The table menus and menu choices that appear in a subtable are no different from those for attribute tables. The only difference is that they pertain to the attributes of the object contained in the attribute.

Tip Defining an Attribute as an Object Instance describes how to define class-specific attributes whose values are object instances, including variables, and parameters.

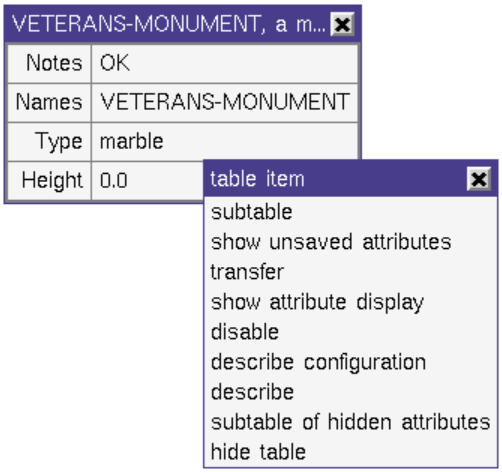
To display the subtable for an attribute that contains an object:

➔ Click on the attribute row in the table to display its menu and select the subtable menu choice.

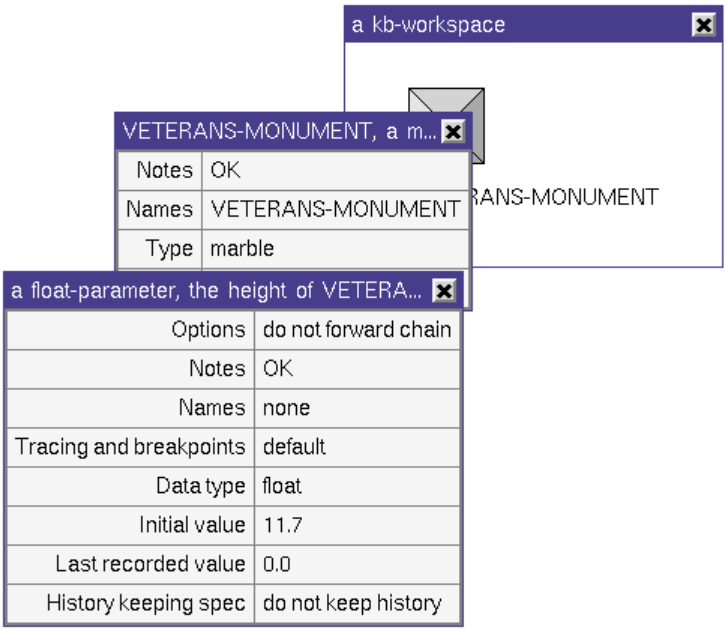
The fact that this table contains the subtable menu choice by default indicates that the value of this attribute is an object.

Note You cannot delete the subtable for a class-specific attribute that is declared to be an instance of some object class, or given by a variable or parameter.

The next figure shows the attribute menu for a suboject attribute:



Selecting the subtable menu choice displays the subtable for an float-parameter, as shown in the next figure:



Creating a Subtable for an Attribute

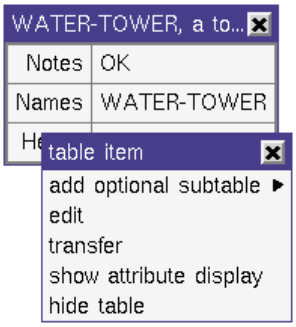
You can interactively create a subtable for an attribute so that it contains an object. The result is similar to defining a class-specific attribute in the definition of a class to be an instance of some object class or given by a variable or parameter.

You can interactively add a subtable to any untyped attribute, or any attribute that has a default value; you cannot interactively add a subtable to a typed

attribute. For information on defining class-specific, typed and untyped attributes in a class definition, see Defining and Initializing Class-Specific Attributes.

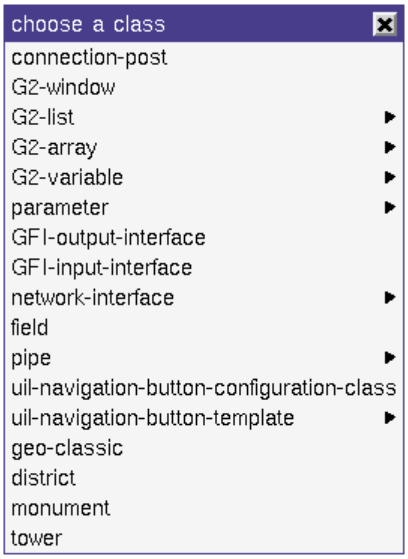
To add a subtable to an attribute interactively:

1 Click on an untyped attribute in an attribute table to display the table menu:



2 Select the add optional subtable menu option.

G2 displays a menu of all system-defined and user-defined object classes:



3 Choose a class.

G2 adds a subobject of the specified class to the attribute and an associated subtable. The table menu for the attribute now displays the subtable menu choice for displaying the interactively created subtable.

To delete an interactively created subtable for an attribute:

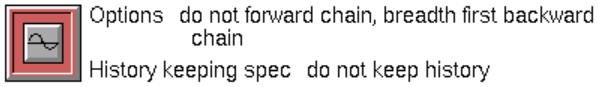
➔ Click on an attribute that contains an object whose subtable was created interactively, and select the delete subtable menu choice.

Adding Attribute Displays to Attribute Tables

An **attribute display** shows the value of an attribute of an item. Using attribute displays, you can display any number of an item's values. You can also optionally display the name of the displayed attribute with its value.

Note You cannot add an attribute display to a hidden attribute table.

This figure shows a variable that has two attribute displays: the options attribute and the history-keeping-spec attribute:



The attribute displays of an item are part of its knowledge. As you move an item within its workspace, its attribute displays move with it. When you transfer an item to another workspace, its displays are transferred with it. G2 saves knowledge about an item's attribute displays in the KB file.

Attribute displays are not items; they cannot be transferred, cloned, and so on. However, since they are part of an item's knowledge, you can access them programmatically using the attribute access facility as described in Adding or Removing Attribute Displays Programmatically.

Attribute displays can only show the value for an attribute that does not contain an item. For example, you cannot display the attributes of a quantitative variable that provides the value of a tank's pressure attribute.

Note An item's name box is not the same as an attribute display of the item's names attribute.

To create an attribute display for an attribute in a table interactively:

➔ Click on the row for any attribute, and select the show attribute display menu choice.

By default, showing an attribute display interactively displays only the value. Showing the attribute name is described in Manipulating an Attribute Display from its Menu.

The text shown in an attribute display appears in the color assigned to the foreground-color color attribute of the item's parent workspace.

You can reposition an attribute display by using the mouse to drag it. Click on the attribute display to edit the text it displays.

Note You cannot create attribute displays for charts, connections, digital clocks, freeform tables, graphs, or readout tables.

Graphs are a superseded capability. For more information see Appendix F, Superseded Practices.

Unlike readout tables, attribute displays do not cause data-seeking. If an attribute's value changes and its value is shown in an attribute display, the display is updated with the new value. In addition, attribute displays are always updated to reflect the current values of the attributes displayed, and cannot be disabled.

Attribute displays behave identically to attributes in an item's table. However, attribute displays place a slight additional burden on your application's performance.

Note If you change the name of an attribute whose value appears in an attribute display, G2 removes the attribute display.

Defining Attribute Displays in Class Definitions

You can also define attribute displays in class definitions so that each instance of the class automatically contains attribute displays.

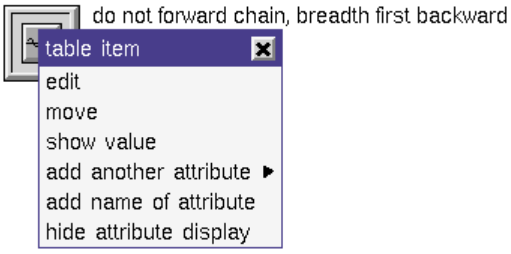
To define attribute displays for instances of a user-defined class:

➔ Add a declaration in the attribute-displays attribute of the class definition.

Setting up the attribute-displays attribute of a class definition is described in Specifying Attribute Displays.

Manipulating an Attribute Display from its Menu

You can manipulate an attribute display by opening its menu:



Note If you click too close to the attribute display value, G2 invokes the Text Editor rather than the attribute display menu.

To add or delete the name of the displayed attribute in the attribute display:

➔ Select the add name of attribute or delete name of attribute menu choice.

To delete an attribute display:

➔ Select the hide attribute display menu choice.

To display another attribute value in an existing attribute display:

➔ Select the add another attribute menu choice, then select another attribute whose value you want to show within the same display.

If you create more than one attribute display for the same item, the attributes you have selected appear as a list, beginning in the standard position for attribute displays (that is, aligned with the top edge of the icon on the right-hand side).

G2 lists the attributes in the order that they appear on the item's attribute table, regardless of what order you use when creating the displays.

Adding or Removing Attribute Displays Programmatically

It is useful to be able to add an attribute display to an item, or remove one or more attribute displays. You can access and manipulate the attribute-displays of items programmatically using the attribute access facility.

To refer to the attribute displays of all items, except those defined on class definitions:

➔ the current-attribute-displays of *item*
-> *display-details*: sequence

The returned sequence consists of one or more structures, each structure representing the attribute display information of a particular attribute as three subattributes:

Subattribute	Type	Description
attribute-list	sequence	A sequence of structures, each with two subattributes: • attribute, a symbol, indicating the attribute name. • display-with-name, a truth-value, indicating whether the name of the attribute is also displayed.
text-color	symbol	The color of the text.
position	structure	A structure consisting of two integer subattributes: • x-offset • y-offset You can also specify the position of an attribute display as the symbol at-standard-position.

To refer to the attribute displays that a class definition defines:

➔ the attribute-displays of *class-definition*
-> *display-details*: sequence

The returned sequence is identical to the sequence returned for the current-attribute-displays of an item.

As an example, the next example illustrates the current-attribute-displays sequence value of a variable auto-count, which has attribute displays for its notes and last-recorded-value attributes.

```
sequence
    (structure
        (attribute-list:
            sequence(structure(attribute: the symbol notes,
                                        display-with-name: false)),
            text-color: the symbol foreground,
            position: structure(x-offset: 29, y-offset: 4)),
    (structure
        (attribute-list:
            sequence(structure(attribute: the symbol last-recorded-value,
                                        display-with-name: false)),
            text-color: the symbol foreground,
            position: structure(x-offset: 29, y-offset: 25)))
```

Example: Adding Attribute Displays to New Objects

In the next example, a whenever rule tracks the creation of instances of hatchback automobile objects, starting the add-displays procedure each time G2 detects a new instance:

```
whenever any instance of hatchback H is created
    then start add-displays(H)
```

To add attribute displays to an item:

1 Conclude a new value into the attribute-displays attribute of an item.

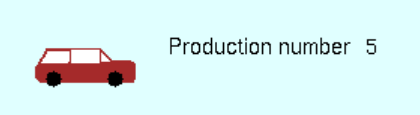
The next procedure increments the value of the parameter called auto-count, concludes that value to an attribute of the new hatchback object, and then displays the value and name of that attribute by setting the subattribute display-with-name to true.

```
add-displays(car: class hatchback)
begin
    {increment parameter tracking total}
    conclude that Auto-Count = Auto-Count + 1;
    conclude that the production-number of car = Auto-Count;
    conclude that the current-attribute-displays of car =
        sequence
            (structure
                (attribute-list:
                    sequence(structure(attribute: the symbol production-
                                                        number,
                                            display-with-name: true)),
            text-color: the symbol foreground,
            position: the symbol AT-STANDARD-POSITION))
end
```

Notice that the procedure specifies the position of the attribute display with:

the symbol at-standard-position

2 The result of showing this attribute display is as follows:



Loading Attribute Values from an Attribute File

Attribute files are a superseded capability. For further information, see Appendix F, Superseded Practices.

Using the Authors Attribute

To support version control for the contents of your KB, system-defined definition classes include the authors attribute. The authors attribute shows the login account name of the person who last edited this item *interactively* and the date and time when the edit occurred.

G2 updates an item's authors attribute by inserting into it the G2 login name and the current time and date of the most recent interactive edit of the item.
If developers do not login to your KB, G2 appends the username associated with each developer's operating-system login account.

You can edit the existing text of an authors attribute, and you can add your own entries.

To update an item's authors attribute:

➔ Set the author-recording-enabled? attribute in the Editor Parameters system table to yes.

This is the default setting. G2 updates the authors attribute only when you *interactively* edit an item of one of these classes.

Tip Because the contents of an item's authors attribute are editable, it might be appropriate to use configurations to restrict access to it by other developers and by your application's end users. Configurations describes how to configure interactive and programmatic access to item attributes.

Using Indexed Attributes

Indexed attributes are a specific kind of attribute that provide an efficient means for G2 to locate a particular item by its attribute value, searching among items of the same user-defined class.

You should declare a user-defined attribute as indexed whenever your application requires an efficient way to reference an item by a particular value for one of its attributes. This is useful when the only alternative is to search a large number of items for the desired attribute value. Techniques for defining indexed attributes appear under Defining an Indexed Attribute.

One example of requiring indexed attributes is an automobile factory that tracks the vehicle identification numbers (VINs) of the vehicles it manufactures. When the KB requires a vehicle with a specific VIN, G2 must search among numerous VIN values to locate the correct object. Using an indexed attribute for the VIN would provide a quick way to locate any automobile item based on its attribute value.

Performance Considerations

For every indexed attribute within a KB, G2 creates and maintains a **hash table**. Each hash table maps the value of an indexed attribute to the item in which it occurs. Hash tables provide rapid search capabilities by using the attribute value as part of a search formula known as the hashing function.

However, the search performance that indexed attributes provide is not without cost. G2 incurs some memory overhead for maintaining the necessary hash tables, especially when the number of values is large. Performance is minimally affected as new indexed values are created and G2 places them into the hash tables.

We recommend that you use indexed attributes sparingly only in situations requiring their capabilities. For accessing smaller numbers of specific values, arrays typically provide adequate results.

Expressions for Indexed Attributes

While any expression can reference an indexed attribute, only three types of references exist for which G2 uses the hash tables for optimal search performance. Other expressions that reference an indexed attribute do not use the optimizations provided for indexed attributes, as follows:

- When using the there exists expression, described in There Exists.
- When using the the count of expression, described in By Iterating Over a Set.
- When using the g2-indexed-attribute-item-list system procedure, described in the *G2 System Procedures Reference Manual*.

For instance, in the next example, G2 would almost directly access the automobile with the correct vehicle-identification-number, if there is one, because vehicle-identification-number is an indexed attribute:

```
if there exists an automobile AM such that
    (the vehicle-identification-number of AM is nc44880100489) then
        post "automobile [the vehicle-identification-number of AM]
                is [the name of AM]"
```

Note Currently, when using an indexed attribute in an expressions with more than one attribute, the indexed attribute should be evaluated first in the expression; otherwise, the expression does not use indexing.

For example, if index-attr is an indexed attribute, this expression uses indexing:

```
if there exists an obj O such that
(the index-attr of O = reference-value and the attr2 of O = ref-value2)
```

whereas, this one does not:

```
if there exists an obj O such that
(the attr2 of O = ref-value-2 and the index-attr of O = reference-value)
```

Using Universal Unique Identifiers

Every item has a UUID attribute that receives a Universal Unique Identifier (UUID) value when the item is created. In previous versions of G2, only items that inherited from unique-identification class had a uuid attribute.

The UUIDs created by G2 conform to the standard OSF/Open DCE UUID format. A UUID incorporates hardware identifiers, creation-time data, and other information that make a UUID unique across KBs created anywhere, at any time.

Within G2, a UUID is stored in a compressed memory-saving format. It is displayed on attribute tables as a text value containing 32 hexadecimal digits. UUIDs are saved with the KB, and are necessary for the successful saving and reloading of a KB.

Uniqueness within a G2 Process

The UUID of every item is unique within a G2 process because:

- G2 generates a UUID in accordance with the DCE UUID format.
- Even though you can edit the value of a UUID, G2 will not accept a new value that does not contain an ordering of 32 hexadecimal digits that is unique within the G2 process. If you enter a value that is a duplicate of the UUID of another item or does not contain 32 hexidecimal digits, the G2 compiler rejects the value.

- When you clone an item that has a UUID, G2 does not clone the UUID; it supplies the clone with a new UUID.
- When you load a KB module, any item without a UUID, or with an invalid UUID, is assigned a new value based on current hardware and time information.

Changing a UUID at Load Time

Having unique UUIDs within a G2 process does not ensure the successful loading of KBs. For saving and reloading items that contain references to items in other modules, G2 relies on UUIDs being unique and stable across all the modules in your KB. Permanent lists, arrays, and the relations of items are examples of KB knowledge that can contain item references across modules.

For example, you might have a permanent g2-list that has an item element, and the list and the item element reside in different modules. G2 saves the UUID of the item element with the list so that the item can be reinserted in the list at load time. If you change the UUID of the item element and save its module independently of the module containing the list, reinsertion will fail when the KB is loaded.

Displaying the UUID of Every Item

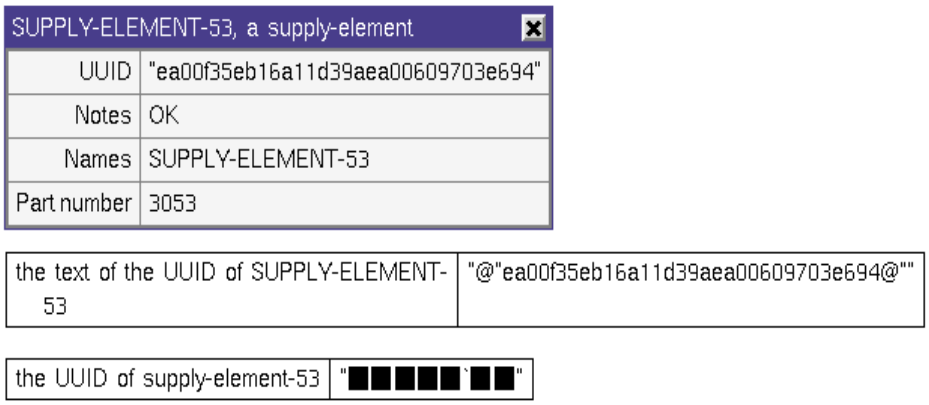
By default, G2 does not display the UUID attribute on the table of an item that does not inherit from unique-identification.

To show the UUID on the attribute table of every item:

1 Choose Main Menu > System Tables > Miscellaneous Parameters.

2 Specify yes for the show-uuids-in-attribute-tables attribute.

The example below shows how G2 displays a UUID on an item attribute table, and how G2 displays the value when it evaluates the text of the uuid of *item* and the uuid of *item* statements. The unreadable blocks are a graphical representation of the internal compressed format of a UUID.



Connections and UUIDs

Not all connections have UUID attributes. That is because connections, by default, are represented by data structures that require less memory than item data structures. Non-item connections are converted to items when you click on them to display their attribute tables and when you show them in the Inspect facility.

Using Other Special-Purpose Attributes

G2 provides other kinds attributes that serve specific purposes:

- Formatting attributes
- Evaluation attributes

Formatting Attributes

Formatting attributes control the visual appearance of an item in charts and free-form tables. For information about declaring formatting attributes for a chart, see Using Chart Annotations. For information about declaring formatting attributes for a freeform table, see Changing Formatting Attributes.

Evaluation Attributes

The evaluation-attributes are a set of two hidden attributes of certain computational items, describing how those items participate in KB processing. For more information, see Hidden Attributes and Evaluation Attributes.

Actions That Affect Attributes

G2 provides the following actions pertaining to attributes.

Changing an Item Name

To change the names attribute of a item and replace any existing values with a transient value:

➔ change the name of *item* to *symbolic-expression*

The permanent value the names attribute had before a change action is reinstated when G2 is reset.

Concluding Attribute Values

To permanently change the value of an attribute:

➔ conclude that the *attribute-name* of *object*
{= *value-expression* | is *symbolic-expression*}

To permanently change the values of user-defined attributes that contain values, variables, or parameters using an indirect reference to the attribute.

➔ conclude that the {*class-name* | *type*}
that is an attribute of *item* named by *symbolic-expression*
{= *value-expression* | is *symbol*}

For more information about using actions, see Actions.

For information on the status of KB knowledge, see Distinguishing Permanent, Transient, and Current Knowledge.

Expressions That Refer to Attributes

These expressions produce values that represent the knowledge contained in the attributes of items.

Referring to Attributes by Name

To refer to attributes of user- or system-defined classes:

➔ the *item-or-value-attribute* [*local-name*] of *item*
-> {*object* | *value*}

Referring to Attributes through a Symbolic Expression

To refer to an attribute using a symbolic expression:

➔ the {*class-name* | *type*} [*local-name*] that is an attribute of *item*
[named by s*ymbolic-expression*]
-> {*object* | *value*}

Indirectly references an attribute of an item using a symbolic expression.

Iterating Over User-Defined Attributes

To iterate over the names of user-defined attributes of an item:

➔ any symbol [*local-name*] that is a user-defined attribute name of *item*
-> *symbol*

This expression returns the symbolic names of user-defined attributes.

This expression converts the specified attribute's value to a text value, then produces that value. You can refer to the text of any system-defined attribute of any item. You can also refer to the text of any user-defined attribute, regardless of the attribute's declared type.

The following rule demonstrates that you can use the the text of expression both to query the value of an attribute and to assign the value of an attribute:

```
for any custom-object O
    if the text of the notes of O /= "OK" then
        change the text of the status of O to "needs-attention"
```

For every custom-object, this generic if rule determines whether the text version of the value of the item's system-defined notes attribute does not contain the text value "OK". For each such custom-object, the rule uses the change the text of action to assign a new value into its status attribute. The change the text of action assigns a transient value to an attribute.

You can also use attribute access to reference and change most attribute values, as described in Attribute Access Facility.

Referring to the Text Attribute of an Item

To refer to the text attribute of an item:

➔ the text [*local-name*] of *item*
-> *text*

Messages and items of certain system-defined classes have a text attribute. The text attribute of an item has no attribute name and contains an editable text value. Classes whose items contain a text attribute are:

procedure
method
rule
free-text
borderless-free-text
message

For example:

```
conclude that the text of my-rule =
    "if true then inform the operator that @"This is a short rule.@""
```

This conclude action sets the text attribute of the rule my-rule. The change is permanent. This is an example of concluding the text of a compiled attribute: in this case, the text of the rule's text attribute. Concluding the text of a compiled attribute causes G2 to recompile the attribute. If G2 detects compilation errors due to recompiling the attribute's text, G2 signals an error.

Referring to an Attribute That is an Instance of an Object

To refer to an attribute that is an instance of an object:

➔ the *item-attribute* [*local-name*] of *item*
    -> *name*

This expression produces the child object that is the value of a user-defined attribute of a parent item of some user-defined class. For example:

```
create a quantitative-parameter by cloning the volume of my-water-tank
```

In this case, the value of the volume attribute of my-water-tank is defined as an instance of a quantitative parameter.

Referring to an Attribute Given by a Variable or Parameter

To refer to an attribute given by a variable or parameter:

➔ the *item-attribute* [*local-name*] of *item*
    -> {*integer* | *float* | *symbol* | *text* | *truth-value*}

This expression refers to a user-defined attribute of an object, connection, or message of a user-defined class. In the class's definition the attribute's value is

declared to be given by a variable or parameter. This expression produces the value given by the associated variable or parameter. For example:

```
change the text of the validity-interval of pipe-scanning-rule to
    the text of the minimum-scan-interval of water-pipe-1
```

In this case, the value of the minimum-scan-interval attribute of the connection water-pipe-1 is given by a float parameter.

You can also use attribute access to reference and change most attribute values, as described in Attribute Access Facility.

Referring to an Untyped Attribute That Contains an Object

To refer to an untyped attribute that contains an object:

➔ the *object-class* that is an attribute of *item* [named by *symbolic-expression*]
    -> *object*

This expression produces the child object contained in an *untyped*, user-defined attribute of an item of a user-defined class.

For example, you might want to refer to the current level of a tank; however, different subclasses of tanks might have differing names for this attribute. You could use the following expression to determine the target attribute name in each case:

```
the custom-quantitative-variable V that is an attribute of water-tank
    named by tank-level-parameter
```

This expression refers to the item of class custom-quantitative-variable that is contained in the attribute whose name is contained in the item named tank-level-parameter, which is a user-defined symbolic parameter that returns the name of the attribute that provides the current level of a water-tank.

Referring Indirectly Using a Symbol

To refer indirectly using a symbol:

➔ any symbol [*local-name*] that is a user-defined attribute name of *item*
    -> *symbol*

This expression produces the names of the user-defined attributes of the specified user-defined item. In combination with indirect attribute references, you can use this expression to access the value of each user-defined attribute.

For example, the unconditionally rule shown next displays the names and values of all the attributes associated with tank-1. (Note that each user-defined attribute of tank-1 is declared as type quantity, which means that each can contain either an integer value or a float value.)

```
for any symbol S that is a user-defined attribute name of tank-1
    unconditionally post
        "The [S] of tank-1: [the quantity that is an attribute of tank-1
            named by S]"
```

Referring to the Parent Attribute Name of a Subobject

To refer to the parent attribute name of an attribute name of a subobject:

➔ the attribute name of *object*
    -> *symbol*

The procedure below iterates over the instances of an object class. When it finds an instance that is the value of an attribute, it displays the parent attribute name and parent item name on the Message Board:

```
declare-subobject-pistons( )
P: piston;
begin
    for P = each piston
        do
            if there exists an item superior to P then post
                "The piston [the name of P] is a subobject
                    of the [the attribute name of P]
                    of [the name of the item superior to P]."
        end
end
```

12

Attribute Access Facility

Presents the capabilities of the attribute access facility.

Introduction

The attribute access facility provides programmatic access to G2 system-defined attributes and the data structures of which they are composed, increasing the level of control that you, as a KB developer, have over item knowledge.

Using the attribute access facility, you can:

- Interactively display the hidden attributes of an item and edit those that are writable. You display the hidden-attributes-table for a item by selecting the table of hidden attributes menu choice from the item's table. For more information, see Using Attribute-Tables and Hidden-Attributes-Tables.
- Conclude values directly into almost all system-defined attributes in exactly the same way as you would use the conclude action to change user-defined attributes. In most cases, attribute access supersedes the use of change the text of statements. Examples are presented throughout this chapter.

- Access internal item knowledge, such as the attribute displays of any item by referring to their values in exactly the same way as you would refer to user-defined attribute values. For an example, see Adding or Removing Attribute Displays Programmatically.
- Access and change an item's name box. For an example, see Showing and Hiding an Item Name Box Programmatically.
- Obtain user-specific information about items, such as which attributes are visible in a particular user mode. For an example of doing this, see Obtaining the Attributes Visible for a User Mode Programmatically.
- Work with history values programmatically. For a description, see Working with History Keeping Using Attribute Access.

Part of using the attribute access facility involves the use and manipulation of structures and sequences. These composite value types, described in detail in Values and Types, provide a flexible, efficient, and convenient way of providing sets of subattributes and ordered lists.

Accessing System-Defined Attributes

The attribute access facility provides an extremely powerful, but optional, set of capabilities. Existing KBs can incorporate these capabilities or continue without using them. Some KB developers may use attribute access to control a minimum set of knowledge and attributes, while others use it to control every programmatically accessible attribute of G2.

The attribute access facility lets you conclude new values into system-defined attributes directly, using standard G2 expressions such as this conclude action:

```
conclude that the class-of-procedure-invocation of insert-field-items =
    the symbol procedure-invocation
```

where the class-of-procedure-invocation refers to a system-defined attribute. You can access most system-defined attributes.

In many cases, accessing system-defined attributes is no different than completing identical tasks on user-defined attributes.

In some cases, however, the internal data structures of system-defined classes consist of multiple embedded sequence and structure values, which require correspondingly complex references to express them. Because of this complexity, we recommend that you experiment gradually with using attribute access.

For example, you can initially use the facility to access and control some key system-defined attributes. Once the grammatical structures for manipulating embedded sequence and structure values become more familiar, you can then extend your use of attribute access to include any number of system-defined attributes that you wish to capture and manipulate.

Attribute Access Terminology

The attribute access facility introduces several terms:

This term...	Refers to...
subattribute	An attribute that is part of another attribute's value. For example, the history-keeping-spec attribute of variables and parameters consists of three subattributes: • maximum-number-of-data-points • maximum-age-between-data-points • minimum-interval-between-data-points
subattribute reference	The expression to reference any subattribute of an attribute, which may itself consist of other subattributes, such as: the minimum-interval-between-data-points of the history-keeping-spec of x
hidden attributes	Attributes inherent within an item, which are interactively accessible from an item's hidden-attributes-table. Examples of hidden attributes are relationships, position-in-workspace, icon-reflection, and transient.
composite attributes	Attributes that appear in an attribute table as one attribute, but which are composed of more than one subattribute.
attribute descriptions	The detailed internal specifications of an attribute. All attribute descriptions appear in the *G2 Class Reference Manual*.

Note Do not confuse composite attributes with composite types. Composite attributes are those that consist of more than one subattribute, while composite types are those whose values can consist of any G2 type, such as structures and sequences described next.

Attribute Descriptions

The ability to access system-defined attributes directly makes it possible to change their values. To do so, however, requires knowledge of the structure of each class attribute, called its **attribute description**.

Attribute descriptions present the internal data structure and type of each accessible attribute in a system-defined class. You need to know this information before accessing and manipulating system-defined attributes.

Obtaining Class Descriptions

The *G2 Class Reference Manual* presents an alphabetical listing of:

- Every system-defined class.
- The attribute descriptions of every accessible attribute.

The information is generated automatically from the internal structures of G2 to ensure its accuracy and validity. Refer to the *G2 Class Reference Manual* to determine the accessibility of any system-defined attribute.

In the class dictionary, all attribute names and their subattributes appear in bold type. Each attribute description includes this information:

This information...	Describes...
attribute-name	A symbol of the attribute name.
text-readable	Truth-value indicating whether it is possible to use the expression: the text of the attribute-name of *item*
text-writable	Truth-value indicating whether it is possible to set the attribute value using the action: change the text of the *attribute-name* of item
value-readable	Truth-value indicating whether it is possible to use the expression: the *attribute-name* of item
value-writable	Truth-value indicating whether it is possible to set the value of the attribute using the action: conclude that the *attribute-name* of item...

This information...	Describes...
is-system-defined	Truth-value indicating whether an attribute is system- or user-defined. This attribute is true for all system-defined classes.
defining-class	A symbol value indicating the class in which the attribute is defined, which could be the current class or a superior class.

You can also obtain the descriptions programmatically from G2, using the system procedure g2-get-attribute-descriptions-of-class.

Differences between the Value and Text of an Attribute

The *G2 Class Reference Manual* describes whether an item attribute is:

- text-readable
- value-readable
- text-writable
- value-writable

The readability and writability of an item determines the way in which you can access a system-defined attribute. For example, if an attribute is defined as only text-readable, you can refer to that attribute using the text of expressions, but you cannot change the attribute, either through a change the text of statement or a conclude that the x of y expression.

This is the way you can refer to or change system-defined attributes:

If an attribute is...	You can...
text-readable	Refer to it as: the text of the *attribute* of *item*
value-readable	Refer to it as: the *attribute* of *item*
text-writable	Change its value using: change the text of the *attribute* of *item*
value-writable	Change its value using: conclude that the *attribute* of *item*

Viewing an Attribute Value or Text

When attributes are both text- and value-readable, you can view them both as a *text* and as a *value*.

Consider the class-specific-attributes attribute of a class definition, which is text- and value-readable and text- and value-writable. If you create a class definition called test-class, which defines two class-specific attributes as follows:

```
attr-1 is an integer, initially is 1;
attr-2 is a text, initially is "two"
```

then returning the *value* of this attribute, using the expression:

```
the class-specific-attributes of test-class
```

would consist of this sequence of structures:

```
sequence
    (structure
        (attribute-name: the symbol attr-1,
        attribute-type-specification: the symbol integer,
        attribute-initial-value: 1,
        attribute-initial-type: the symbol number),
    structure
        (attribute-name: the symbol attr-2,
        attribute-type-specification: the symbol text,
        attribute-initial-value: "two",
        attribute-initial-type: the symbol text)
```

Returning the *text* of this attribute, using the expression:

```
the text of the class-specific-attributes of test-class
```

would consist of this text:

```
attr1 is an integer, initially is 1; attr2 is a text, initially is @"two@"
```

Referencing Limited-Access Attributes

Some system-defined attributes have a limited access to their textual representation and values. Such attributes are those that either have:

- No grammar and thus can only be values, such as an item's name-box.
- No value equivalents because their attributes consist of a deeply complex set of grammar that can be expressed only as text.

All system-defined attributes are minimally readable, and may also be writable. Some attributes are text- or value-readable, or both. A complete list of system-defined attributes with limited access is not given here, because it may change.

However, the next table presents some examples of attributes that have various combinations or text- and value-accessibility:

Attribute	Text-readable	Value-readable	Text-writable	Value-writable
format-of-image	✓			
item-status		✓		
notes	✓	✓		
authors	✓		✓	
change-log	✓		✓	

KB developers can determine the status of each attribute's text-readable, text-writable, value-readable, and value-writable access by referring to the *G2 Class Reference Manual*, or by testing for these attributes programmatically as described next.

Example of Obtaining an Attribute Description

You can use the g2-get-attribute-descriptions-of-class system procedure to return the attribute description of one or more attributes programmatically. You can then test any aspect of the attribute to determine its functionality.

As an example, the next procedure accepts any class and a sequence of attribute names as its arguments, and then tests to see if each attribute is value-writable, indicating that you can conclude a value into it:

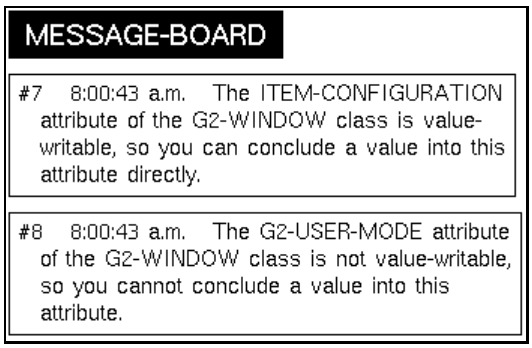
```
get-attribute-informaton(CheckClass: item-or-value, CheckAttributes:
    sequence)
Description: sequence;
Index, Elements: integer;
begin
    Elements = the number of elements in CheckAttributes;
    for Index = 0 to Elements -1
        do
            Description =
                call g2-get-attribute-descriptions-of-class(CheckClass,
                    Check-Attributes);
            if the value-writable of Description[Index] is false
                then post "The [the attribute-name of Description[Index]]
                    attribute of the [CheckClass] class is not value writable,
                    so you cannot conclude a value into this attribute."
                else post "The [the attribute-name of Description[Index]]
                    attribute of the [CheckClass] class is value-writable, so
                    you can conclude a value into this attribute directly."
        end;
end
```

When called with the arguments to get information about two attributes of a g2-window item:

```
start get-attribute-information
    (the symbol g2-window, sequence (the symbol item-configuration,
    the symbol g2-user-mode) )
```

to test whether you can conclude a value into the item-configuration and g2-user-mode attributes of a window item, the procedure determines that the first attribute is value-writable, but that the second attribute is not.

The next figure shows these results:



Thus, you must use change the text of statements to change the g2-user-mode attribute of the current window programmatically.

Hidden Attributes

Hidden attributes are interactively accessible from the table of hidden attributes menu choice of an item. They are also programmatically accessible using the same access grammar as other attributes. All accessible attributes, including **hidden attributes,** are documented in the *G2 Class Reference Manual*.

These are some frequently encountered hidden attributes of items:

Hidden Attribute	Description
evaluation-attributes	Two attributes referring to the computational aspects of G2 items. See Evaluation Attributes.
history	A structure for providing history values and their collection times.
history-using-unix-time	The POSIX-compliant timestamp indicating when each history value is saved.
user-restrictions	A superseded hidden attribute; replaced with item-configuration.

Hidden Attribute	Description
name-box	The structure of an item's name box when it is visible, including its position as an offset to the item.
attribute-displays	The structure for the attribute displays of each item attribute.
icon-variables	A structure of the variables you can define within an icon. Every iconic system- and user-defined class has at least two icon variables: width and height. User-defined classes can include other icon variables. For more information about icon-variables, see The Icon Editor and Icon Management.
relationships	The relations in which an item participates. This attribute returns a sequence of structures, as described in Referring to the Relationships of an Item.

Evaluation Attributes

The evaluation-attributes hidden attribute contains two characteristics of G2 items and how they participate in KB processing. Only one evaluation attribute is actively used.

The evaluation attributes are:

Evaluation Attribute	Description
may-refer-to-inactive-items	A truth-value indicating whether the item can reference items that are disabled or deactivated. This subattribute must be set to true for G2 to be able to invoke the event detection rule: whenever *item* is disabled For more information about this attribute, see Whenever Rules.
may-run-while-inactive	An unsupported truth-value subattribute, for future use.

Composite Attributes

Some attributes are composed of more than one subattribute, which you can refer to individually. The notes attribute is an example of a composite attribute. These are the subattributes of notes:

Attribute	Description
item-status	A symbol used to indicate item status. The value can be ok, incomplete, or bad. You can refer to the value of this attribute, but not its text. You cannot change the text or the value of item-status.
item-notes	The additional part of the item notes attribute, specified as a sequence of text strings, describing notes about an item.

To reference both of its subattribute values requires two expressions:

```
the item-status of x
the item-notes of x
```

where the item-status returns the standard ok, incomplete, or bad status, and item-notes returns a sequence of text values with actual notes.

Referencing System-Defined Attributes

The attribute description of an item describes the internal structure of each attribute, including its value type. The values of many system-defined attributes consist of a specific type, integer, float, text, and so on, while other attribute values consist of more composite types, including one or more structures and sequences. If the value type is of a composite type, it may be further described by one or more boolean operators, which specify which subattributes are required, which must exist together, which are exclusive, and so on. Such complex value types are used exclusively in system-defined class, and do not appear in user-defined attributes.

Typically, system-defined attribute values that consist of a determinate set of subattributes, such as the history-keeping-spec attribute, are provided by a structure, while those comprised of a variable number of subattributes, such as item-configuration, are provided by a sequence.

To change system-defined attributes with a specific type such as integer, float, or symbol, you can use a conclude action in a statement such as this:

```
conclude that the system-defined-attribute of object = new-value
```

A number of G2 items have a system-defined attribute named text. For example, the text of a message, free-text, and procedure are stored in an attribute named text. To change the text of the system-defined text attribute, you cannot use change the text of because the grammar does not support it for an attribute named text. Instead, use this syntax to conclude the value of the attribute named by the symbol text:

conclude that the value that is an attribute of *item* named by *symbol* = "*new-text*"

where *item* is the item whose text attribute you want to change, and *symbol* is a symbolic expression evaluating to the symbol TEXT.

Manipulating the values of system-defined attributes provided by structures and sequences requires an additional reference, because you can refer to the subattribute of an attribute, which itself may consist of other subattributes. Referring to the substructure of attributes is called a **subattribute reference** operation.

A subattribute reference is one that lets you reference multiple levels of subattributes within a single attribute value. Mastering subattribute reference expressions is key to harnessing the power of the attribute access facility to its fullest extent.

Typically, you use subattribute references to change one or more subattributes of an existing structure or sequence. To conclude a set of values into a sequence or structure, use the sequence ( ) and structure ( ) functions.

Creating Subattribute References

One way of creating a subattribute reference expression is to begin with the least-specific reference you can make (the actual item attribute), and work outwards in the expression, towards the most-specific reference (the subattribute you wish to change).

Example of Referring to Class-Specific-Attributes

As an example of creating a subattribute reference in this manner, this section describes how to change a subpart of the class-specific-attributes of a class definition called attribute-initial-value. The class-specific-attributes is represented as a sequence.

To begin, the sample class definition used in this example is called station, and has this specification:

```
height initially is 3;
total-square-footage initially is 15000
```

The subattribute reference in this example changes the attribute-initial-value of the height attribute from 3 to 5.

Displaying Subattributes Values

When creating subattribute references, it is often helpful to see the internal structure of the attribute value that you want to change, especially to locate one or more subattributes. One way to display the internal structure of any system-defined attribute dynamically is to view it in a readout-table, which lets you display any value-readable attribute directly.

You can still display the text of a system-defined attribute by using a Readout Table.

To display the text of a system-defined attribute:

➔ In the expression-to-display attribute of the Readout Table, enter an expression such as:

```
the text of the class-specific-attributes of station
```

This is what displays:

the text of the class-specific-attributes of station	"height initially is 3; total-square-footage initially is 15000"

To display the value of a system-defined attribute:

1 Create a Readout Table item by choosing KB Workspace > New Display > readout-table > readout-table.

2 Edit the table of the Readout Table.

3 In the expression-to-display attribute, enter the item attribute expression for the value to see, for example:

```
the class-specific-attributes of station
```

The attribute value appears when the Readout Display is updated:

the class-specific-attributes of station	sequence (structure (attribute-name: the symbol height, attribute-initial-value: 3), structure (attribute-name: the symbol total-square-footage, attribute-initial-value: 15000))

4 To change a subattribute, first find the subattribute you want to reference. For our example, the subattribute to change is the attribute-initial-value of the defined attribute height.

Referencing a Structure Attribute within a Sequence

The attribute-initial-value subattribute to change is a structure subattribute, within a the main attribute sequence.

To create the subattribute reference:

1 Obtain the type of the system-defined attribute from its attribute description in the *G2 Class Reference Manual*.

The type specification for a class-specific-attributes attribute of a class-definition item is a sequence, and each element of that sequence is a structure value for a defined attribute.

2 Begin the expression by referencing the name of the item attribute to change:

```
the class-specific-attributes of station
```

3 Locate the sequence element that is the value of the height attribute, the first structure in the sequence:

```
(the class-specific-attributes of station)[0]
```

4 Finally, reference the attribute-initial-value subattribute by prefixing it to the expression:

```
the attribute-initial-value of (the class-specific-attributes of station)[0]
```

5 Conclude a new value for the attribute-initial-value of the height attribute:

```
conclude that the attribute-initial-value of
    (the class-specific-attributes of station)[0] = 5
```

Referencing a Sequence within a Sequence

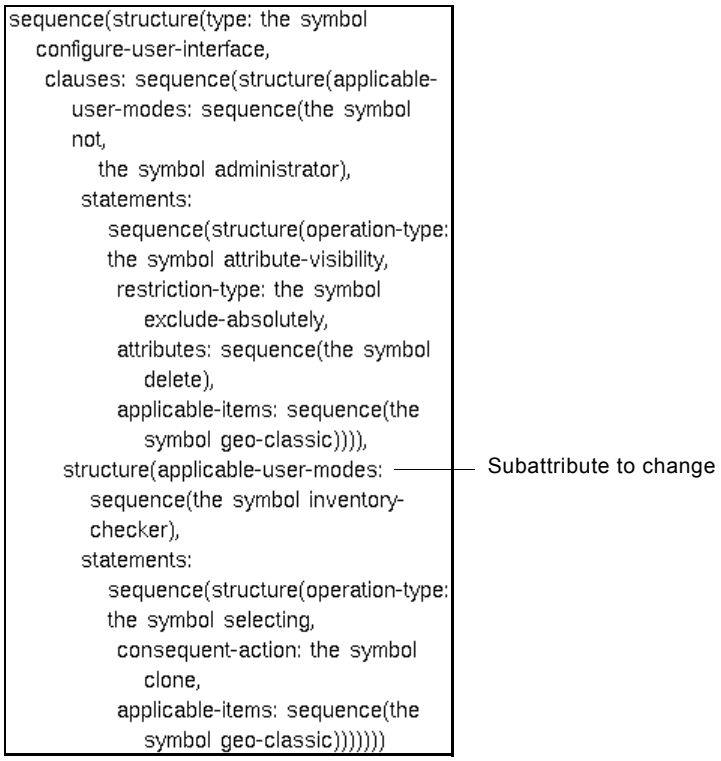
As another example of creating subattribute references, this example changes the subattribute of an instance-configuration. The existing configuration statement is as follows:

```
configure the user interface as follows:
    unless in administrator mode:
        attributes visible for geo-classic exclude absolutely: delete;
    when in inventory-checker mode:
        selecting any geo-classic implies clone
```

The attribute-description for the class-definition class determines that the value of the instance-configuration attribute is a sequence.

A Readout-table with this expression displays the value in the figure:

```
the instance-configuration of geo-classic
```



This subattribute reference will add a user-mode. Notice that the name of the subattribute to change is applicable-user-modes.

To add a new user-mode:

1 Begin with the most general reference:

```
the instance-configuration of geo-classic
```

2 Since the value of this attribute is a sequence, add an element reference to the expression:

```
(the instance-configuration of geo-classic) [0]
```

3 Prefix a reference to the next subattribute, clauses:

```
the clauses of (the instance-configuration of geo-classic) [0]
```

4 Since the clauses subattribute is a sequence, and the subattribute to change is in the second element of that sequence, add a reference to its second element:

```
(the clauses of (the instance-configuration of geo-classic) [0] ) [1]
```

5 Prefix a reference to the subattribute to change (applicable-user-modes), which is a structure, so requires no element specification:

```
the applicable-user-modes of
    (the clauses of
        (the instance-configuration of geo-classic) [0] [1]
```

6 Since the value of the applicable-user-modes subattribute is a sequence, you can conclude a new value into it by using the sequence ( ) function, specifying the user modes you require (inventory-checker and end-user) as follows:

```
add-user-mode(auto: class class-definition)
new-mode: sequence;
begin
    conclude that the applicable-user-modes of
        (the clauses of (the instance-configuration of auto) [0]) [1] =
        sequence(the symbol inventory-checker, the symbol end-user)
end
```

The resulting instance-configuration in the definition is as follows:

```
configure the user-interface as follows:
    unless in administrator mode:
        attributes visible for geo-classic exclude absolutely: delete;
    when in inventory-checker or end-user mode:
        selecting any geo-classic implies clone
```

Referencing Elements in Sequences That Represents a Matrix

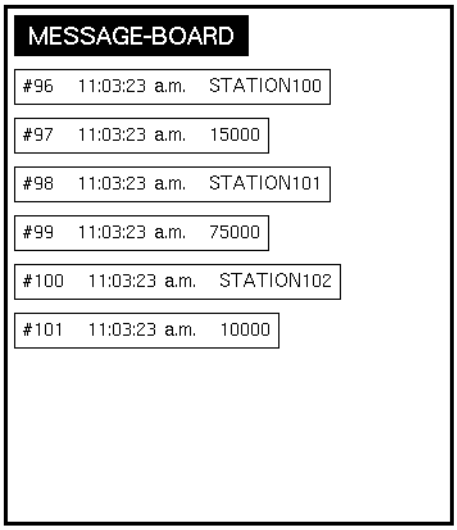
Compound values can be a memory-sparing data-structures for representing matrices. This is an example of sequences that represent a 2-by-3 matrix:

```
sequence(
    sequence(the symbol station100, the symbol station101,
                the symbol station102),
    sequence(15000, 75000, 10000))
```

This example procedure iterates over the "columns" of the value and displays the values on the Message Board:

```
display-sequence-values(sequence-value: sequence)
number-of-columns, column-index, number-of-rows, row-index: integer;
begin
    number-of-columns = the number of elements in sequence-value[0];
    number-of-rows = the number of elements in sequence-value;
    for column-index = 0 to number-of-columns -1
        do
            for row-index = 0 to number-of-rows -1
                do
                    post "[sequence-value[row-index][column-index]];
                end
        end
end
```

These values are displayed on the Message Board:



Tips for Using Subattribute References

Keep these points in mind when working with subattribute references:

- When you return the value of a system-defined attribute, only the subattributes that have a value appear in the sequence or structure. Subattributes that do not have a value are omitted, rather than being returned by name with a value of none.

  The only way to determine definitively all of the subattributes of a system-defined attribute is by checking its attribute description in the *G2 Class Reference Manual*, or by returning that information programmatically using the g2-get-attribute-descriptions-of-class system procedure, as presented in Example of Obtaining an Attribute Description.

- To change the value of a complex attribute consisting of sequences and structures, you must either:
  - Conclude directly into the value, specifying the exact subattribute to change.
  - Provide a new sequence and structure to represent those values. You cannot pass the current attribute value to a local variable, conclude back to that local variable, and affect the original sequence. This is because the sequences and structures returned from an attribute reference are always copies of the value stored in the item.

For example, using this sample code:

```
x: sequence;

x = the sequence-attribute of new-item;
conclude that x = sequence (new-value, new-value);
```

does not change the value of the sequence-attribute of new-item. Instead, it assigns the value of an attribute given by a sequence to the local variable x (a sequence), and then concludes a new value into x.

To conclude into the sequence attribute, use an expression such as:

```
conclude that the sequence-attribute of new-item =
    sequence (new-value, new-value)
```

- When the type attribute of an attribute is given as a construct such as:

  ```
  none |
  structure
  ```

  and is currently none, you cannot conclude a subattribute value into the attribute, without first concluding the entire structure.

  For example, if the history-keeping-spec attribute of a float-variable float-var-1 is do not keep history, which is represented as a value of none internally, an expression such as:

  ```
  conclude that the maximum-number-of-data-points of the
      history-keeping-spec of float-var-1 = 10
  ```

  causes G2 to signal an error.

  The correct way to conclude such a value would be to replace the none value with a structure specifying one or more attributes, using the structure function:

  ```
  conclude that the history-keeping-spec of float-var-1 =
      structure (maximum-number-of-data-points: 10)
  ```

- To change a sequence or structure that currently has values to a value of none, use this expression:

  conclude that *structure-or-sequence* has no value

  For example, if an item X has item-configuration statements that you want to remove, you can do so as follows:

  ```
  conclude that the item-configuration of X has no value
  ```

Concluding Values Directly or Incrementally

When changing the values of system-defined attributes, consider the types of changes to make and the order in which to make them. For example, if you are changing multiple attributes that specify the appearance of an item, making the

changes incrementally results in the item appearing to *jump* through each individual change.

Multiple updates to attributes that specify appearance are usually best done by first creating appropriate structure and/or sequence values, and then concluding those values in a single action to update all of the subattributes at once. Multiple changes to the non-graphical aspects of an item can typically be completed in whatever order the KB requires.

Attribute Access System Procedures

The attribute access system procedures are introspective in that they describe an item's internal structure, including:

- System- and user-defined attributes.
- The default values of each attribute.
- Hidden attributes.

One system procedure returns the current values of an item's attributes.

These are the system procedures to use in conjunction with the attribute access facility:

g2-get-attribute-descriptions-of-class
(*class-name*: symbol, *specific-attributes*: value)
-> *attribute-descriptions*: sequence

Returns a sequence of the attribute descriptions of the given attributes for a class.

g2-get-attributes-visible-in-mode
(*class-or-item*: item-or-value, *user-mode*: symbol)
-> *list-of-attributes*: sequence

Returns a sequence of the attributes that are visible in a particular mode.

g2-get-attribute-values-of-item
(*itm*: class item, *specific-attributes*: value)
-> *attribute-values*: structure

Returns a structure of the current values of the class attributes that you provide.

g2-get-attribute-texts-of-item
(*itm*: class item, *specific-attributes*: value)
-> *attributes-as-text*: structure

Returns a structure of the textual representation of the class attributes that you provide.

13

Classes and Class Hierarchy

Describes the principles, structure, and use of the G2 class hierarchy.

Introduction

G2 knowledge bases (KBs) consist of items that represent or contain knowledge. All items within a KB are based on definitions that are organized into a hierarchy called the **class hierarchy**. This hierarchy is the basis of all knowledge representation in G2.

This chapter describes the structure of the G2 class hierarchy, but does not show you how to extend and use it. Definitions shows you how to extend the G2 class hierarchy and use it to represent knowledge.

- If you are thoroughly familiar with object-oriented programming, you need only skim this chapter to be sure you understand G2's particular approach.
- If you have limited familiarity with object-oriented programming, you should read this chapter as needed to refresh and increase your understanding, and extend it to G2.
- If you are not at all familiar with object-oriented programming, this chapter will probably not provide a sufficient introduction. Please consult a standard text on the subject before you proceed.

This chapter describes multiple inheritance in detail, because it is more complex and less widely understood than single inheritance. You should read the general information about multiple inheritance to determine whether you need to use it, but you can skim or skip over the more detailed information unless you have a specific need for it.

The G2 Class Hierarchy

This section briefly summarizes the G2 class hierarchy, and introduces the terminology G2 uses to describe it. The rest of this chapter builds on and expands the information in this section.

Items and Classes

To represent knowledge in a KB, you need abstractions that represent the things that the KB models. G2 calls these abstractions **items**. Many object systems call such abstractions objects, but in G2 terminology an object is a particular kind of item, as described later in this chapter.

To perform automated reasoning about items, we need a way to categorize them. G2 calls categories of item **classes**. Classes facilitate automated reasoning just as the categories from which they derive facilitate ordinary thought.

G2 implements every class as a set of properties called **attributes** that are common to all members of the class. Examples: name, size, weight, status, color, cost. G2 implements every item as an **instance** of some class. Creating an instance

is called **instantiation**. Every instance can define a particular **value** for each attribute characteristic of its class. Examples: Tank-1, Large, 100 pounds, Repaired, Green, $49.95.

For more information on:

- Items and attributes, see G2 Items.
- Values, see Values and Types.
- Instances and instantiation, see Definitions.

Methods

Ordinary thinking requires understanding the behavior as well as the properties of real things. Similarly, automated reasoning requires procedures that define what items do and how they change. G2 calls such procedures **methods**. Examples: clean, fill, pressurize, sample, empty. For complete information about methods, see Methods.

Inheritance

Just as things can have common properties, allowing them to be grouped into categories, so categories can have common properties, allowing them to be grouped into supercategories. These in turn can be grouped, and so on until the most general possible category is reached.

The grouping of categories into more encompassing categories is called **generalization**. The inverse process, in which categories are subdivided into increasingly specific subcategories, is called **specialization** or **refinement**. These terms are common in object-oriented programming generally; they are not specific to G2.

G2 uses common properties to organize classes into a **class hierarchy**. The highest class is called the **root class**. It has only attributes and methods common to all classes, and is the only class that has no parents.

The root class has **subclasses**, each of which **inherits** those attributes and methods, and defines additional attributes and/or methods specific to its purpose. These subclasses can in turn have lower-level subclasses, with yet more specific attributes and/or methods, and so on to any depth.

The defining of increasingly specific subclasses is called **subclassing**. Attributes and methods inherited by a subclass from a more general class are called **inherited attributes** and **inherited methods**. Additional attributes and methods defined by a subclass are called **class-specific attributes** and **class-specific methods**.

Single Inheritance

When every subclass inherits from only one class, the result is a tree of classes. G2 calls such inheritance **single inheritance**, and describes trees of classes using the usual terminology for hierarchies: root, parent, child, sibling, and leaf. The classes from which a class inherits, directly or indirectly, are its **superior classes**. A class's parent in the hierarchy is its **direct superior class**.

Multiple Inheritance

Single inheritance can represent most knowledge, but not all. For example, mules, the offspring of horses and donkeys, cannot be represented via single inheritance.

To provide for any possible information structure, G2 allows a subclass to have any number of parents (direct superiors). The subclass then inherits all the attributes and methods of all of its parents. G2 calls such inheritance **multiple inheritance**. A class structure that includes multiple inheritance is not a tree, but it remains a hierarchy because it has a unique root class and contains no circular structures.

Linearization

Multiple inheritance entails the possibility of inheriting duplicate or inconsistent attributes and methods. These could introduce ambiguities and conflicts into a subclass's definition. Preventing such problems requires assigning a precedence order to the contributors to a subclass's inheritance, and using it to prevent problems. Establishing such an order is called **linearization**.

Purpose of Inheritance

G2 uses a class's inheritance in three different ways:

- To determine the essential characteristics of the class.
- To determine the correct default value for each attribute of a new instance of the class.
- To determine what methods to use with an instance of the class.

This chapter describes these three uses, and provides pointers to additional information in other chapters.

System-Defined Classes

The preceding overview emphasized the use of items and classes to represent knowledge. However, G2's object system is completely general. G2 uses it to provide all KB components: workspaces, procedures, rules, message boards, charts, buttons, meters, and many other things are all items.

G2 contains a hierarchy of **system-defined classes** that provides all of the classes G2 needs to construct a KB, plus additional classes that you can use to represent knowledge. System-defined classes are inherent in G2. They do not appear as icons on workspaces, and you cannot access, inspect, modify, or delete them.

The root class of the system-defined class hierarchy is named item. This class provides several attributes for all classes, as described in G2 Items and Definitions.

Varieties of System-Defined Classes

System-defined classes have various properties that control how you can use them. These properties determine whether a class can be instantiated, and whether and in what ways it can be inherited by a subclass.

If you know the purpose of a class, you will rarely attempt to use it incorrectly, because the incorrect use would make little sense. Therefore, you don't need to memorize which system-defined classes have which properties. If you do try to use a class incorrectly, the Text Editor will catch and describe the error.

G2 uses various terms to categorize system-defined classes according to their properties. The essential terms are:

- **Extensible class**: Can be inherited by user-defined classes.
- **Reserved class**: Cannot be inherited by user-defined classes.
- **Instantiable class**: Can be instantiated by the user.
- **Noninstantiable class**: Cannot be instantiated by the user.

Terms exist to denote combinations of extensibility, instantiability, and purpose:

- **Concrete class**: An extensible instantiable class.
- **Abstract class**: An extensible noninstantiable class that provides common definitions for use in defining a collection of related of subclasses.
- **Mixin class**: An extensible noninstantiable class that adds specialized capabilities to another class via multiple inheritance.
- **Foundation class**: Generic term for a concrete or abstract class (a foundation class is any extensible system-defined class except a mixin).

Instantiating System-Defined Classes

You can instantiate system-defined classes interactively or programmatically.

To instantiate system-defined classes interactively:

1 Select Main Menu > New Workspace to create an instance of the system-defined class kb-workspace.

2 Select KB Workspace > New *Class-Type* > *system-defined-class*.

For example, KB Workspace > New Button > action-button creates an instance of the system-defined class action-button.

To instantiate system-defined classes programmatically:

➔ create a *class-name*

You can also use create to instantiate user-defined classes, as described in the next section. For information on the create action, see Actions.

Instances of some system-defined classes can appear as icons on workspaces. The icon characteristic of each such class is predefined for the class, and cannot be changed.

Viewing the Class Hierarchy with the Inspect Facility

You can use the Inspect facility to show the entire class hierarchy.

To view the entire class hierarchy:

➔ show on a workspace the class hierarchy

The class hierarchy that Inspect displays includes all system-defined classes, and any user-defined classes as well. The names of user-defined classes appear inside boxes. The figure at the end of this chapter shows the complete system-defined class hierarchy as displayed by Inspect.

Note that Inspect shows class hierarchies from left to right, so that the root class of the displayed hierarchy is at the left rather than at the top.

You can also use Inspect to examine the inheritance of any class.

To view the class hierarchy of a particular class:

➔ show on a workspace the class hierarchy of *class*

where *class* is the name of the class you want to examine. Inspect displays that part of the class hierarchy that relates to *class*.

For further information on using Inspect, see The Inspect Facility.

User-Defined Classes

You can create user-defined classes for two purposes:

- To tune and/or extend the machinery that G2 provides.
- To represent specific types of knowledge within a KB.

The two purposes overlap, because much of G2's machinery exists to represent knowledge in some way. The techniques for creating subclasses are the same in either case.

A class that you define to extend G2 or represent knowledge is called a **user-defined class**. Not all KBs require user-defined classes to create customized KB components, but essentially all KBs require such classes to represent knowledge. Techniques for creating user-defined classes appear in Definitions.

Extending G2's Machinery with User-Defined Classes

G2 provides a wide variety of components for constructing and accessing KBs, such as workspaces, streams, buttons, and tables. These constitute G2's essential machinery. All are provided as system-defined classes. Examples: kb-workspace, g2-stream, action-button, and system-table.

None of these components needs to be customized in order to provide effective G2 machinery. Nevertheless, almost all of them can be. You can therefore extend G2 in almost any way to provide whatever environment you need. Only extensions that would break G2 are precluded.

Representing Knowledge with User-Defined Classes

Some KB knowledge comes in standardized forms, such as connections, relations, methods, and rules. G2 represents such knowledge using system-defined classes. Examples: connection, relation, method, rule. You can use these classes as G2 provides them, or subclass them as needed to provide specialized or extended knowledge-representation capabilities.

Other KB knowledge takes arbitrary forms that cannot be anticipated by predefined system classes. To represent such knowledge, you can define classes that have any desired attributes, and use them as needed to represent knowledge.

Creating User-Defined Classes

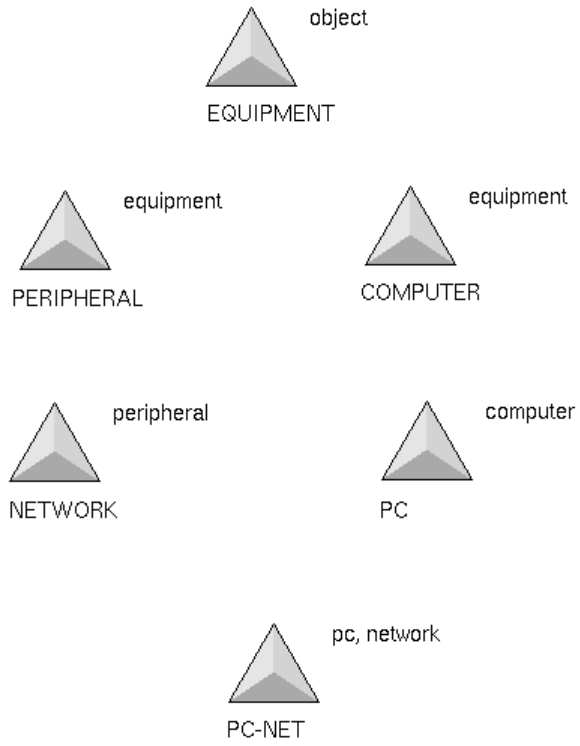
Every user-defined class is based directly or indirectly on one or more system-defined classes, from which it inherits various attributes. Such attributes are called **system-defined attributes**. Most user-defined classes include additional

attributes specific to the class. Such attributes are called **user-defined attributes**. Many user-defined classes also have associated methods.

To create a user-defined subclass, you create a class-definition using the New Definition command of the KB-Workspace menu, then fill in the definition's table to give the new class the desired name and inheritance, as described in Definitions. A user-defined class's inheritance consists of one or more system-defined classes, and optionally one or more user-defined classes.

User-defined class-definitions appear as icons on workspaces. A class-definition's icon is a shaded triangle.

For example, the following figure shows a workspace with seven class-definitions. In this user-defined hierarchy, equipment is at the top of the hierarchy, and pc-net is at the bottom. The direct-superior-classes attribute displays at the left of each class-definition show how the classes inherit from one another in this and the following examples.



The table of a typical user-defined class, specifically the class pc-net in the above figure, looks like this:

PC-NET, a class-definition	
Notes	OK
Authors	ghw (5 Jun 2000 9:37 a.m.)
Change log	0 entries
Item configuration	none
Class name	pc-net
Direct superior classes	pc, network
Class specific attributes	none
Instance configuration	none
Change	none
Instantiate	yes
Include in menus	yes
Class inheritance path	pc-net, pc, computer, network, peripheral, equipment, object, item
Inherited attributes	use; location; network::location; network::use; capital-expense
Initializable system attributes	attribute-displays, stubs
Attribute initializations	none
Icon description	inherited

Note in particular the definition's direct-superior-classes and class-inheritance-path attributes. These attributes are the key to understanding and using G2 class inheritance, as described later in this chapter.

Instantiating User-Defined Classes

You can instantiate user-defined classes interactively and programmatically.

To instantiate a user-defined class interactively:

➔ Select the create instance menu choice of the class definition.

To instantiate a user-defined class programmatically:

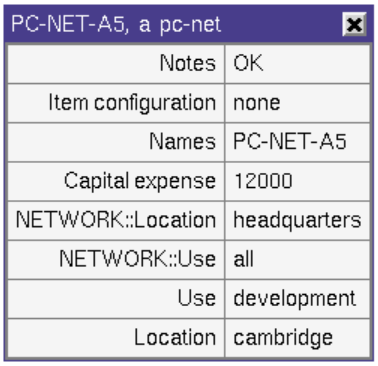
➔ create a *class-name*

You can also use create to instantiate system-defined classes, as described in the previous section. For information on the create action, see Actions.

Instances of user-defined classes can appear as icons on workspaces. The icon characteristic of each class can be inherited from a superior class, or the user-

defined class can specify an icon of its own. The Icon Editor and Icon Management shows you how to customize a user-defined class's icon.

The table of an instance of pc-net, the class whose table appears in the previous figure, looks like this:



Inheritance in Class Hierarchies

Classes are bound into hierarchies by inheritance. If you understand how G2 handles inheritance, you will be able to design class hierarchies to suit any need.

Single inheritance is very straightforward: a subclass inherits everything characteristic of its direct superior class (parent). If you do not redefine any system-defined attributes in the subclass definition, or add any class-specific attributes to the definition, an instance of the subclass differs only in name from an instance of its parent class.

Single inheritance is convenient where information is tree-structured, or sufficiently close to it that an acceptable amount of duplication in class definitions can make up the difference. Single inheritance can usually provide everything that a user-defined class hierarchy needs.

Where information is not tree-structured, you can use multiple inheritance to represent the information efficiently. Multiple inheritance differs from single inheritance in that it allows a subclass to have more than one direct superior class.

Various complexities can arise in multiple inheritance, because the parent classes can include duplicate and conflicting attributes. If you understand how G2 copes with these, as described later in this chapter, you should have no difficulty using multiple inheritance.

Every definition has two attributes for controlling and displaying inheritance: direct-superior-classes and class-inheritance-path. You must understand these attributes in order to understand G2 inheritance.

Direct-Superior-Classes Attribute

This attribute of a class lists the class's parent(s). In single inheritance, only one class appears: the **direct superior class**. In multiple inheritance, more than one class appears. The first class named in the attribute is called the **primary direct superior**, and all subsequent classes are **secondary direct superiors**.

When more than one direct superior class exists, the order in which they appear in the direct-superior-classes attribute, in conjunction with the structure of the existing class hierarchy, determines the class inheritance path, which controls what happens in case of duplicate or conflicting inheritance.

Class-Inheritance-Path Attribute

This attribute of a class lists all classes that contribute to the class's inheritance, in precedence order. The list begins with the class itself: the **immediate class**. The list ends with the root of the class hierarchy, which is always item. The contents of the list between the immediate class and the root class vary with the details of the immediate class's inheritance.

G2 automatically derives a class's inheritance path based on its direct superior class(es) and the existing class hierarchy, and updates the path as needed when you modify the hierarchy.

Class inheritance paths are the key to working with class hierarchies. Almost everything discussed in this chapter either explains how G2 generates class hierarchy paths, or shows how G2 uses such paths to determine class properties.

G2 uses a class's inheritance path to resolve duplicate and conflicting attributes among the superior classes that the class inherits. Such resolution is exactly the same in single and multiple inheritance: G2 scans the class inheritance path and gives precedence to the first relevant attribute definition that it encounters.

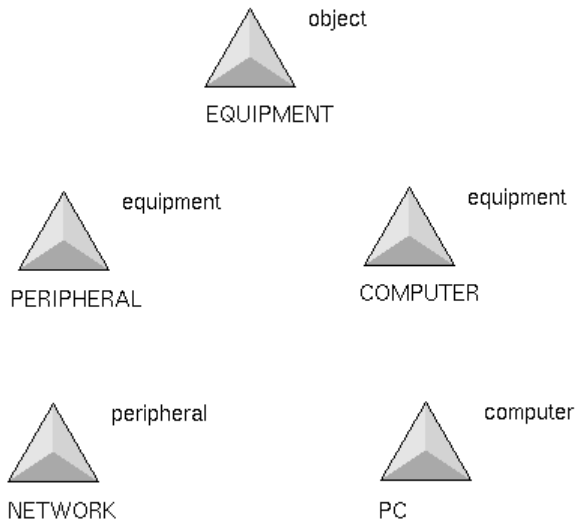
Since the immediate class is always the first class on its own inheritance path, this technique allows a class to **override**, or **shadow**, any inherited attribute definition with a definition of its own.

Single Inheritance

A **single inheritance class** is a class that has a unique inheritance path leading from itself to the root class, because neither the class nor any of its ancestors has more than one parent.

For a single inheritance class, there is only one reasonable way to order the class inheritance path: it lists the class itself, its parent, it's parent's parent, and so on, culminating in the root class.

For example, consider first the following single inheritance class hierarchy:



The figure shows several classes in a hierarchy based on equipment. The direct-superior-classes attribute displays show the class inheritance. The structure of the hierarchy is:

- equipment is a subclass of object, that is, it specifies object as its direct superior class.
- peripheral and computer are two subclasses of equipment, which each specifies equipment as its direct superior class.
- network and pc are subclasses of peripheral and computer, respectively.

Following the rule for single inheritance given above:

- The class inheritance path of network is:

  network, peripheral, equipment, object, item

- The class inheritance path of pc is:

  pc, computer, equipment, object, item

Inheritance of Default Values

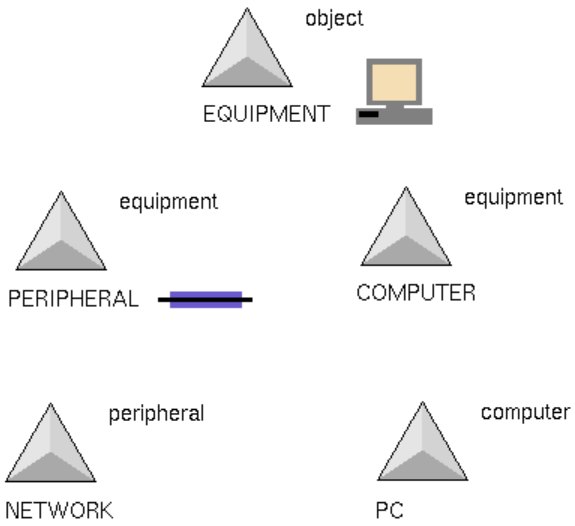
When you create an instance of a class, some system-defined attributes and any user-defined attribute can receive an initial value. G2 calls such a value the **default value** of the attribute. This value is derived from the definition of the instantiated class.

When you reset a KB, attribute values do not return to their default values: they keep the values that they had prior to the reset. This behavior distinguishes default values of attributes from initial values of variables and parameters, which are restored each time a KB is reset.

Caution Be careful not to confuse a *default value*, as described in this section, with a *default attribute*, as described in the next section.

For every attribute, a class can inherit a default value from a superior class, or it can specify a default value of its own, overriding any inherited value. When a class does not explicitly define a default value, a new instance of the class has the value specified by the first class in the class's inheritance path that defines a default value for the attribute.

For example, consider the following hierarchy:



The figure shows the same classes that we looked at earlier. Two of the classes, equipment and peripheral, explicitly define icons. A class instance appears by each class definition that defines a class icon.

An item's icon is specified by the value of the icon-description attribute of its class description or the icon-description of its closest superior on its class inheritance path. This is a system-defined attribute that has a default value provided by G2. A user-defined class can override this default by specifying its own value for icon-description, as equipment and peripheral both do.

With the icon-description values shown, an instance of equipment has a workstation icon, because the class defines the value for the attribute. An instance of computer has the same icon, because computer inherits the default value from equipment. The class pc similarly inherits that default value from computer, so a pc instance also has the workstation icon.

However, peripheral explicitly specifies a different icon-description value: the node icon. This definition shadows equipment's icon description, and is inherited

by network. Thus instances of both peripheral and network have node icons, not workstation icons.

Inheriting Default Values for Stubs

Icons and stubs are closely related, but they are specified by two different attributes of a definition. This independence could result in mismatched icons and stubs, so the G2 class inheritance rules contain a special provision that prevents it.

To prevent such a mismatch, a class can inherit a stubs definition only from the class from which it inherits its icon definition, or from a descendent of that class: classes superior to the class that supplies the icon definition are disallowed. If no acceptable superior class provides a stubs definition, the class's stubs default value is none.

Inheritance of Methods

A given class can have any number of associated methods that define the behavior of instances of the class. These methods can take arguments that tailor the behavior to suit a particular class of instances. Methods defined for different classes can have the same name.

When you call a method on an instance of a class, G2 searches the class-inheritance-path of that class and uses the first explicitly defined method it encounters that has the same number of arguments as the calling expression. If no match is found with any class in the class inheritance path, G2 signals an error.

For more information about methods and their use of the class inheritance path, see Methods.

Duplicate Attributes

To facilitate modular design and encapsulation, G2 allows a subclass to define an attribute whose name duplicates that of an inherited user-defined attribute. Instances of the subclass then have both the inherited attribute and the locally defined attribute. G2 does not permit a user-defined attribute to duplicate the name of a system-defined attribute.

Attributes with the same name are customarily called **duplicate attributes**, though actually only their unqualified names are duplicates. Duplicate attributes have no more relation to one another than any two attributes have. They may or may not hold data of the same type, or share any other characteristics.

If you change a duplicate attribute's definition to eliminate the duplication, nothing changes but the name; the attributes themselves are unaffected. The same applies if you change a name in a way that creates a duplication. In either case, G2 updates all instances and tables as needed to reflect the change.

Duplicate attributes can lead to confusion, and complicate the task of referring to attributes in expressions. Use them sparingly if at all, and only where they provide a significant advantage.

Naming Duplicate Attributes

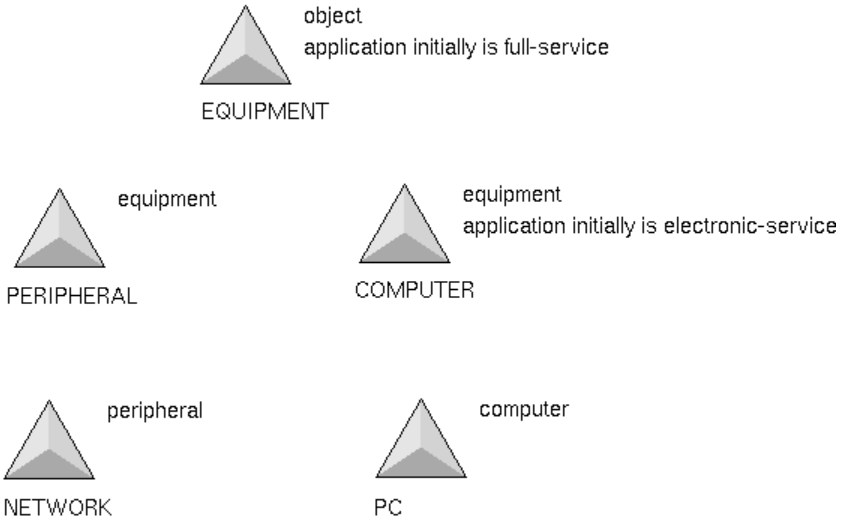
To prevent ambiguity, G2 qualifies the names of duplicate attributes as needed with the name of the class that defines each attribute. The syntax for a qualified attribute name is:

class-name::*attribute-name*

For every class, G2 determines what duplicate attribute name(s) to qualify by following the class's inheritance path. G2 uses without qualification the first attribute with a given name that it encounters. This attribute is called the **default attribute**. G2 qualifies each subsequently defined duplicate attribute with the name of the class that defines it.

Caution Be careful not to confuse a *default attribute*, as described in this section, with a *default value*, as described in the previous section.

For example, consider the following hierarchy where the attribute displays show the direct-superior-classes attribute value for each class and the value of the class-specific-attributes for two classes:



Both equipment and computer explicitly define an attribute named application. The table of a computer instance would show two attributes: application, representing the locally defined attribute, and equipment::application, representing the inherited attribute.

G2 lists attributes from the bottom up in the order in which their definitions appear on the class's inheritance path, so the table looks like this:

COMPUTER-45, a computer	
Notes	OK
Item configuration	none
Names	COMPUTER-45
EQUIPMENT::Application	full-service
Application	electronic-service

An instance of pc has the same application attributes as an instance of computer, because the pc class inherits them from the computer class:

BTT-PC, a pc	
Notes	OK
Item configuration	none
Names	BTT-PC
EQUIPMENT::Application	full-service
Application	electronic-service

In the preceding two tables, the default application attribute is the same for both computer and pc: it is pc's attribute. The reason is that, for both computer and pc, computer is the first class on the class's inheritance path to define an application attribute.

However, if pc also explicitly defined an application attribute, an instance of pc would have three such attributes: application, computer::application, and equipment::application:

a pc	
Notes	OK
Item configuration	none
Names	none
EQUIPMENT::Application	full-service
COMPUTER::Application	electronic-service
Application	local

The attribute now named computer::application is the same attribute that was named application before pc defined its own application attribute. It is no longer pc's default application attribute, because pc's own application definition is now the first such definition by a class on pc's inheritance path.

Referencing Duplicate Attributes

When you reference one of a set of duplicate attributes, you must qualify the reference as needed to specify the correct attribute. The name for any attribute of an item appears with the necessary qualification in the item's table.

If you do not qualify the name in a reference to a duplicate attribute, G2 uses the default attribute of the class of the item. For example, if pc-1 is an instance of pc in the equipment hierarchy pictured above, then:

```
the application of pc-1
```

refers to the application attribute that pc inherits from computer, and:

```
the equipment::application of pc-1
```

refers to the application attribute that pc inherits from equipment.

The meaning of an unqualified name can change if the hierarchy changes. For example, if pc were to define its own application attribute, then:

```
the application of pc-1
```

would thereafter refer to the application attribute that pc itself defines. To reference the computer's application attribute, you would need a qualified name:

```
the computer::application of pc-1
```

To guard against such changes in meaning, you can qualify a reference to a default attribute for a class. Such qualification, though initially redundant, ensures that your code will mean the same thing even if subsequent changes give the class a different default attribute. For example:

```
the computer::application of pc-1
```

means the same thing whether or not pc defines an application attribute of its own.

When you give a qualified name in a reference, G2 always does the same thing: it uses the default attribute of the class named in the reference. Thus a qualified reference need not name the class that actually defines the attribute.

For example, in the hierarchy shown previously, the names network::application, peripheral::application, and equipment::application all refer to equipment's application attribute, because that is the default application attribute for all three classes.

Duplicate Attributes and Default Values

When duplicate attributes exist, the default values for each attribute are completely independent of one another. They have no more relationship than they would if each attribute had a different name.

Be careful not to define a duplicate attribute when all you want to do is shadow an inherited attribute's default value. You can define a new default value for any

attribute in any class definition, as described under Specifying Default Values for Inherited Attributes.

Multiple Inheritance

A **multiple inheritance class** is a class that has, or inherits any class that has, more than one parent. Such a class does not inherently have a unique inheritance path leading from itself to the root class. Multiple inheritance is appropriate when different branches of a class hierarchy define attributes and/or methods that would be useful if available in a single class.

For example, a class on one branch of a hierarchy might define attributes appropriate to a PC, and a class on another branch might define attributes appropriate to a network. A class that combined both sets of attributes might be useful to represent network controllers.

With multiple inheritance, you could create a network controller class by defining a subclass that inherits both the PC class and the network class. With single inheritance your subclass could inherit only one of the existing classes, and would have to redundantly define the attributes of the other.

Multiple inheritance is also useful for adding specific properties to general-purpose classes. G2 defines various classes called mixins that are useful for this purpose. For example, the mixin class gsi-message-service gives any class that includes it a gsi-interface-name attribute and a data-server-for-messages attribute. You can include these attributes by adding the gsi-message-service class in the direct superiors of the user-defined class.

If you do not need to use multiple inheritance, you can skip to Defining Classes in Bottom-up Order. Before you proceed, be sure that you thoroughly understand single inheritance, as described in the previous section.

Multiple Inheritance and Class Inheritance Paths

In both single and multiple inheritance, a class's inheritance path is the key to working with the class. G2 uses class inheritance paths in exactly the same way for both single and multiple inheritance classes. However, deriving a class's inheritance path is more difficult in multiple inheritance than in single inheritance.

For a single inheritance class, there is only one reasonable way to order a class's inheritance path: the path lists the class itself, the class's parent, it's parent's parent, and so on, culminating in the root class. Thus the class hierarchy intrinsically provides every single inheritance class with a unique inheritance path.

For a multiple inheritance class, deriving an inheritance path is more complex. Some of the ancestors of the class exist on parallel branches of the hierarchy. The

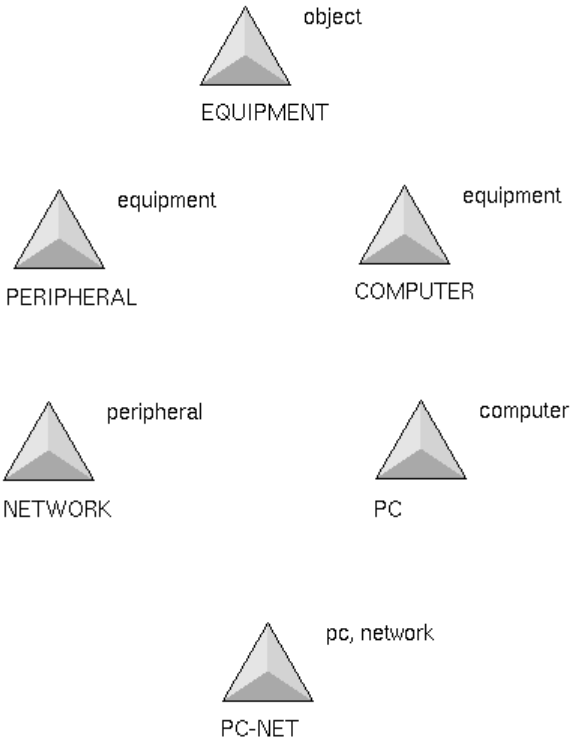
classes on the different branches have no intrinsic ordering relative to one another, so the class hierarchy does not inherently provide a unique inheritance path.

Multiple inheritance requires a determinate technique that can order all ancestors of a multiple inheritance class into a sequential class inheritance path. The task of assigning an order to the classes that contribute to a subclass via multiple inheritance is called **linearization**.

Many linearization techniques exist. The technique that G2 uses follows the same specifications as the L*Loops technique, which has become the standard technique for linearizing multiple inheritance. This chapter refers to G2's linearization technique as **G2 linearization**.

Linearizing Multiple Inheritance

A simple example can show you intuitively how G2 linearization works. The following figure shows the same class hierarchy that previous examples used, with the addition of a new multiple inheritance class called pc-net. Pc-net's direct superiors are pc and network, in that order.



The direct-superior-classes attribute display shows the two direct superiors in their preference order. The question is: what is pc-net's class inheritance path?

Like all class inheritance paths, pc-net's inheritance path must begin with the immediate class, and end with the root class. Thus pc-net's inheritance class begins with pc-net and ends with item. Because equipment is a single-inheritance class (equipment < object < item), linearizing pc-net's inheritance requires ordering the contributions of the two inheritance branches between pc-net and equipment.

There is no reason to shuffle the contributions of the two branches together, or to reorder their constituent classes. The most obvious approach is to string them together one after another. But in which order? pc appears before network in pc-net's list of direct superior classes, so the obvious answer is to give pc's contribution to pc-net's inheritance precedence over network's contribution.

Following these principles, the class inheritance path of pc-net is:

pc-net, pc, computer, network, peripheral, equipment, object, item

This simple example shows the general principle of G2 linearization: G2 linearizes a class's inheritance by interleaving the inheritance paths of its parents. The order in which the parents appear in the class's direct-superior-classes attribute, in conjunction with the existing class hierarchy, controls the contributions of those parents to the class's inheritance path.

How G2 Linearizes Multiple Inheritance

The inheritance path shown in the previous example is intuitively reasonable. However, linearization in more complex inheritance networks is not always so obvious. To deal with complex cases, you must know how G2 constructs a class inheritance path. You will then be able to predict the linearization of complex inheritance networks without having to construct them.

If you need multiple inheritance only to create simple structures that branch and rejoin in straightforward ways, as in the previous example, you do not need to study G2 linearization beyond this point. You can let G2 handle linearization automatically; the class inheritance paths that G2 produces will be intuitively obvious.

If you do not need further information on G2 linearization, skip to Defining Classes in Bottom-up Order.

The G2 Linearization Algorithm

G2 creates an inheritance path for a class by interleaving the inheritance paths of its direct superiors. The merge proceeds in the order given by the class's direct superiors list. G2 starts with the inheritance path of the primary direct superior, and merges the path of the next direct superior into it, yielding a path that expresses the contributions of both superiors.

G2 next merges in the inheritance path of the third superior class (if any), and so on through the list of direct superior classes. Thus G2 merges inheritance paths in pairs: at any given time, it has a **primary path**, and a **secondary path** to be merged into it. Having completed the merge, it proceeds to the next secondary path (if any), and so on.

G2 merges a secondary path into a primary path by applying the following algorithm. The algorithm is not easily grasped at first reading, but its action is essentially quite simple. You may want to just skim the algorithm, then refer back to it as you read through the examples that follow it.

1. Start with the immediate classes of the two inheritance paths.
2. Scan up the two paths to the first class that exists on both paths.

   Any class found on both inheritance paths is called a **common ancestor**.
3. Insert the classes that exist before the common ancestor on the secondary path into the primary path directly below the common ancestor.
4. Unless both paths are exhausted, continue to scan to the next common ancestor.
5. Insert any classes that exist in the secondary path between the previous common ancestor and the current common ancestor into the primary path just before the current common ancestor.

   Step 5 is functionally identical with Step 3, as explained below.
6. Continue thus to scan and merge until both paths are exhausted.

Since all class inheritance paths end with the root class, any two paths always end with a common ancestor. The only difference between Steps 3 and 5 is that Step 3 handles the boundary case where no previous common ancestor has been encountered.

Linearizing Two Superior Classes

To help you understand how G2 linearization works, the rest of this section shows several examples of its operation. In these examples, the top-level user-defined class inherits singly from the system-defined class object. To simplify the examples, object and its direct superior, item, are omitted from the examples.

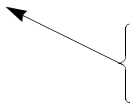
First let's take a more formal look at the previous example. Pc-net's primary and secondary class inheritance paths, shown in tabular form, are:

Primary (pc)	Secondary (network)
equipment	equipment

Primary (pc)	Secondary (network)
computer	peripheral
pc	network

The immediate classes of the two paths are network and pc, the direct superiors of pc-net. The first common ancestor is equipment. Therefore Step 3 inserts peripheral and network, from the secondary path, between equipment and computer on the primary path:

Primary (pc)	Secondary (network)
equipment	equipment
computer	peripheral
pc	network



The resulting primary path is:

Primary
equipment
peripheral
network
computer
pc

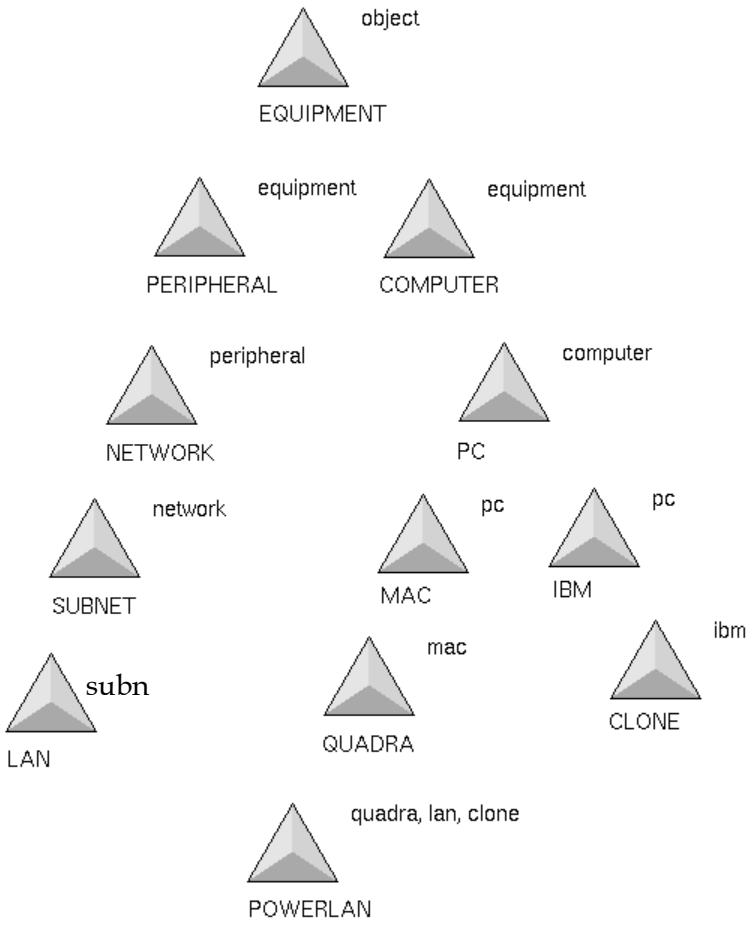
Since there are no additional classes superior to equipment (except object and item), and no more secondary paths to merge in, the algorithm is finished. The resulting interleaved path linearizes pc-net's inheritance. Adding pc-net itself to this path yields pc-net's class inheritance path:

pc-net, pc, computer, network, peripheral, equipment. object, item

This is the same path that we previously derived informally.

Linearizing Several Superior Classes

To get an idea of how G2 handles more complex inheritance, consider the following example:



If you specify powerlan's direct superiors as:

quadra, lan, clone

powerlan's class inheritance path is:

powerlan, quadra, mac, clone, ibm, pc, computer, lan, subnet, network, peripheral, equipment, object, item

The following tables show how G2 derives powerlan's inheritance path by merging the inheritance paths of powerlan's direct superiors:

Primary (quadra)	Secondary (lan)	Secondary (clone)
equipment	equipment	equipment
computer	peripheral	computer
pc	network	pc
mac	subnet	ibm
quadra	lan	clone

Primary	Secondary (clone)
equipment	equipment
peripheral	computer
network	pc
subnet	ibm
lan	clone
computer	
pc	
mac	
quadra	

Which yields the class inheritance path given above:

> powerlan, quadra, mac, clone, ibm, pc, computer, lan, subnet, network, peripheral, equipment, object, item

This inheritance path is probably not what you would expect intuitively. The classes clone and ibm, which exist in the path only by virtue of the third direct superior, clone, take precedence over classes contributed by both the second and the primary direct superior. This is definitely not an ideal ordering.

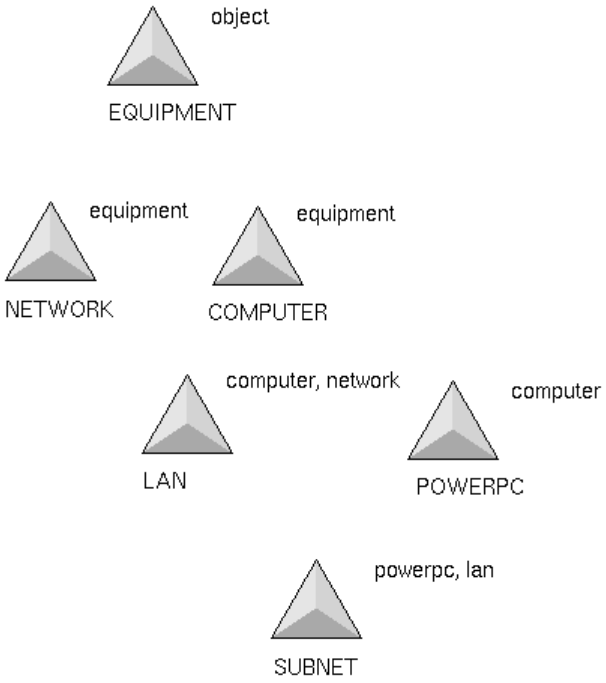
However, notice the bizarre nature of powerlan's direct superiors list. Inserting the distantly related class lan between the closely related classes quadra and clone makes very little sense. clone and ibm take precedence over so many other classes, not because G2 linearization has produced an undesirable result, but because it was given an illogical inheritance to linearize.

This illustrates a general principle of linearization: strange class inheritance paths usually reflect strange inheritance patterns that should be reorganized to have a more logical structure.

Linearizing Networks of Classes

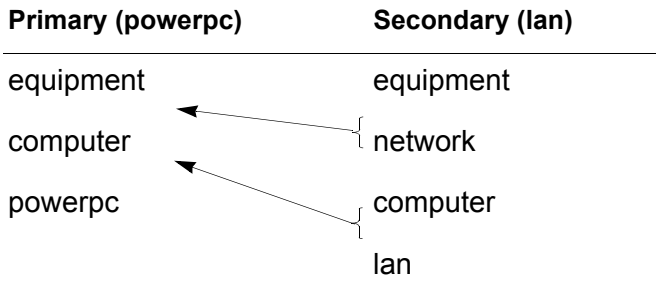
In the preceding examples of linearization, the class hierarchy was essentially a tree: the root class diverged into parallel branches that merged at their tips to create a multiple inheritance class. Thus no ancestor of a class with multiple inheritance also had multiple inheritance.

Inheritance networks can become very complex, with multiple inheritance classes inheriting other such classes. For example, consider the following inheritance network:



G2 linearizes class inheritance by working from the top down. In this case, it first linearizes lan's inheritance, then proceeds to linearize subnet's inheritance. Since multiple inheritance networks are hierarchies (have a unique root and do not contain circular structures), G2 can always linearize from the top down without encountering mutually dependent needs for a previously computed class inheritance path.

In this case, lan's inheritance path is lan, computer, network, equipment, and powerpc's inheritance path is powerpc, computer, equipment, so subnet's inheritance path is subnet, powerpc, lan, computer, network, equipment:



Why G2 Linearizes As It Does

You may wonder why a relatively complex algorithm like G2 linearization is necessary to accomplish something that seems initially to be so simple. The reasons are largely beyond the scope of this document, but this section provides a brief overview. If you do not need this information, skip to Defining Classes in Bottom-up Order.

For more detailed information, consult a text on the theory of object-oriented programming, and/or the paper "Monotonic Conflict Resolution Mechanisms for Inheritance", Ducournau, Habib, Huchard, and Mugnier, ACM Proceedings for OOPSLA '92.

Ideal Linearization

The intuitively ideal linearization algorithm would always produce a class inheritance path that has the following properties:

- Every class inherited from a higher-precedence direct superior appears before any class inherited from a lower-precedence direct superior.
- Every inherited class C appears before any inherited class that is superior to C in C's own class inheritance path.

That is, we want a linearized class inheritance path to reflect the precedence in the class's direct superiors list without allowing any superior class to take precedence over its own descendents.

Feasible Linearization

Unfortunately, these two requirements are intrinsically at odds: no possible linearization algorithm can satisfy both of them in every case. The ability to linearize any multiple inheritance structure therefore requires that one of these goals be compromised in some cases. The question is: which one?

- Compromising hierarchical precedence

  Allowing a more general class to ever take precedence over one of its own descendents creates a structure in which both A>B and B>A. The result would be an unusable tangle of mutually contradictory inheritances. The requirement to maintain strict hierarchy cannot be compromised.

- Compromising direct superior precedence

  Allowing a class inherited from a lower-precedence superior to sometimes take precedence over a class inherited from a higher-precedence superior produces results that are counterintuitive, but not disastrous. Successful linearization, and thus the ability to use multiple inheritance at all, requires accepting this compromise. A good linearization algorithm minimizes its negative effects.

G2 Linearization

G2 linearizes as it does because the algorithm has several desirable properties:

- In most cases, it does in fact linearize inheritance in the intuitively ideal way.
- It never allows a parent class to take precedence over a child class under any circumstances.
- It minimizes the degree to which classes inherited from lower-precedence superiors take precedence over classes inherited from higher-precedence superiors.

While no linearization technique gives ideal results in every case, G2 linearization, though more complex than most linearization algorithms, gives results as good as or better than any other.

Illegal Patterns of Multiple Inheritance

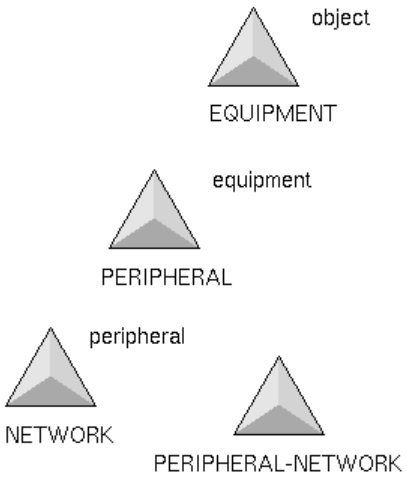
G2 prohibits you from specifying some patterns of multiple inheritance because they would cause disordered or meaningless results. If you specify an illegal pattern of multiple inheritance, G2 displays a descriptive error message when you try to close the edit of the direct-superior-classes attribute.

As long as you use multiple inheritance in straightforward ways, you will rarely if ever specify an illegal pattern of multiple inheritance. Thus you do not need to

study the subject in detail so as to avoid trouble later on. This section describes the general nature of the two types of illegal inheritance.

Disordered Multiple Inheritance

Some patterns of multiple inheritance are illegal because the resulting inheritance could not have a logically ordered linearization. To get a general idea of this type of illegal inheritance, consider the following figure:



The class-definition named peripheral-network was planned to have peripheral and network, in that order, as its direct superior classes.

This specification is illegal because peripheral would be unable to precede network in the class inheritance path as dictated by the direct-superior-class precedence ordering. This is because peripheral is superior to network in the class hierarchy. A subclass cannot precede a superior class in a class inheritance path.

To prevent this kind of problem, G2 requires that any class specified in a list of direct superiors must precede any superior class that also appears. That is, a more general class cannot take precedence over a more refined subclass in a list of direct superiors.

In the current example, you could solve the problem by reversing the order of inheritance, so that peripheral-network specifies its direct superior classes as network and peripheral, in that order. However, such a specification, though not illegal, is pointless: peripheral-network can inherit from network anything it would inherit from peripheral. Thus there is no need for multiple inheritance in this case.

This illustrates a general principle of multiple inheritance: where illegal multiple inheritance occurs, something is probably wrong with the plan that led to it.

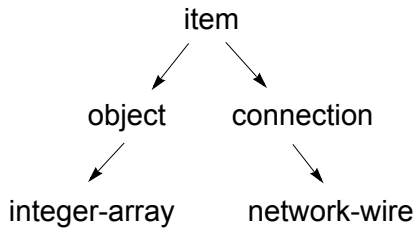
Meaningless Multiple Inheritance

Some patterns of multiple inheritance are illegal because the resulting class could not serve a reasonable purpose because of contradictory functionalities. For example, consider a class that inherits g2-window and connection. The resulting class would be nonsensical.

G2 defines any system-defined concrete or abstract class as a **foundation class**. To prevent meaningless multiple inheritance, G2 enforces the following rule:

> When a class inherits more than one foundation class,
> the classes must all be in the same line of inheritance.

For example, integer-array is a subclass of object, which is a subclass of item, and network-wire is a subclass of connection which is a subclass of item.



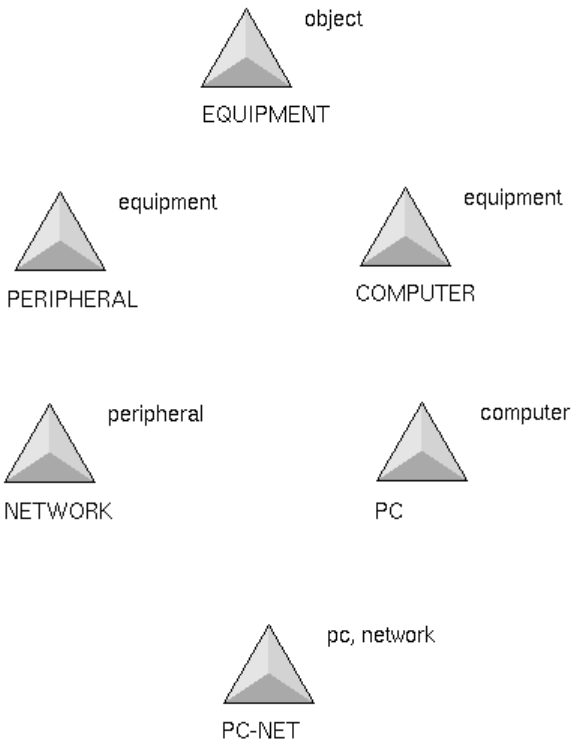
A user-defined class could inherit from a combination of item, object, and integer-array, because all three are in the same line of inheritance. Similarly, a user-defined class could inherit from any combination of item, connection, and network-wire. However, a class that tried to inherit object and network-wire, or any other combination of foundation classes in different lines of inheritance, would be illegal.

As pointed out in the previous section, inheriting both a class and a direct superior of that class serves no purpose, because the class already has everything that the superior has. However, G2 applications sometimes define subclasses of different foundation classes, then combine those subclasses using multiple inheritance. Such a subclass can exist only when all foundation classes in its ancestry are in the same line of inheritance.

Viewing Multiple Inheritance with the Inspect Facility

The Inspect facility does not attempt to show multiple inheritance as a network: for inheritance networks of any significant complexity, such displays are typically indecipherable. Instead, Inspect duplicates the representation of a class with multiple inheritance as needed to show all connections between the class and its ancestor.

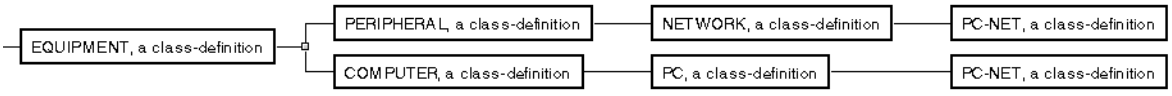
For example, given the following class hierarchy:



this Inspect command:

```
show on a workspace the class hierarchy of pc-net
```

results in the following class hierarchy (with item and object omitted):



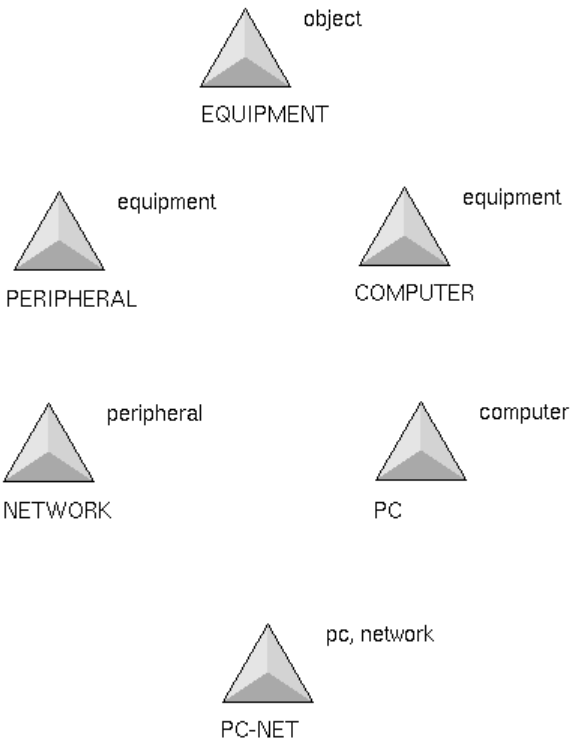
For further information on using Inspect, see The Inspect Facility.

Default Values in Multiple Inheritance

In single inheritance, a class has the same default values that its parent class has, except where the class overrides those values with values of its own. This property results from the fact that the class has only one parent, and therefore only one source from which it can inherit default value definitions.

In multiple inheritance, specifying a class as the primary direct superior does not guarantee that the subclass inherits all default values from that class. A subclass inherits the default value of an attribute from the first class in the class inheritance path that *explicitly defines* one, *not* from the first class that has one by inheritance.

The examples in this section show how G2 uses the class inheritance path to determine which default values a multiple inheritance class inherits. The examples focus on the system-defined attribute icon-description, which defines the icon displayed by an instance. The examples use the pc-net multiple inheritance structure:

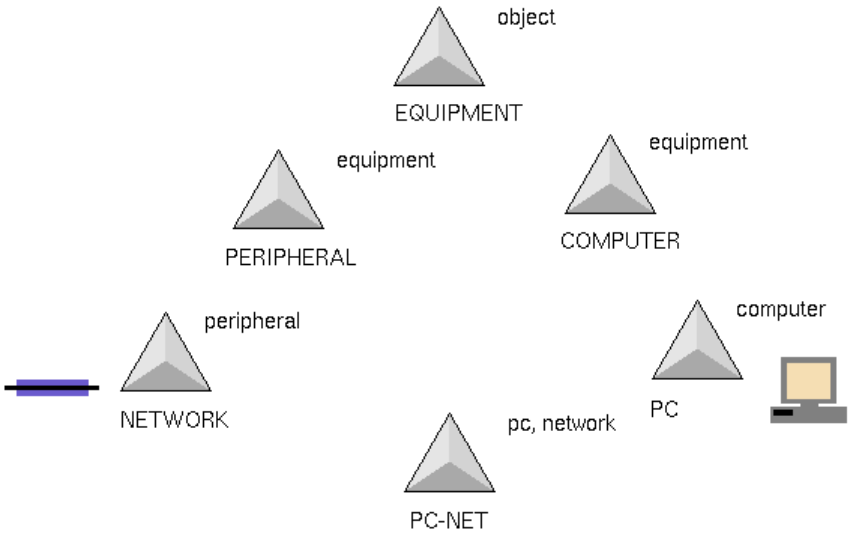


Pc-net's direct superiors are pc and network, in that order; pc's class inheritance path is therefore:

pc-net, pc, computer, network, peripheral, equipment, object, item

Inheriting a Default Value from a Direct Superior

In the next figure, both the pc and network classes have their own icon definitions. As in the previous example, the icon by pc is called the workstation icon, and the icon by network is called the node icon.



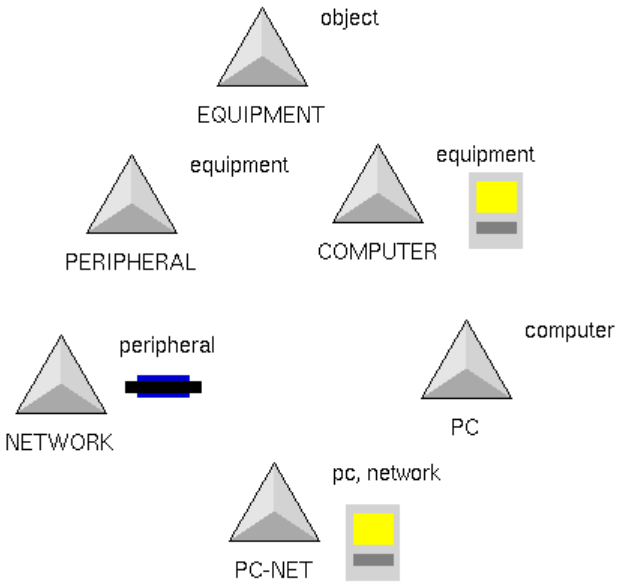
Since pc appears before network on pc-net's class inheritance path, pc-net inherits the workstation icon.

Networks's icon definition has no effect on pc-net: from pc-net's perspective, network's icon definition does not exist: pc's definition has overridden network's definition.

If you deleted pc's icon definition, and did not define any other icon on the path between pc-net and network, network's icon definition would cease to be overridden, and would be pc-net's icon definition also. A pc-net instance would then have a node icon.

Overriding the Default Value of a Direct Superior

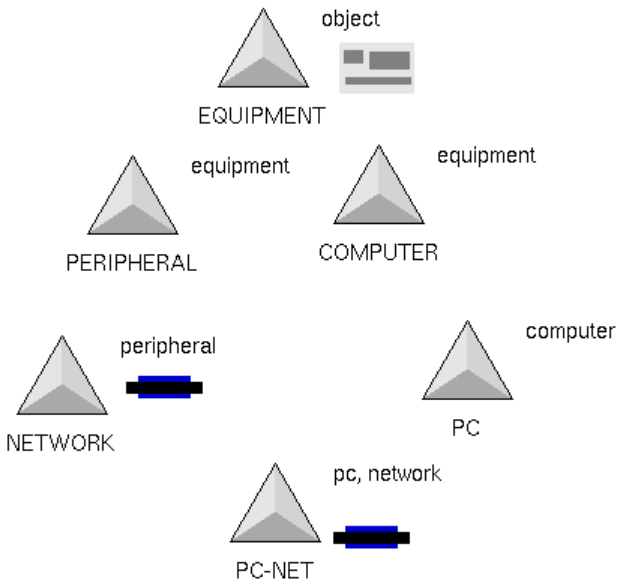
In the next figure, pc does not define an icon, but computer and network do:



Since computer appears before network on pc-net's class inheritance path, pc-net inherits the workstation icon: the default value given by network, a direct superior is overridden. The class inheritance path, *not* the list of direct superiors, determines default value inheritance.

Overriding an Inherited Value with an Explicit Value

In the next figure, network and equipment both define icons:



Since network appears before equipment on pc-net's class inheritance path, pc-net inherits the node icon: the default value that pc inherits from equipment via computer is overridden. Only explicit specifications, *not* inherited specifications, can provide default values.

Inheriting Default Values for Stubs

Icons and stubs are closely related, but they are specified by two different attributes of a definition. Where multiple inheritance exists, this independence could result in mismatched icons and stubs, so the G2 class inheritance rules contain a special provision that prevents it.

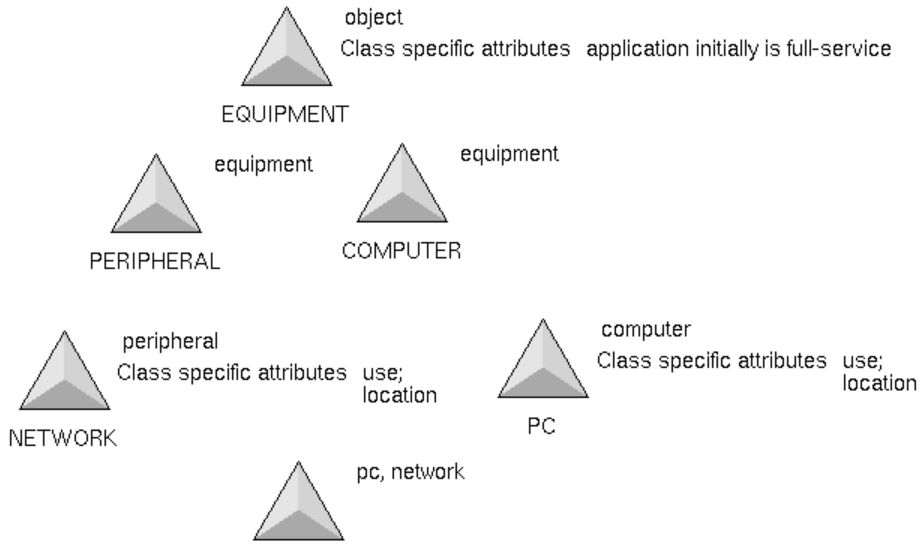
To prevent such a mismatch, a class can inherit a stubs definition *only* from the class from which it inherits its icon definition, or from a descendent of that class. If none of these provides a stubs definition, either inherited or locally defined, the class's stubs default value is none.

Duplicate Attributes in Multiple Inheritance

Where multiple inheritance exists, a subclass can inherit user-defined attributes with the same name from more than one superior class. Such duplication does not arise when a subclass inherits the same attribute through more than one path. It occurs only when different attributes with the same name exist in different ancestral classes.

G2 handles duplicate attributes that arise from multiple inheritance just as it does those arising from single inheritance: it follows the class inheritance path, uses the first definition it encounters without qualification, and qualifies any others with the relevant class name.

For example, consider again the pc-net multiple inheritance structure:



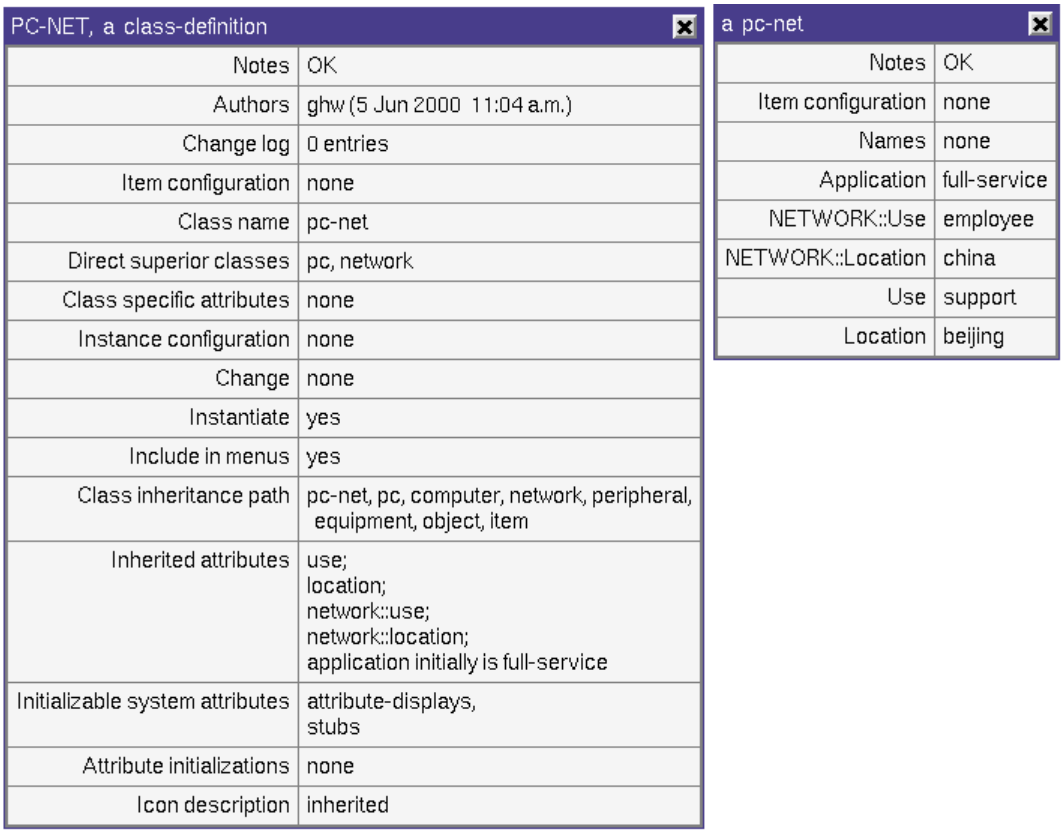
Suppose that, as indicated in the figure:

- equipment defines the attribute application
- pc defines attributes use and location
- network defines attributes use and location
- pc-net defines the attribute use
- pc-net's direct superior classes are pc and network, in that order

Pc-nets's class inheritance path is:

pc-net, pc, computer, network, peripheral, equipment, object, item

The following figure shows the definition table for pc-net, along with the table of an instance of pc-net:



PC-NET, a class-definition

Notes	OK
Authors	ghw (5 Jun 2000 11:04 a.m.)
Change log	0 entries
Item configuration	none
Class name	pc-net
Direct superior classes	pc, network
Class specific attributes	none
Instance configuration	none
Change	none
Instantiate	yes
Include in menus	yes
Class inheritance path	pc-net, pc, computer, network, peripheral, equipment, object, item
Inherited attributes	use; location; network::use; network::location; application initially is full-service
Initializable system attributes	attribute-displays, stubs
Attribute initializations	none
Icon description	inherited

a pc-net

Notes	OK
Item configuration	none
Names	none
Application	full-service
NETWORK::Use	employee
NETWORK::Location	china
Use	support
Location	beijing

Note that the application attribute appears only once, even though pc-net inherits it from two different sources: pc and network. Both of these definitions are really the same definition, inherited from equipment, so G2 defines only one application attribute in pc-net. G2 qualifies the other inherited attributes as needed by following the class inheritance path, as described above.

If pc-net's direct superior classes were network and pc, in that order, giving pc-net the class inheritance path:

pc-net, network, peripheral, pc, computer, equipment, object, item

the attributes of a pc-net would be:

use
location
pc::use
pc::location
application

Defining Classes in Bottom-up Order

In order for a class to be fully defined, the class must have a name, and either an existing direct superior class (single inheritance), or an acceptable list of such classes (multiple inheritance). This information allows G2 to determine the class's inheritance path, add the class to the G2 class hierarchy, and thereafter instantiate the class on demand.

If you define classes top-down, completely specifying each one before defining any subclasses, each class becomes fully defined, and can be instantiated, as soon as you complete its definition. However, this order of creation is not always convenient. Therefore G2 does not require you to define a class's direct superior(s) before you name them in the class's definition.

This provision allows you to define classes in bottom-up order. A class definition with one or more direct superior classes that have not yet been defined does *not* add the new class to the hierarchy, and does *not* permit the class to be instantiated.

You can have any number of partially completed class definitions in a KB without affecting the class hierarchy or KB execution in any way. The notes attribute of any such definition has an incomplete status, and states that one or more direct superior classes is not defined.

If many class definitions specify the same nonexistent superior class, defining the superior will complete all of the definitions that depend on it, and add them all to the hierarchy at once.

Deleting a Class Definition

Deleting a class definition automatically deletes all instances of that class, and all instances of any subclasses that inherit it. Instance deletion occurs because an instance requires a complete set of information from every superior class within its class hierarchy. When you delete any superior class within that instance's class hierarchy, such information no longer exists, and thus, neither can the instance.

All subclasses of a deleted superior class retain their class definitions. The notes attribute of any such subclass, however, has an incomplete status, and states that one or more direct superior classes is not defined. The same notification would appear if the subclass had been created as part of bottom-up development, and the deleted parent class had never existed.

Planning a Class Hierarchy

Planning a class hierarchy for your G2 knowledge base is an important part of completing your application. Following are some general guidelines:

- Try to plan as many of your classes at one time as possible. The hierarchy you develop depends on the interrelationships of your classes. One of the first steps to planning a class hierarchy is to list the most important properties that each of your prospective classes will have.
- Use *factoring* to build a provisional class hierarchy based on the properties. In factoring, you develop classes by grouping all common properties as high up in the class hierarchy as you can. For instance, the classes vertebrates and invertebrates group all animals with and without spines into two high-level classes. Subclasses beneath these specify additional groups with common properties.
- Consider the probable patterns of referencing objects when deciding which kinds of properties should be the basis for your hierarchy. For instance, in a taxonomy, the class basis is *structural*. If, in your application, you are more likely to reference objects by some other property, say habitat, your high level classes should reflect this. You might have high-level classes such as land-animal, air-animal, and water-animal instead of the structural categories.
- Multiple-inheritance allows you to classify entities under two or more major classifications. Don't neglect its potential to help you to represent complex structures of information.

G2 allows you to change user-defined classes and class hierarchies as needed. Such changes automatically propagate to all existing subclasses and instances. However, you may also have to change other parts of the KB to be consistent with the new class definitions, and this is sometimes difficult. Careful thought in laying out user-defined classes and class hierarchies can save significant time later on.

14

Definitions

Describes class definitions and shows you how to use them.

Introduction

The ability to create custom classes, and thus to extend the G2 class hierarchy, is fundamental to representing the particular kinds of knowledge most suitable to your KB's requirements. G2 provides a variety of classes useful for representing knowledge. You can extend these as needed to represent knowledge of any type.

You can also use custom classes to extend the machinery of G2 itself. Almost every class that G2 uses to implement a KB is extensible. For example, you can create a subclass of kb-workspace and instantiate it as needed to create workspaces that have whatever attributes and default values you need.

The G2 class hierarchy includes four special classes that you can use to extend the class hierarchy. These classes are **definition classes**, and an instance of any of them is a **definition**. Creating a definition adds a new class to the hierarchy. The definition specifies the inheritance and attributes of the new class. The new class can be used as soon as the definition is complete.

You can use a definition to create a subclass of any extensible class (single inheritance) or classes (multiple inheritance), including user-defined classes, subject to the restrictions described under Illegal Patterns of Multiple Inheritance on page 523. These subclasses can be abstract or concrete, as described under Specifying Instantiability on page 560. No practical limit exists to the number of classes in the class hierarchy, or the depth to which it can be extended.

The four types of definitions available in G2 are:

- class-definition: Creates a subclass of any kind.
- object-definition: Creates a subclass of object.
- connection-definition: Creates a subclass of connection.
- message-definition: Creates a subclass of message.

You can define anything with a class-definition that you can with any of the more specialized types of definitions. The specialized definitions are supported to provide compatibility with previous versions of G2, which did not provide a generic class definition capability.

Most of this chapter describes class-definitions. Information relating to the specialized types of definitions appears under Using Specialized Definitions on page 585.

Note Before you read this chapter, you should understand G2 classes and the G2 class hierarchy, as described in Chapter 13, Classes and Class Hierarchy on page 497.

Terminology

The terminology for class definitions can be confusing, because object, connection, and message definitions are class definitions in a generic sense, in that they define classes, but are not the same as class-definitions. To prevent ambiguity, we use the following conventions:

- The informal term "class definition" refers generically to any kind of class definition.
- The formal terms class-definition, object-definition, connection-definition, and message-definition refer to specific types of definition.
- The term "*type* class definition" refers to any definition that creates a subclass of *type*. For example, an "object class definition" creates a subclass of object, and could be either a class-definition or an object-definition.

Overview of the Class Definition Process

The following outline summarizes the use of a class-definition to define a class.

To create a class-definition:

➔ Instantiate a class-definition onto a workspace.

To specify the attributes of the class:

1. Specify the class's name and superior class(es).
2. When the superior classes have been specified, G2 supplies values to read-only attributes that show:
   - The class inheritance path.
   - The initializable attributes inherited from the system-defined superior(s).
   - Any attributes inherited from user-defined superiors.
3. Define any attributes that are specific to the new class, and specify their default values.
4. Override default values as needed for attributes inherited from user-defined superior classes.
5. Provide default values for attributes inherited from system-defined superior classes.

To specify other properties of the class:

1. Specify any configurations that apply to the class-definition itself.
2. Specify any configurations that apply to instances of the class defined.

3 Specify the instantiability of the class (if applicable).

4 Specify the icon of the class (if applicable).

This outline orders the various steps to highlight their functional groupings. The order in which the steps are described in this chapter is optimized to facilitate learning and reference. The order in which the steps are actually performed varies widely. Any order that results in a correct definition will work.

You can define classes either interactively, by instantiating and completing a definition, or programmatically, by writing procedures that use the create action. Most of this chapter describes interactive class definition. Programmatic class definition is described under Creating New Classes Programmatically on page 591.

The development of a class often requires changing various aspects of its definition over time. Such changes can be made in any order: all that matters is the ultimate correctness of the definition. When no class instances exist, changing a class requires only changing the relevant attribute(s) of the definition. When instances exist, they must be then updated to reflect the new definition, as described under Changing Definitions on page 591.

Creating Class Definitions

A class-definition can define a subclass of any extensible class.

To create a new class-definition:

1 Select KB Workspace > New Definition > class-definition > class-definition.

2 Click to place the new definition on a workspace:

Storing Definitions on Workspaces

A definition specifies the attributes of a new class. Because G2 uses the definition to interpret the attributes of each class instance, the definition must be available any time G2 references the instance. Similarly, subclasses (and their instances) also require the definitions of the superior classes.

By default, G2 stores a definition item on the workspace from which you choose the new-definition option. When choosing a workspace to contain your definitions, select one that will remain enabled or active as long as any workspace containing instances and subclasses is active. A disabled or inactive workspace disables all items that reside upon it, along with their instances and subworkspaces.

Class Definition Attributes

The class-specific attributes of a class-definition are as follows:

Attribute	Description
item-configuration	Configuration statements that apply to this item and to all items below it in the workspace hierarchy. Compare with instance-configuration.
Allowable values:	Described in Chapter 7, on page 291.
Default value:	none
class-name	The name of the class being added to the class hierarchy.
Allowable values:	Any unique symbol.
Default value:	none
direct-superior-classes	The names of one or more direct superior classes.
Allowable values:	Any list of class-names that result in acceptable class inheritance.
Default value:	none
class-specific-attributes	The attributes specific to this class.
Allowable values:	For attribute names, any symbol that is not the name of an inheritable system-defined attribute.
Default value:	none

Attribute	Description
instance-configuration	Configuration statements that apply to all instances of this class. Compare with item-configuration.
Allowable values:	Described in Chapter 7, Configurations on page 291.
Default value:	none
change	Changes certain user- and system-defined attribute values.
Allowable values:	Described in Using the Change Attribute on page 592.
Default value:	none
instantiate	Whether the class is can be instantiated. Inapplicable to connections.
Allowable values:	yes, no
Default value:	yes
include-in-menus	Whether the class appears in the G2 definition menus. Inapplicable to connections.
Allowable values:	yes, no
Default value:	yes
class-inheritance path	The class inheritance path of the class.
Allowable values:	G2 provided.
Default value:	none

Attribute	Description
inherited-attributes	User-defined attributes that the class inherits.
Allowable values:	Any inherited attribute descriptions.
Default value:	none
initializable-system-attributes	The names of all initializable system attributes inherited from the class's superiors.
Allowable values:	Depend on the particular superiors.
Default value:	none
attribute-initializations	Class overrides of the default values of some inherited system-defined attributes, and of user-defined inherited attributes.
Allowable values:	See Specifying Default Values for Inherited Attributes on page 556.
Default value:	none
icon-description	The textual description of an item's icon. Not applicable to subclasses of connection or message, or other classes that have no iconic representation, such as kb-workspace.
Allowable values:	See Chapter 46, The Icon Editor and Icon Management on page 1637.
Default value:	inherited

Formatting the Text of Attributes

In some cases, when you enter attributes in a definition table, G2 does not save the formatting characters you enter, such as Control + j for a new line, after you exit from the Text Editor.

In other cases, G2 adds formatting automatically. For example, in the attributes-specific-to-class attribute, G2 stores each attribute on a separate line, even if you enter multiple attribute names on one line in the Text Editor.

Order of Attributes in Tables

When you create an instance of a user-defined class, the order of attributes in its table is determined by the order in which the attributes are inherited and defined in the class definition. Attributes defined by the class itself appear at the bottom of the table. Those defined by the first direct superior class appear above them, and so on.

The inherited attributes appear in the order that the superior class defined them as class-specific attributes. An item's attribute table lists attributes in this order, from the top of the attribute table to the bottom:

1. System-defined attributes
2. User-defined inherited attributes
3. Class-specific attributes

Configuring Class Definitions

This section describes:

- Attributes and techniques that supply information needed in every class definition.
- Attributes that provide information that is useful for completing every class definition.

Subsequent sections describe attributes and techniques whose applicability depends on the type of subclass being defined.

Specifying the Item Configuration

The item-configuration attribute determines which configurations are in effect for this class. These configurations apply only to the definition itself, *not* to instances of the class that it defines. Instance configurations are described under Specifying Instance Configurations on page 546. For a description of configuration clauses, see Chapter 7, Configurations on page 291.

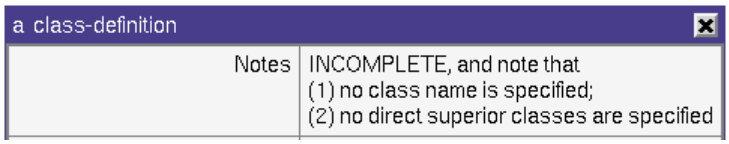
Providing a Class Name

Use the class-name attribute to specify the name of the class. You can use any symbol that is not already in use. You cannot use a G2 reserved word, or any symbol that denotes a G2 data type. Once specified, the class name appears below the definition's icon:

The class-naming convention in G2 is to use hyphens to separate words in a class name. You can use another allowable G2 character, such as an underscore (_), if you wish.

Note All class names must be unique within your KB. A class name cannot conflict with another class name even if the definition is of a different type. For instance, you cannot create a class-definition and an object-definition with the same class name.

You must complete both the class-name and direct-superior-classes attributes of your definition before G2 considers the definition complete and adds it to the class hierarchy and the menu structure. Until you complete both of those attributes, a definition displays incomplete in its notes attribute, as shown in the partial definition that follows:



Specifying the Superior Class(es)

The direct-superior-classes attribute determines the name of one or more direct superior classes for the new class.

Clicking on any foundation-class in the Text Editor displays a list of all system-defined and user-defined classes (other than mixins) that can be inherited.

Entering Direct Superior Classes

Providing one or more extensible direct superior classes (along with a valid class name) adds the new class to the class hierarchy. When a class exists within the class hierarchy, it inherits attributes from all of its superior classes. Note that you cannot:

- Change the system-defined class hierarchy.
- Create a circular inheritance between or among classes; for example, you cannot make two classes the superior of each other.

A class exists in a KB when it has a unique name and a set of acceptable existing direct superior classes. These prerequisites make it possible to determine the inheritance of the new class, add it to the class hierarchy, and instantiate the class.

If you are creating a class with multiple inheritance (more than one direct superior class), the name of the direct superior class you enter first is significant. The first class in the list is the primary direct superior class. All other direct superior classes are secondary superior classes.

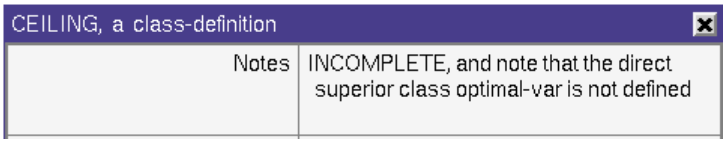
Some restrictions on multiple inheritance must be observed, as described under Illegal Patterns of Multiple Inheritance on page 523.

Specifying Direct Superiors Before Creating Their Definitions

You can enter the names of direct superior classes before you have defined them. Specifying a definition with one or more direct superior classes that are not defined, however, does not add the new class to the hierarchy. It merely provides a means for you to create definitions prior to creating all of the required direct superior classes.

A class that does not exist may have potential subclasses, but it cannot have instances. The potential subclasses come into existence simultaneously as soon as you define the missing direct superior class.

When you specify a nonexistent class as a direct superior, the notes attribute of the class-definition that specifies the missing superior includes an incomplete message such as:



Using Mixin Classes

You can use some classes, called mixin classes, with one or more other direct superior classes to add specific properties to a subclass. To use a mixin class, define a class whose direct-superior-classes attribute includes one or more of the following mixins:

gsi-data-service
gsi-message-service
g2-to-g2-data-service
g2-meter-data-service
unique-identification

The mixins gsi-message-service and unique-identification can be mixed in with any class. The other mixins shown can be used *only* with subclasses of variable. Mixin classes add the following system-defined attributes to a subclass:

This mixin class...	Provides this attribute(s) to your subclass...
gsi-data-service	gsi-interface-name attribute gsi-variable-status attribute Both attributes are described in Chapter 63, G2 Gateway on page 1985.
gsi-message-service	gsi-interface-name attribute data-server-for-messages attribute These attribute is described in Chapter 63, G2 Gateway on page 1985.
g2-to-g2-data-service	g2-to-g2-interface-name attribute remote-g2-expression attribute Both attributes are described in Chapter 62, G2-to-G2 Interface on page 1943.
g2-meter-data-service	g2-meter-name attribute This attribute is described in Chapter 54, G2-Meters on page 1841.
unique-identification	uuid This attribute is described in Using Universal Unique Identifiers on page 471.

Specifying Instance Configurations

The instance-configuration attribute specifies the configuration statements for all instances or subclasses of a definition. These configurations apply only to instances of the defined class, *not* to the definition itself. Definition configurations are described under Specifying the Item Configuration on page 542. For a description of configuration clauses, see Chapter 7, Configurations on page 291.

For instance, you may want to configure all instances of a class so that they do not include the delete menu option. To do this, you could enter the instance configuration as shown here:

```
configure the user interface as follows:
    when in user mode:
        menu choices for menu-test exclude: delete
```

Determining the Class Inheritance Path

The class-inheritance-path attribute specifies the inheritance path from the class you are defining to item. For details, see Class-Inheritance-Path Attribute on page 507 and Multiple Inheritance and Class Inheritance Paths on page 514.

G2 completes this attribute as soon as you enter one or more direct superior classes and close the edit for the direct-superior-classes attribute. You cannot edit the value of the class-inheritance-path attribute.

For G2 to complete the class-inheritance-path attribute, the direct superior classes you specify must already exist and the inheritance you specify must be valid.

Determining the Initializable System Attributes

The initializable-system-attributes attribute lists the names of all initializable attributes that the class definition inherits from the system-defined class(es) on its class inheritance path. This display is read-only, and appears as soon as you specify one or more legal direct superiors as the value of a class definition's direct-superior-classes attribute. The default value is none.

For example, the next figure shows the system-defined attributes that you can initialize for a procedure subclass:

DISTRIBUTION-PROCEDURE, a class-definition

Notes	OK
Authors	ghw (5 Jun 2000 3:07 p.m.)
Change log	0 entries
Item configuration	none
Class name	distribution-procedure
Direct superior classes	procedure
Class specific attributes	none
Instance configuration	none
Change	none
Instantiate	yes
Include in menus	yes
Class inheritance path	distribution-procedure, procedure, item
Inherited attributes	none
Initializable system attributes	tracing-and-breakpoints, class-of-procedure-invocation, default-procedure-priority, uninterrupted-procedure-execution-limit, attribute-displays, stubs
Attribute initializations	default-procedure-priority: 3; uninterrupted-procedure-execution-limit: 3 seconds
Icon description	inherited

System-defined attributes you can override using the attribute-initializations attribute.

Techniques for providing default values for initializable system attributes are described under Specifying Default Values for Inherited Attributes on page 556.

Determining the Inherited User-Defined Attributes

The inherited-attributes attribute lists the user-defined attributes that the class inherits from its superior classes. This display is read-only, and appears as soon as you specify one or more legal direct superiors as the value of a definition's direct-superior-classes attribute. The default value is none.

Inherited user-defined attributes appear in the order of the class hierarchy of the class you are defining. Techniques for providing default values for such attributes appear under Specifying Default Values for Inherited Attributes on page 556.

Defining and Initializing Class-Specific Attributes

The class-specific-attributes attribute defines all user-defined attributes that are specific to the class you are creating. The specified value of an attribute can be a value or an item. The default value of this attribute is none.

The G2 compiler prevents you from defining an attribute that has the same name as a system attribute defined by a user-extensible system-defined class. For examples, module-search-path can be a user-defined attribute name because its system-defining class, server-parameters, is not user-subclassable; whereas validity-interval is rejected because it is defined for the user-subclassable variable system-classes.

You can specify **simple attributes** (an attribute name without a type or value). Simple attributes have a value of none when they appear in instances and can contain a value of any G2 type.

Note We recommend that your attributes have a default value and a specified type to simplify attribute access and to take advantage of G2's type checking facility.

You can optionally specify user-defined attributes to be of a G2 type, with or without a default value, or to be an instance of an object. When an attribute is an instance of an object, G2 creates two items whenever you instantiate the class, one for the class instance, and another for the attribute value. An object created as the value of an attribute does not have an iconic representation and does not appear on a workspace.

When specifying user-defined attributes, you can declare the attribute to be untyped or typed, and to have a default value explicitly specified or provided automatically by G2. For information on G2 types, see Chapter 9, Values and Types on page 379.

To specify that the attribute has...	Enter a statement like this...
No type or default value	temp; weight; ranch-type
No type and a specified default value	temp initially is 98.6; weight initially is given by a float-variable; emergency-code has values red-alert, blue-alert, or green-alert

To specify that the attribute has...	Enter a statement like this...
A type with a specified default value	temp is a float, initially is 98.6; weight is given by a quantitative-variable, initially is given by a float-variable
A type with a default value provided automatically by G2	temp is a float; weight is given by a float-variable; ranch-type is an instance of a house

After you enter an attribute name, the editor prompts you to specify the attribute in one of several ways:

- *attribute-name* is a/an
- *attribute-name* is an instance of
- *attribute-name* is given by a/an
- *attribute-name* initially is
- *attribute-name* has values

You can optionally follow each of these choices with the with an index clause. The next sections describe each kind of attribute expression, as well as various ways in which you can format class-specific attributes.

Defining an Untyped Attribute with No Default Value

Specify an attribute that has no predefined properties except its name by entering the name. For example:

```
temperature;
maximum-height
```

Defining an Untyped Attribute with a Default Value

Specify an attribute that has no type with a default value by using the initially is phrase. For example:

```
past-time initially is 10 minutes;
operator-reading initially is cool;
set-up-message initially is "The beginning of this is:";
number-of-plants-online initially is 10;
temp initially is sequence(5, 6, 7);
dim initially is structure(length: 767, height: 45387);
list-of-messages initially is an instance of a g2-list;
temperature initially is given by a float-variable
```

You can follow the initially is phrase with:

- Any symbol.

- Any string.
- Any number, optionally followed by a unit of measure.
- true or false.
- The given by phrase, followed by a variable or parameter class.
- An instance of any subclass of object.

Defining a Typed Attribute with a Specified Default Value

Specify an attribute as a particular type with a default value using the is a and initially is phrases. For example:

```
temperature is a float, initially is 98.6;
maximum-height is an integer, initially is 12;
temp is a sequence, initially is sequence(6, 7, 8);
dim is a structure, initially is structure(length: 767, height: 45387)
```

Defining a Typed Attribute with a Default Value

Specify an attribute to have a particular type by using the is a phrase.
For example:

```
temperature is a float;
maximum-height is an integer
dim is a structure
```

The type can be: item-or-value, symbol, value, truth-value, quantity, integer, float, text, structure, or sequence.

Specifying an attribute with a specific type constrains the attribute value to be of that type when:

- Editing that attribute value in a class instance, where the text editor prompts you to enter a value of the attribute type.
- Concluding a new value for an attribute.
- Using the attribute-initializations to provide an attribute value (see Specifying Default Values for Inherited Attributes on page 556).
- Changing the value of an attribute within a remote procedure call (see Using Remote Procedure Calls on page 1955 for more information).

If you declare an attribute as a type but do *not* give it a default value, G2 provides one automatically when you complete the edit, by adding an initially is phrase such as this:

```
temperature is a float, initially is 0.0;
maximum-height is an integer, initially is 0
```

The default values that G2 provides for each type are:

Attribute Type	Default Value
float	0.0
integer	0
item-or-value	0.0
quantity	0.0
sequence	sequence( )
structure	structure( )
symbol	G2
text	""
truth-value	true
value	0.0

Defining an Attribute as an Object Instance

You can specify an attribute whose value is an instance of any object class, including a variable or parameter class, by using the is an instance of phrase:

attribute-name is an instance of a[n] *class-name*

For example:

```
heat-sensor is an instance of a thermometer
```

If you do not provide a default value, G2 provides one automatically by adding an initially is an instance of phrase that instantiates the specified class:

```
inner-pressure is an instance of a float-variable,
    initially is an instance of a float-variable;
```

You can give an initially is an instance of phrase that specifies the default value as a more specific class than the type of the attribute requires:

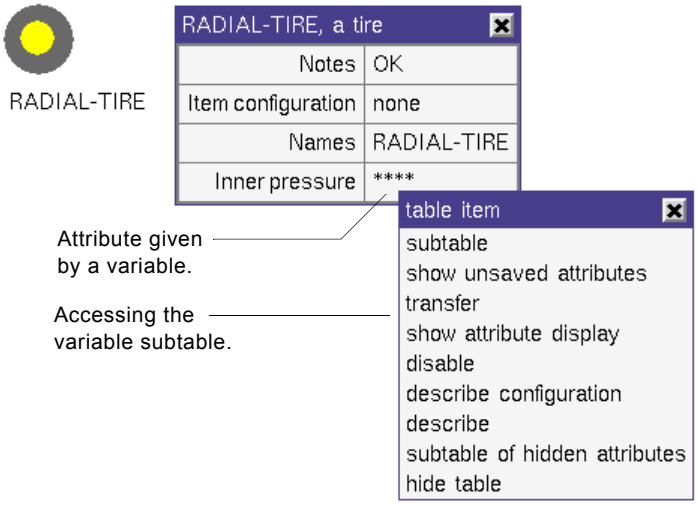
```
outer-pressure is an instance of a quantitative-parameter,
    initially is an instance of an integer-parameter
```

Instantiating a class that has an attribute whose type is an object class actually instantiates two classes: the class itself, and the class specified as the default value of the attribute. The instance of the latter class:

- Becomes the attribute's value in the instance of the defining class.
- Does not have an iconic representation.

- Becomes permanent or transient as the containing item does.
- Can be referenced, accessed, and changed as any object instance can be.
- Is automatically deleted if the containing item is deleted.

The next example shows a tire object, whose inner-pressure attribute is given by a float-variable. The variable is shown in the object attribute table as asterisks (****), because it has no value. Clicking in the value of that attribute and choosing subtable displays the attribute table of the variable.



You can specify the value of an attribute to be an instance of the class that defines the attribute. To prevent an infinite regress, G2 limits the depth to 20 items when you instantiate such a class. The value of the attribute in the most deeply nested instance is none.

Defining an Attribute for Implied Symbolic Reference

Specify an attribute that is to be used with an implied symbolic reference as follows:

```
pc-operating-system has values windows/xp or windows/2000,
    initially is windows/xp
```

Use attributes defined with the has values phrase in expressions that imply an attribute. For example, using the example above for the PC class, once instances of the class exist, you could use an implied attribute reference such as:

```
if PC is windows/xp
```

to reason about the value of the pc-operating-system attribute. The statement specifies the class and a *value*, rather than an attribute name. From this statement, G2 infers that the expression refers to the pc-operating-system attribute, because it is the only attribute that can have the specified value.

Defining an Indexed Attribute

Specify an attribute with an index by using the with an index clause:

```
temp initially is 98, with an index
```

Use indexed attributes when you need an efficient way to locate a particular attribute value among many objects. G2 provides various expressions to use with indexed attributes.

While the Text Editor does not prevent you from entering the with an index clause after any attribute, you can index only attributes that are:

- Defined as the type integer, text, symbol, or truth-value.
- Given by an integer-, text-, symbolic-, or logical-parameter.

For information about using and referring to indexed attributes, see Using Indexed Attributes on page 470.

Formatting Class-Specific Attributes

You can format class-specific attributes of user-defined classes, using one of these format statements, depending on the type of attribute:

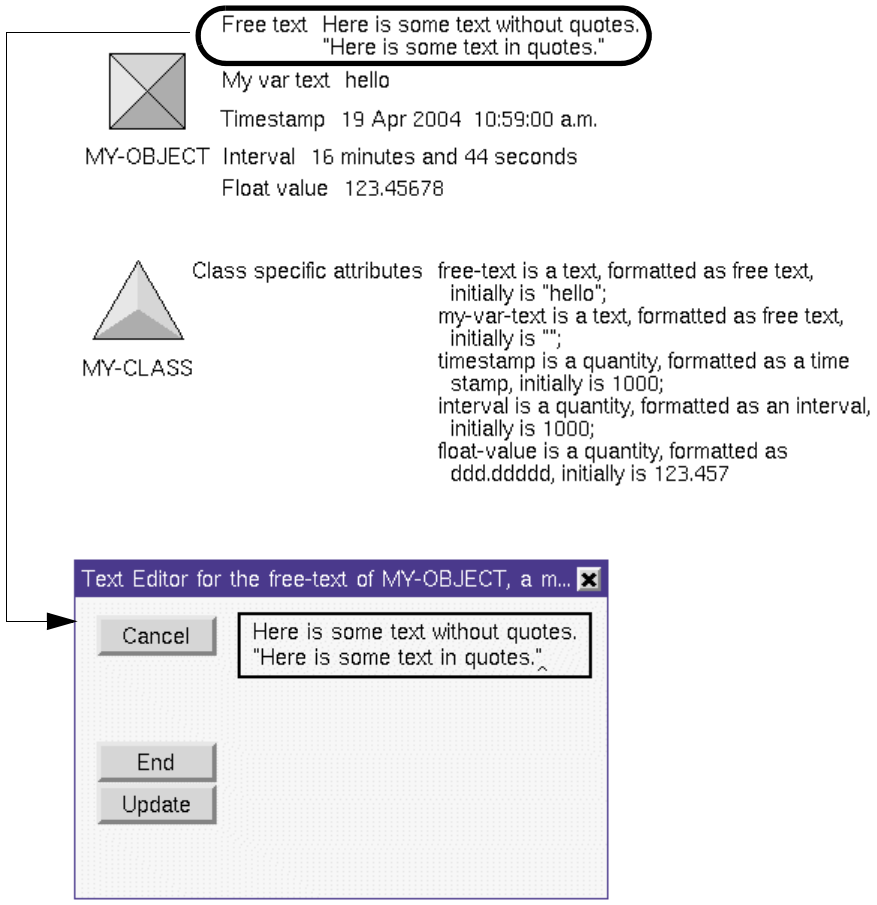
Format Statement	Attribute Type	Description
formatted as free text	text	Displays the attribute without quotation marks, and allows you to enter text without quotation marks in the text editor. To enter quotation marks, simply type the quote character; you do not need to use the escape character (@) to enter a quotation mark.
formatted as a time stamp	quantity	Displays the quantity as a G2 time stamp.
formatted as an interval	quantity	Displays the quantity as a G2 time interval.

Format Statement	Attribute Type	Description
formatted as *ddd.dddd-format*	quantity	Displays the quantity as a floating point number with the specified number of decimal digits to the left and right of the decimal point.
formatted as mm-dd-yyyy-hh-mm-ss formatted as dd-mm-yyyy-hh-mm-ss formatted as yyyy-mm-dd-hh-mm-ss formatted as mm-dd-yyyy-hh-mm-ss-am-pm formatted as mm-dd-yyyy-hh-mm-am-pm formatted as yyyy-mm-dd-hh-mm-ss-am-pm formatted as dd-mm-yyyy-hh-mm-ss-am-pm formatted as dd-mm-yyyy-hh-mm-am-pm formatted as yyyy-mm-dd-hh-mm-am-pm formatted as mm-dd-yyyy-hh-mm formatted as dd-mm-yyyy-hh-mm formatted as yyyy-mm-dd-hh-mm formatted as mm-dd-yyyy formatted as dd-mm-yyyy formatted as yyyy-mm-dd formatted as mm-yyyy formatted as yyyy-mm formatted as dd-hh-mm-ss as an interval formatted as hh-mm-ss as an interval formatted as hh-mm as an interval formatted as mm-ss as an interval formatted as hh.hh as an interval	quantity, float, or integer	Displays the quantity, float, or integer as a date and time format.

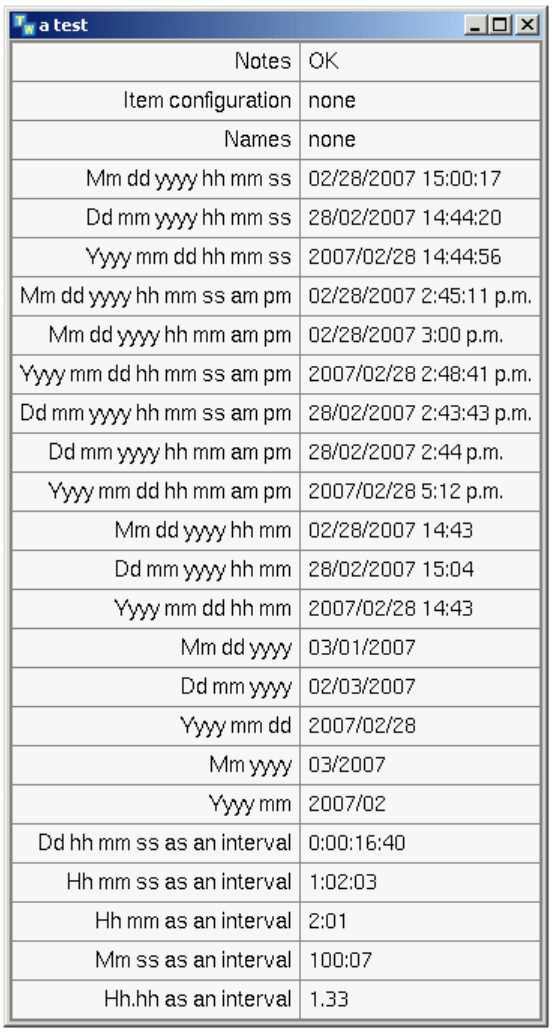
The following example shows four class-specific attributes of a user-defined class and the resulting attribute displays for an instance of the class:

- free-text is a text, formatted as free text, initially is "hello";
- my-var-text is a text, formatted as free text, initially is "";
- timestamp is a quantity, formatted as a time stamp, initially is 1000;
- interval is a quantity, formatted as an interval, initially is 1000
- float-val is a quantity, formatted as ddd.dddd, initially is 123.4567

The resulting textual attribute displays show the text without quotation marks, except where they are entered explicitly, without escape characters, in the text editor. They also show the quantities formatted as a timestamp and G2 time interval.



The following table shows examples of all date and time formats:



Notes	OK
Item configuration	none
Names	none
Mm dd yyyy hh mm ss	02/28/2007 15:00:17
Dd mm yyyy hh mm ss	28/02/2007 14:44:20
Yyyy mm dd hh mm ss	2007/02/28 14:44:56
Mm dd yyyy hh mm ss am pm	02/28/2007 2:45:11 p.m.
Mm dd yyyy hh mm am pm	02/28/2007 3:00 p.m.
Yyyy mm dd hh mm ss am pm	2007/02/28 2:48:41 p.m.
Dd mm yyyy hh mm ss am pm	28/02/2007 2:43:43 p.m.
Dd mm yyyy hh mm am pm	28/02/2007 2:44 p.m.
Yyyy mm dd hh mm am pm	2007/02/28 5:12 p.m.
Mm dd yyyy hh mm	02/28/2007 14:43
Dd mm yyyy hh mm	28/02/2007 15:04
Yyyy mm dd hh mm	2007/02/28 14:43
Mm dd yyyy	03/01/2007
Dd mm yyyy	02/03/2007
Yyyy mm dd	2007/02/28
Mm yyyy	03/2007
Yyyy mm	2007/02
Dd hh mm ss as an interval	0:00:16:40
Hh mm ss as an interval	1:02:03
Hh mm as an interval	2:01
Mm ss as an interval	100:07
Hh.hh as an interval	1.33

Specifying Default Values for Inherited Attributes

The attribute-initializations attribute lets you set default values for all user-defined attributes and many system-defined attributes. System-defined attributes are part of each class in the hierarchy. You cannot change or delete them. Some system-defined attributes, like Notes and Names, appear in every G2 item.

Overriding Default Values of Inherited User-Defined Attributes

The inherited-attributes attribute lists the user-defined attributes that a class inherits from its superior classes. Each of these already has an inherited default value. Using the attribute-initialization attribute, you can override the inherited default value of any inherited user-defined attribute.

The grammar for such initializations differs from the grammar for overriding the default value of a system-defined attribute, which is described under Specifying Default Values of Initializable System-Defined Attributes on page 558.

To specify a default value for an inherited user-defined attribute:

➔ *attribute-name* initially is *default-value*

You can select one or more of the applicable phrases shown in the next figure to specify attribute-initializations. Separate consecutive initializations with a semicolon (;).

If user-defined attribute is...	Then you can...
Without a type, such as: temp	Supply a new default value of your choice, optionally followed by a unit of measure, such as: length initially is 2 feet; temp initially is hot
With a type, such as: temp is a float, initially is 0.0	Supply a new default value of the same type, such as: temp initially is 100.1 If you provide a value of a different type than the inherited attribute, G2 lets you complete the edit. However, a message displays in the notes attribute, indicating that the value will not take effect.
Given by a variable or a parameter, such as: temp is given by a quantitative-variable, initially is given by a quantitative-variable	Supply a new default value only if the value is a subclass of the inherited class of the attribute, such as (for the example on the left): temp initially is given by an integer-variable
Initially an instance of an object subclass, such as: temp initially is an instance of a float-array	Change the instance of one object to any other allowable object, such as: temp initially is an instance of a symbol-list

As part of the capability to create and complete definitions in bottom-up order, G2 allows you to specify user-defined attribute initializations for currently non-existent inherited attributes. When such an initialization exists, the definition's notes attribute displays a message such as the following:

TIRE, a class-definition	
Notes	OK, and note that the initialization for material has no corresponding inherited attribute and cannot be implemented
Authors	ghw (5 Jun 2000 3:34 p.m.)
Change log	0 entries
Item configuration	none
Class name	tire
Direct superior classes	object
Class specific attributes	inner-pressure is given by a float-variable, initially is given by a float-variable
Instance configuration	none
Change	none
Instantiate	yes
Include in menus	yes
Class inheritance path	tire, object, item
Inherited attributes	none
Initializable system attributes	attribute-displays, stubs
Attribute initializations	material initially is rubber

Specifying Default Values of Initializable System-Defined Attributes

The initializable-system-attributes attribute lists the names of all initializable attributes that the definition inherits from the system-defined class(es) in its list of direct superiors. Each of these already has an inherited default value. Using the attribute-initialization attribute, you can override the default value of any initializable system-defined attribute.

For example, you can specify the system-defined attributes array-length and the element-type for an array definition in attribute-initializations. These two attributes appear on the attribute table of an array.

The grammar for such initializations differs from the grammar for overriding the default value of a user-defined attribute, which is described under Overriding Default Values of Inherited User-Defined Attributes on page 556.

To specify a default value for an initializable system-defined attribute:

➔ *attribute-name*: *default-value*

Separate adjacent initializations with a semicolon (;). The following definition for a subclass of procedure specifies default values for the initializable system attributes tracing-and-breakpoints and default-procedure-priority:

DISTRIBUTION-PROCEDURE, a class-definition

Notes	OK
Authors	ghw (5 Jun 2000 3:42 p.m.)
Change log	0 entries
Item configuration	none
Class name	distribution-procedure
Direct superior classes	procedure
Class specific attributes	none
Instance configuration	none
Change	none
Instantiate	yes
Include in menus	yes
Class inheritance path	distribution-procedure, procedure, item
Inherited attributes	none
Initializable system attributes	tracing-and-breakpoints, class-of-procedure-invocation, default-procedure-priority, uninterrupted-procedure-execution-limit, attribute-displays, stubs
Attribute initializations	tracing-and-breakpoints: breakpoint level 3 (breakpoints at every step); default-procedure-priority: 8
Icon description	inherited

Initializations of inherited user-defined attributes and initializable system-defined attributes can be intermixed as desired. Separate the adjacent initializations with a semicolon (;). Be careful not to confuse the different grammars of the two types of initialization.

For information on the initializable system attributes of any class, see the documentation in this manual for that class. For additional information on the initializable system attributes of:

- Objects, including variables, parameters, arrays, and lists, see Creating Object Classes on page 564.
- Connections, see Creating Connection Classes on page 577.
- Messages, see Creating Message Classes on page 582.

Specifying Instantiability

Instantiation applies to all user-defined classes except subclasses of connection. Instantiability is controlled by two definition attributes: instantiate and include-in-menus. These two attributes are collectively called the **instantiation attributes**. When you create a new definition, the default value of both instantiation attributes is yes.

To specify that a class is/is not instantiable:

➔ Set the class's instantiate attribute to yes/no.

To specify that a class does/does not appear in G2 menus:

➔ Set the class's include-in-menus attribute to yes/no.

Effects of Setting Instantiability Attributes

When you add a class to the class hierarchy, its name always appears in the list of available classes (via the direct-superior-classes attribute), and in the class hierarchy schematic available through the Inspect facility (show on a workspace the class hierarchy).

Other than that, the accessibility of the class depends on the values of its instantiability attributes. The effects and interaction of these attributes are summarized in this table:

		include-in-menus	
		yes	**no**
instantiate	**yes**	The class can be instantiated programmatically and appears in G2 menus, allowing it to be instantiated interactively.	The class can be instantiated programmatically, but does not appear in G2 menus, so it cannot be instantiated interactively.
	no	The class cannot be instantiated, but appears in G2 menus where necessary to permit navigation to a subclass that is instantiable and appears in menus.	The class cannot be instantiated and does not appear in any G2 menu. The class may have subclasses that are instantiable and/or appear in menus.

Instantiable Classes That Appear in Menus

Whenever you create a new class, G2 adds the class to the class hierarchy. If the class is instantiable and appears in menus, the G2 menu of the class definition then includes the choice create instance. Choosing create instance instantiates the class.

G2 also adds the class to the G2 menu hierarchy. For instance, an object class appears at some level under KB Workspace > New Object, and a subclass of kb-workspace appears at some level under Main Menu > New Workspace.

When a class that is instantiable and appears in menus has a subclass that is also instantiable and appears in menus, the higher-level class appears more than once in the menu hierarchy:

- In a leaf menu to permit instantiation of the class.
- In higher-level menus as needed to permit navigation to the leaf menu.

Creating a subclass with more than one direct superior class adds the new subclass to the menu structure of each of the superior classes. For instance, creating a subclass of object and g2-to-g2-interface called g2-to-g2-object, adds the g2-to-g2-object class to both the new object and the g2 to g2 interface submenus.

Noninstantiable Classes That Appear in Menus

A class that is noninstantiable does not include create instance in its G2 menu, but it appears as a higher-level entry in the menu hierarchy *if and only if* it is an ancestor of a class that is instantiable and appears in menus. Its appearance at the higher level facilitates navigating the menu hierarchy to the instantiable class.

Classes That Do Not Appear in Menus

Such a class does not include create instance in its G2 menu, and does not appear in the menu hierarchy even if has an instantiable subclass that appears in menus. The instantiable subclass appears in the hierarchy as a subclass of the first of its ancestors that does appear in menus, or perhaps of more than one such ancestor if it has multiple inheritance.

Order of Classes in the G2 Menu Hierarchy

There is no particular order to the way in which G2 adds new classes to menus, nor is the order of display permanent across G2 sessions. User-defined classes may appear on the menus before system-defined classes, or after them, and then appear in a different order after you start a new G2 session.

Uninstantiable Subclasses

In order to be instantiable, a class must inherit at least one instantiable class. When a class is not instantiable because its system inheritance does not include an instantiable class, and the class's instantiate attribute is yes, G2 ignores the value of the attribute, and includes a Note that states:

```
note that the value of instantiate? should be changed to "no"
    because system class inheritance precludes instantiation
```

Specifying an Icon

Almost every G2 class has an iconic representation. The only exceptions: connection, message, and a few others for which an icon is obviously inappropriate. For example, a kb-workspace has no icon: its visual representation is the workspace itself. When a class has an iconic representation, the icon-description attribute of its definition specifies the icon that represents an instance of the class. The default value of icon-description is inherited.

The icon-description attribute is a special case. It could appear as an initializable system-defined attribute, but an icon description can be very long and complex: including it with other attribute initializations could make the other attributes difficult to access. Making icon-description a separate attribute prevents such problems, with the side-effect that the attribute appears in every definition whether or not the defined class has an iconic representation. When a class has no iconic representation, the icon-description attribute of its definition cannot be edited and has no effect.

System-Defined and User-Defined Icons

G2 provides a system-defined icon for each type of item that has an iconic representation. For instance, if you specify logical-parameter as one of the direct-superior-classes, and no other icon has been defined through another class, the icon for class instances will look like this:

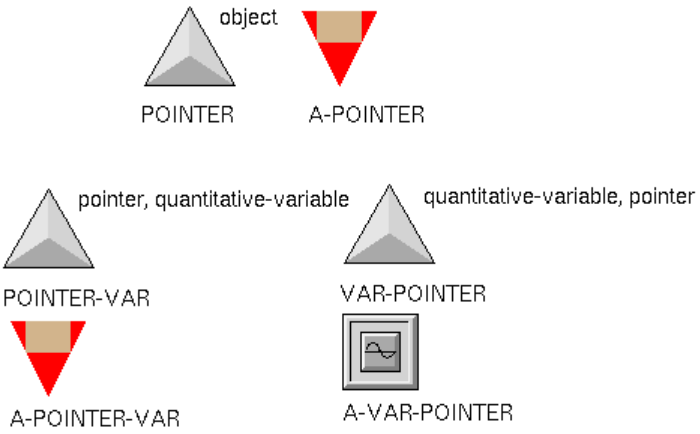
You can replace the default, system-defined icon with your own design by editing the icon in the Icon Editor. Most user-defined icons represent items used for knowledge representation. For example, you may wish to draw a car icon to represent automobile objects, a book icon for a help object, or a rocket icon for space-vehicle objects. You can also define icons for user-defined subclasses of G2 components, such as procedures or image definitions.

Icon Inheritance

The icon that a class inherits depends on its class inheritance path. For a detailed explanation of the class inheritance path, see Chapter 13, Classes and Class Hierarchy on page 497.

A class inherits the first explicitly defined (rather than inherited) icon-description that G2 locates on the class inheritance path, which may be a user- or a system-defined icon. The next diagram presents an example of a class inheriting a user-defined icon, or a system-defined icon, depending on which way the direct superior classes are specified.

The diagram shows a pointer superior class, with an instance, a-pointer. The two other definitions, pointer-var and var-pointer, display their direct superior classes attribute. Both of these classes have instances with different icons, illustrating how an icon is inherited through the class inheritance path.



Using the Icon Editor

The icon-description attribute describes an icon textually, using a graphical description language. However, icons are rarely specified by typing in their descriptions.

To create a new icon for a definition:

➔ Use the Icon Editor to define the icon graphically.

When you save from or close the Icon Editor, G2 automatically replaces the textual description in the icon-description attribute with a description corresponding to the icon you defined.

For a complete description of the Icon Editor, and of the text syntax for the icon-description attribute, see Chapter 46, The Icon Editor and Icon Management on page 1637.

Creating Object Classes

Objects represent physical items, (such as a tank, a pump, or a car), abstract ideas and concepts, relations, variables, parameters, lists, and arrays. The G2 class hierarchy provides:

- The system-defined class object, for use in defining classes of objects.
- Various system-defined subclasses of object that provide variables, parameters, lists, arrays, and many other things.

This section refers to object and any system-defined or user-defined subclass of object as an **object class**.

To create an object class:

1 Create a class-definition item whose primary direct superior is object class or any subclass of object class.

For details, see:

- Creating Class Definitions on page 538
- Class Definition Attributes on page 539
- Configuring Class Definitions on page 542
- Specifying Instantiability on page 560
- Specifying an Icon on page 562

2 Provide additional information as described in this section.

System-Defined Object Attributes

Two system-defined attributes (in addition to icon-description) exist in every object class:

- attribute-displays
- stubs

Attribute	Description
attribute-displays	The attribute displays of this class
Allowable values:	any system-defined attribute name any user-defined attribute name inherited none
Default value:	inherited
stubs	The stubs that a class specifies or inherits.
Allowable values:	{a | an} [input | output] connection-class [portname] located at [top | bottom | right | left] integer [with style {diagonal | orthogonal}] [with line-pattern { solid | dot | fine dot | coarse dot | dash | short dash | long dash | { [*pattern*,] ... } { [on *integer*, off *integer*] ... [, not scaled by line width] } }] inherited none
Default value:	inherited

When you create a class-definition that inherits any object class, the attribute-displays and stubs attributes appear as initializable system attributes in the definition. The grammar for providing their default values is described under Specifying Default Values for Inherited Attributes on page 556. For details, see:

- Specifying Attribute Displays on page 566.
- Specifying Connection Stubs on page 567.

Each system-defined subclass of object class adds the initializable system attributes that it needs in order to carry out its purpose. Some of these are described in this section under:

- Attribute Initializations for Variables and Parameters on page 575.
- Attribute Initializations for Lists and Arrays on page 576.

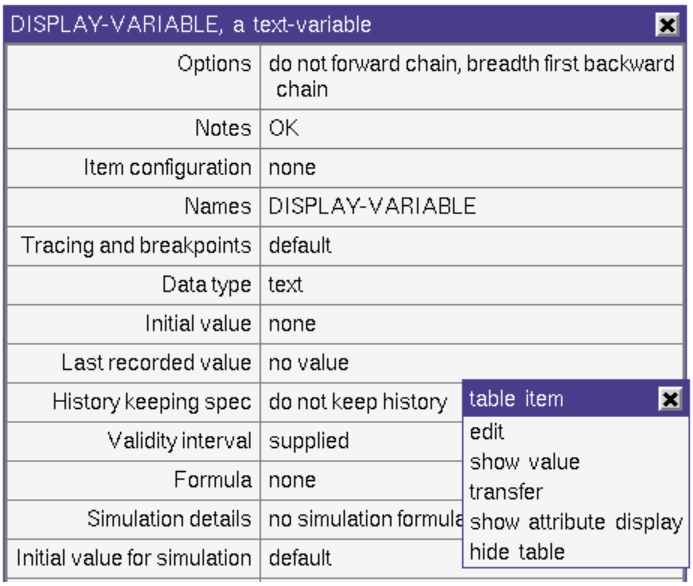
One type of object class, the connection-post class, is functionally more closely associated with connections than with objects, which are typically used to represent knowledge. Instructions for creating connection posts appear under Creating Connection Post Classes on page 581.

For information on other initializable system attributes of object classes, see the documentation in this manual for the particular class.

Specifying Attribute Displays

The attribute-displays attribute lets you display the values and, optionally, the names of, one or more system- or user-defined attributes of class instances.

Most attribute tables in G2 have a show attribute display option in their table menu, as shown here on the history-keeping-spec attribute of a variable.



While you can display most attribute values by choosing that option, the attribute-displays attribute in a definition provides additional functionality.

In addition to displaying the attribute value, the attribute-displays attribute lets you optionally display the attribute name, and position the display at a location of your choice.

To define the attribute-displays attribute, enter the name of any system- or user-defined attributes, and select one or more of the phrases that appear in the Text Editor.

Separate attribute displays with a semi-colon (;). Here is how to specify attribute displays:

To display the attribute...	Enter the value like this...
Value (with or without the attribute name) at the standard position	batch-number at standard position; mileage with name at standard position Displaying only the attribute value at the standard position is the default setting for attribute displays. The standard position aligns with the top edge of the object to the right-hand side. The with name statement displays the attribute name along with its value.
Value (and name) at a non-standard position	batch-number with name offset by (10, 10) where (10, 10) are x and y coordinates that define the location of the attribute display as an offset from the center of the icon.

When you specify attribute displays, instances of the object include those displays at creation time and remain in effect until you change them in the attribute table of the instance. Alternatively, you can use the change attribute in the class definition to update the attribute displays of all instances. For information about changing attribute displays with the Change attribute, see Using the Change Attribute on page 592.

Specifying Connection Stubs

Connection stubs are short connections attached to and extending from an object. They are the starting point for connections between objects. This figure shows an object with stubs extending to the left and right:

All stubs have the qualities of elasticity, style, and line pattern. You can lengthen a stub by dragging the stub end that is not attached to the object or item. In style, a stub is either orthogonal or diagonal. Its line pattern can be solid, dotted, dashed, or a combination.

The stubs attribute indicates the number of stubs and the location of where each attaches to class instances.

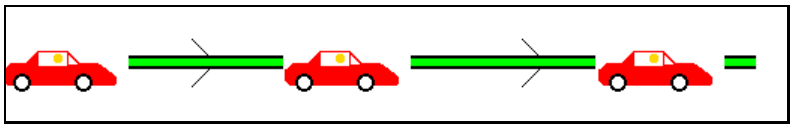
Use this syntax to specify stubs:

```
{none | inherited | {a | an} [ {input | output} ] connection-class
  [portname] located at {top | bottom | right | left} integer
  [with style {orthogonal | diagonal} ]
  [with line-pattern { solid | dot | fine dot | coarse dot |
                       dash | short dash | long dash |
                       { [pattern,] ... }
                       { [on integer, off integer] ... [, not scaled by line width] } } ] }
```

where:

- *pattern* is one of: dot, fine dot, coarse dot, dash, short dash, long dash.
- *integer* is the number of workspace units to make visible (on) or invisible (off).
- not scaled by line width causes the custom specification to use the actual number of specified workspace units, rather than scaling them by the width of the connection.

A stub can optionally have a direction. The next diagram shows a particular connection style, and its direction, indicated by arrows:



By default, stubs are non-directional and associated with the system-defined connection class. For a complete description of using connections, see Chapter 18, Connections on page 703.

You can assign a name to a stub, and reference it by that name in expressions. A named stub is called a **port**. Specify a port's name by including a *portname* in the stubs attribute that specifies the stub.

You can indicate direction in a stub by specifying it to be an input or output stub in the stubs attribute. Providing direction to a stub gives you more control over connections made to it, such as restricting the direction of flow. For example, G2 does not let you connect an input stub to another input stub, or an output stub to another output stub. When a connection is directional, G2 can reason about the objects it connects this way:

```
any auto-object connected at an input of auto-object2
```

The example shows the stub specification for the stubs in the previous diagrams. You can specify multiple stubs in the stubs attribute:

```
an output movement-connection located at right 5;
an input movement-connection located at left 5
```

You can optionally provide a portname in the stub specification, which you can also refer to in expressions this way:

```
any auto-object connected at the outflow-port of auto-object2
```

By assigning a connection class to a stub, the stub has the same visual properties of that class. You can then provide expressions that could, for example, change a particular stripe color of the connection this way:

```
whenever the mileage of auto-object > 100
    change the inside-stripe stripe-color of every production-line-connection to red
```

By default, a stub appears as a single black line extending from the object or item. The following table shows techniques for specifying stubs:

To specify stubs with...	Enter a statement such as this...
Direction, either input or output, at a specific location	an input connection located at right 10; an output connection located at bottom 25 The location can be top, bottom, right, or left, followed by a positive integer specifying the position on the object icon in workspace units
A style	an input connection with style diagonal Specifies whether the connection is orthogonal or diagonal. The default is orthogonal, indicating that the connection can be drawn using only straight and right angles. Diagonal specifies that the connection can be drawn using straight or diagonal lines with an angle. Stubs with a diagonal style are drawn with the default single black line.
A line pattern	an input connection with line-pattern dot Specifies a solid, dashed, or dotted line pattern, or a combination. See Specifying Line Patterns on page 570.
A connection-class and a portname	an input *connection portname* Specifies the name of the connection class. You can specify a connection class that does not yet exist, but instances will not have stubs unless the connection class exists. The *portname* that you enter is the name of the port where a connection attaches to an object.

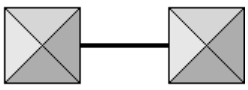
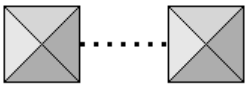
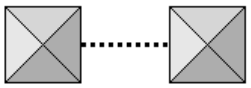
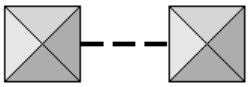
Note G2 does not restrict connections to the stubs you define. Unless you specify no-manual-connections as a Configuration in the class definition (which limits connections to the stub locations), you can make any number of additional connections to an object.

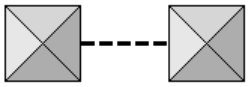
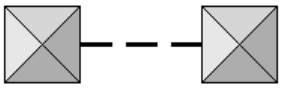
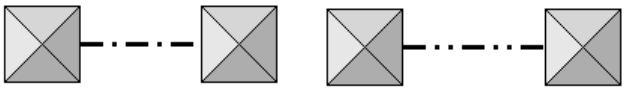
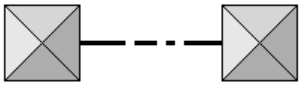
Specifying Line Patterns

You can configure the stubs attribute of a class definition to initialize the line-pattern, in the same way that you can initialize the connection-style. You can also configure this attribute for individual connection instances, in the same way that you can configure the connection-style.

Once a connection exists, you can either change the text of the line-pattern attribute or conclude a value for the line-pattern attribute, using the attribute access facility.

This table shows the options for line-pattern and an example of each, using a connection width of 3 workspace units:

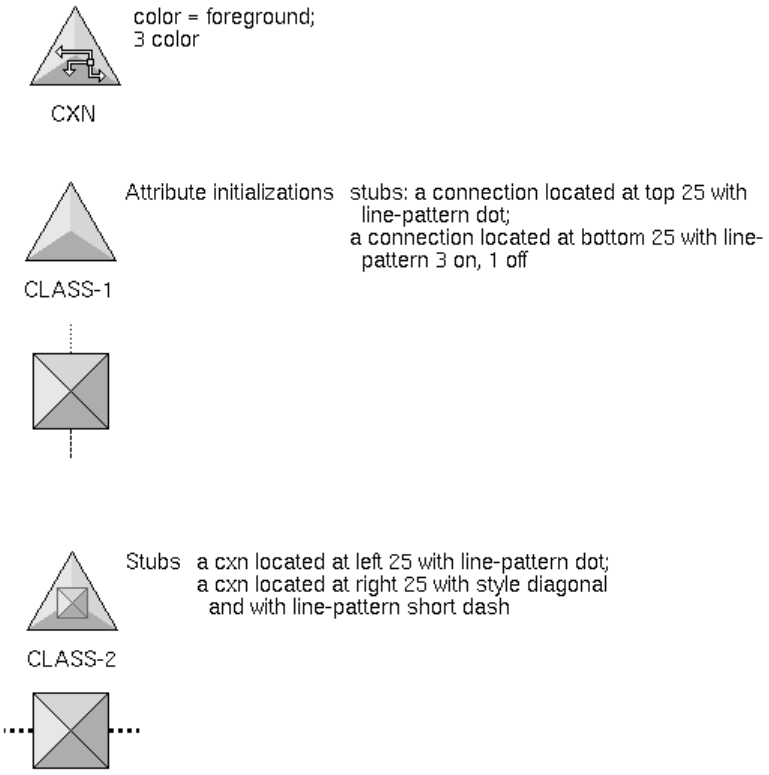
Line-Pattern	Example (Width = 3)
solid (the default)	
dot	
fine dot	
coarse dot	
dash	

Line-Pattern	Example (Width = 3)
short dash	
long dash	
Any combination of the above, for example, dash, dot or dash, dot, dot	
A custom specification of the number of workspace units that are on and off, scaled by the line width, for example, 10 on, 2 off, 5 on, 2 off, 2 on, 2 off	
A custom specification of the number of workspace units that are on and off, not scaled by the line width	

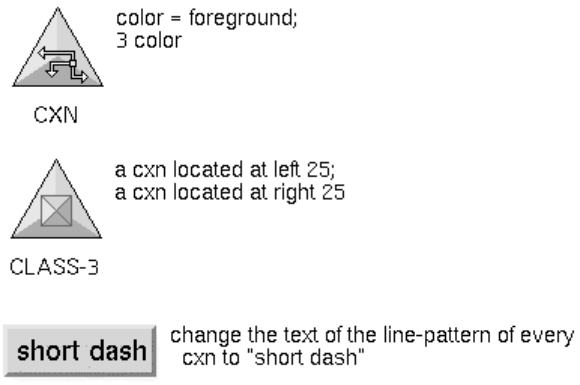
Here are some examples of the with line-pattern syntax:

```
with line-pattern dot
with line-pattern fine dot
with line-pattern coarse dot
with line-pattern dash
with line-pattern short dash
with line-pattern long dash
with line-pattern dash, dot
with line-pattern dot, dot, dash
with line-pattern on 2, off 5
with line-pattern on 5, off 3, on 10, off 6
with line-pattern on 2, off 5, not scaled by line width
```

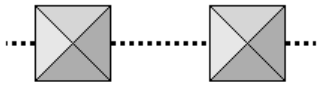
The following figure shows two examples of how to initialize the line-pattern attribute of a connection of type cxn. class-1 is a class definition that configures the attribute-initializations for the stubs attribute. class-2 is an object definition that configures the stubs attribute directly.



This figure shows how to configure the line-pattern attribute of individual connection instances, using an action button:



Here is the resulting connection and its table:



a cxn	
Notes	OK
Item configuration	none
Names	none
Connection style	orthogonal
Line pattern	short dash

This table shows various examples of how to use the attribute access facility to conclude values for the line-pattern attribute of a connection named cxn:

Line-Pattern Attribute Value	Concluding the Line-Pattern Attribute
dot	conclude that the line-pattern of cxn = the symbol dot
dash	conclude that the line-pattern of cxn = the symbol dash
fine dot	conclude that the line-pattern of cxn = the symbol fine-dot
coarse dot	conclude that the line-pattern of cxn = the symbol coarse-dot
short dash	conclude that the line-pattern of cxn = the symbol short-dash
long dash	conclude that the line-pattern of cxn = the symbol fine-dot
dash, dot	conclude that the line-pattern of cxn = (sequence (the symbol dash, the symbol dot))
dot, dot, dash	conclude that the line-pattern of cxn = (sequence (the symbol dot, the symbol dot, the symbol dash)

Line-Pattern Attribute Value	Concluding the Line-Pattern Attribute
on 2, off 5	conclude that the line-pattern of cxn = (sequence (2,5))
on 5, off 3, on 10, off 6	conclude that the line-pattern of cxn = (sequence (5,3,10,6))
on 2, off 5, not scaled by line width	conclude that the line-pattern of cxn = structure (pattern: sequence (2, 5), not-scaled-by-line-width: true)

Inheriting Default Values for Stubs

Icons and stubs are closely related, but they are specified by two different attributes of a definition. Where multiple inheritance exists, this independence could result in mismatched icons and stubs, so the G2 class inheritance rules contain a special provision that prevents it.

To prevent such a mismatch, a class can inherit a stubs definition *only* from the class from which it inherits its icon definition, or from a descendent of that class. If none of these provides a stubs definition, either inherited or locally defined, the class's stubs default value is none.

Specifying Other Object Class Attributes

Each system-defined subclass of object adds the initializable system attributes that it needs in order to carry out its purpose. This section describes the attributes of variables, parameters, lists, and arrays. For information on other initializable system attributes of object classes, see the particular class.

Attribute Initializations for Variables and Parameters

The attribute-initializations applicable to variables and parameters are as follows:

Attribute-initialization	Variable	Parameter
data server	any data-server-alias inference engine g2 simulator g2 meter, g2 data server gfi gsi GFI and the G2 Simulator are superseded capabilities. For further information see Appendix F, Superseded Practices on page 2169.	N/A
data type (for quantitative-variable, or quantitative-parameter)	quantity time-stamp pure-number	quantity time-stamp pure-number
default update interval	any non-negative time interval, including subsecond values	N/A
history-keeping spec	[do not] keep history with maximum number of data points = *any integer* and maximum age of data point = *time-interval*, with minimum interval between data points = *any non-negative-number time-interval*	[do not] keep history with maximum number of data points = *any integer* and maximum age of data point = *time-interval*, with minimum interval between data points = *any non-negative-number time-interval*
initial value for initial values for	any legal value for type of variable	any legal value for type of parameter

Attribute-initialization	Variable	Parameter
options for	do [not] forward chain; do [not] seek data; do [not] backward chain; depth first backward chain; breadth first backward chain;	do [not] forward chain
validity interval	*any integer* (not a float) indefinite supplied	N/A

You can also select supply simulation subtable as a default setting for a variable definition. Instances will then include a simulation subtable available from the variable's simulation-details attribute.

The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices on page 2169.

Note For more information about specifying history keeping for variables and parameters, see Chapter 15, Variables and Parameters on page 607.

Attribute Initializations for Lists and Arrays

The attribute-initializations applicable for lists and arrays are as follows:

Attribute-Initialization	Lists	Arrays
allow duplicate elements for g2-list	{yes | no}	Not applicable
array-length for g2-array	Not applicable	Any non-negative integer
element type for	item-list (any class)	item-array (any class)
initial values for	Not applicable	Any legal item or value for the type of array. If the *number* of default values disagrees with the array length, G2 assigns the default array value to all elements.

Note You cannot change the allow duplicate elements for a g2-list or the element type for a g2-array while G2 is running when instances exist.

For array classes of a specific type, such as an integer-array, the Text Editor does not prevent you from entering elements of the incorrect type. Specifying such elements causes G2 to replace any element of an incorrect type with the default value for the array class.

Creating Connection Classes

Connections are graphical items that embody logical relationships. You can use connections to represent almost anything that provides a pathway or route between two or more objects. For information about using connections, see Chapter 18, Connections on page 703.

The G2 class hierarchy provides the system-defined class connection for use in defining connections. This section refers to connection and any system-defined or user-defined subclass of connection as a **connection class**. You can create a connection class that has the properties you need. For example, you may wish to create a connection class that displays as a green stripe with a black border.

Connections use stubs on objects (described under Specifying Connection Stubs on page 567), **junction blocks**, and **connection posts**, both described in this section.

To create a connection class:

1. Create a class definition whose primary direct superior is connection or any subclass of connection, following the directions under:
   - Creating Class Definitions on page 538
   - Class Definition Attributes on page 539
   - Configuring Class Definitions on page 542
2. Provide additional information as described in this section.

Instantiability does not apply to connections. G2 ignores the values of a class definition's instantiability attributes when the primary direct superior is a connection class.

System-Defined Connection Attributes

Three initializable system-defined attributes exist in every connection class:

- cross-section-pattern
- stub-length
- junction-block

Attribute	Description
cross-section-pattern	The style and appearance of the connection.
Allowable values:	Described in Defining Connection Regions on page 578.
Default value:	none
stub-length	The default length of the connection of this class, in workspace units.
Allowable values:	any positive integer
Default value:	inherited
junction-block	The name of the junction-block class that this connection shall use.
Allowable values:	Any junction-block class.
Default value:	none

When you create a class definition that inherits any connection class, these three attributes appear as initializable system attributes in the definition. The grammar for providing their default values is described under Specifying Default Values for Inherited Attributes on page 556. The rest of this section shows you how to specify values for the attributes.

Defining Connection Regions

The cross-section-pattern attribute lets you define connection regions to which you can assign a width and a color. When connection regions exist, you can refer to them in expressions. For example, during the execution of a KB, you could change the color of one or more connection regions to signal changes in events and status.

Also, completing the cross-section-pattern attribute automatically creates a corresponding default-junction class.

Hint While it is not a requirement, we recommend that you define connection cross-section patterns with a symmetrical design.

Here is how to specify the cross-section-pattern attribute:

To specify...	Enter a statement such as this...
Connection regions	*region1* = color, *region2* = color; Where you specify one or more regions (*region1* and *region2*) as any unreserved symbol in G2 and assign each a color. Enter as many regions as necessary. An example is: outside-wire = black, electrical-flow = green
The size of the regions	2 outside-wire, 6 electrical-flow, 2 outside-wire Enter the size of each region as a positive integer (in workspace units). This example creates a connection with a black border and a green stripe.

Here is an example of a cross-section-pattern attribute value:

```
outside-wire = black, electrical-flow = green;
2 outside-wire, 5 electrical-flow, 4 outside-wire
```

Once region names exist for a connection class, you can refer to them in expressions by using this syntax:

change the *region-name* stripe-color of *connection-class* to *color*

An example is:

```
change the electrical-flow stripe-color of my-connection to red
```

Specifying a Stub Length

The stub-length attribute specifies the length of stubs as an integer in workspace units.

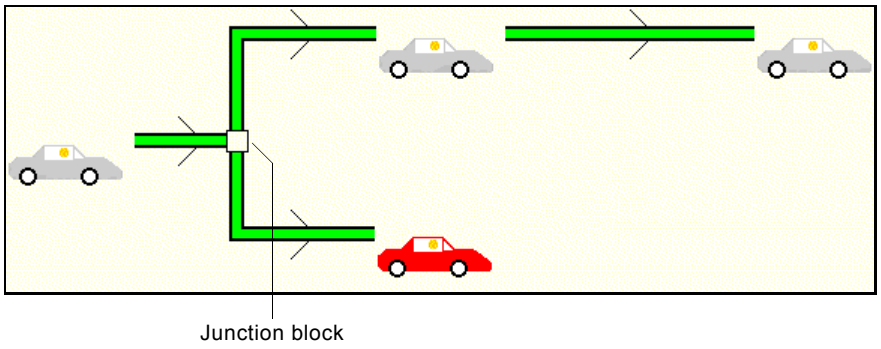
While an object class definition defines the class, location, and direction of the connection flow, the stub-length attribute specifies the default length of the stub on a class instance. The default length is 20 (workspace units).

Defining the Junction Block to Use

The junction-block attribute defines which junction block a connection will use.

A connection is drawn from point-to point, directly from the stub of one object to the stub (or any available location) of another object. To terminate a connection at another connection (rather than at an object), you can use a **junction block**. In that way, two incoming connections can be joined to an input stub or two inputs fed

from an output stub. The junction-block attribute indicates the class of junction block to use at the intersection of two connections. Here is a junction block:



Whenever you complete the cross-section-pattern attribute, G2 automatically creates a corresponding junction-block class with the name of the connection class preceded by junction-block-for-. For example, if the connection name is water-line, and you change the cross-section-pattern attribute, G2 creates a junction block for that connection with the name:

junction-block-for-water-line

A junction-block is a default-junction subclass, which is an abstract object class. When G2 creates a new junction-block class automatically, the new class does not appear on the hierarchy of menus, but you can create an instance of it programmatically, or whenever you terminate connections.

If a connection class inherits a cross-section-pattern, G2 does not create a new junction block class for the subclass automatically. However, if you edit the cross-section-pattern attribute and choose copy inherited path, G2 does two things:

- Displays the cross-section-pattern attribute exactly as it was specified in the superior class.
- Creates a new junction-block class dynamically with the naming convention noted above.

Hint While you can create a new junction block class interactively (by creating a new object class definition using default-junction as the Direct-superior-class), we recommend that you do not. G2 creates a a junction-box class *dynamically* any time you specify a cross-section-pattern in a connection definition.

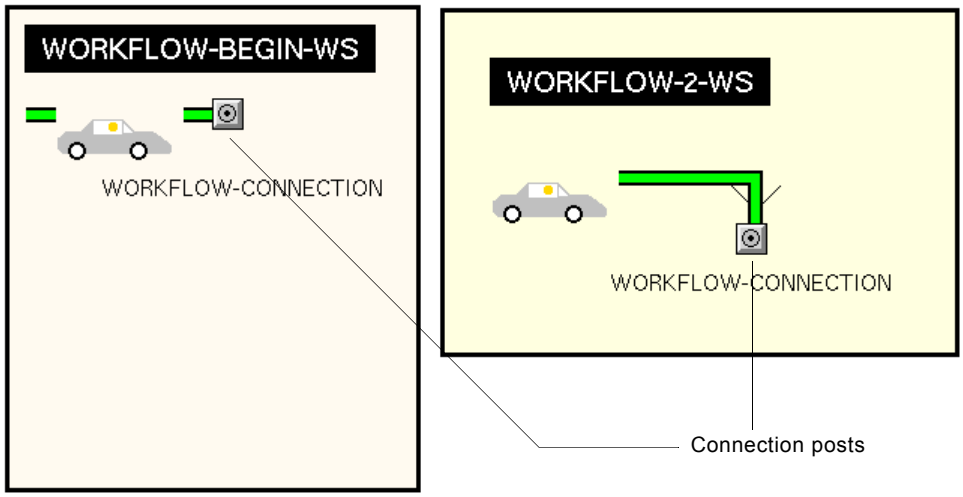
Creating a Junction-Block Subclass

If it is necessary to create a junction block subclass, create a new object class definition and specify default-junction as the direct superior class.

By default, the icon for a default-junction subclass is a small grey square.

Creating Connection Post Classes

Connection posts are objects that connect objects across workspaces by indicating that the endpoints of connections on separate workspaces are joined. All connection posts of the same name are connected to each other. The next figure shows connection posts for connections on two separate workspaces.



If the connection posts have the same name, the two cars are functionally connected to each other, just as if they existed and were connected on the same workspace. For information about using connection posts, see Using Connection Posts on page 717.

The G2 class hierarchy provides the system-defined class connection-post for use in defining connections posts. In most cases, that class is sufficient, but you can create subclasses of it that have the properties you need. This section refers to any subclass of connection-post as a **connection-post class**.

To create a connection-post class:

1. Create a class definition whose primary direct superior is connection-post or any subclass of connection-post, following the directions under Creating Object Classes on page 564.
2. Provide additional information as described in this section.

System-Defined Connection Post Attribute

One initializable system-defined attribute exists in every connection post class, in addition to those characteristic of every object class:

Attribute	Description
superior-connection	The stub (if any) to which the connection post connects.
Allowable values:	A portname or a specified location.
Default value:	none

When you create a class definition that inherits any connection class, this attribute appears as an initializable system attributes in the definition. The grammar for providing its default values is described in Specifying Default Values for Inherited Attributes on page 556. The rest of this section shows you how to specify values for this attribute.

Specifying the Superior Connection

The superior-connection attribute lets you indicate a particular stub on an object to which the connection post is connected. Use this attribute when a connection post is on the subworkspace of an object and you want to specify that it is connected to a specific port or location on the object. Specify the object connection as either a portname or a specific location, using this syntax:

the connection {at *portname* | located at [top | bottom | right | left] *integer*}

The connection specified in this attribute is in addition to other connection posts of the same name with which the connection post is associated. An example is:

```
the connection located at right 5
```

You can specify the statement with a location, as shown here, or a portname. The default value is none.

Creating Message Classes

A message is an item that displays text. Messages provide information to the user. For example, as a result of an inform or post action for the operator, G2 creates and displays a message on the message board. For information about using messages, see Chapter 35, Messages on page 1227.

The G2 class hierarchy provides the system-defined class message for use in defining messages. In many cases, that class is sufficient, but you can create subclasses of it that have the properties you need. This section refers to any subclass of message as a **message class**.

For example, you could create a message class warning-message, which displays in red with large type. You could then use such a message class only for displaying warning messages.

To create a message class:

1 Create a class definition whose primary direct superior is message or any subclass of message, following the directions under:

- Creating Class Definitions on page 538
- Class Definition Attributes on page 539
- Configuring Class Definitions on page 542
- Specifying Instantiability on page 560

2 Provide additional information as described in this section.

System-Defined Message Attribute

One initializable system-defined attribute exists in every message class:

Attribute	Description
default-message-properties	The style of each attribute of this message class.
Allowable values:	Described in the next section.
Default value:	none

When you create a class definition that inherits any message class, this attribute appears as an initializable system attributes in the definition. The grammar for providing its default value is described under Specifying Default Values for Inherited Attributes on page 556. The rest of this section shows you how to specify values for this attribute.

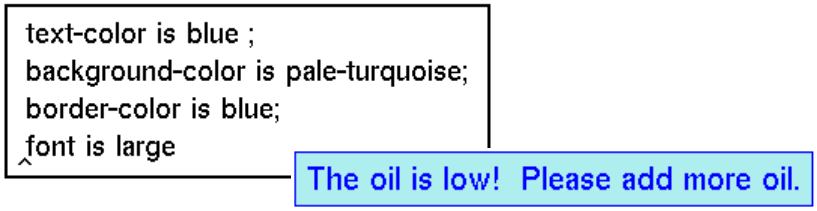
Specifying Default Message Properties

The default-message-properties attribute controls how instances of a message class appear. For instance, if you want a message class to have a beige background color and brown type, you specify that information in this attribute. Here are the message properties you can specify:

This message property...	Specifies...
background-color	The background color of the message. The default is transparent.
border-color	The color of the message border. The default is foreground.
fonts	The font size for the text. Available font sizes are small, large, and extra-large. The default is large.
minimum-width	The minimum width of the message in workspace units that you enter as an integer value. If you do not specify a value for this attribute, G2 makes the message width the size of the text. The default is none.
minimum-height	The minimum height of the message in workspace units that you enter as an integer value. If you do not specify a value for this attribute, G2 makes the height the size of the text. The default is none.
text-color	The color of the message text. The default is foreground.
text-alignment	The alignment of the message text. Available alignments are left, right, and center. The default is left.

Note Entering values for the minimum-width and minimum-height message properties does not center the message text within those dimensions when an instance exists.

Here is an example of a completed default-message-properties attribute and a message of that message class:



Using Specialized Definitions

In addition to the class-definition class, which can define a subclass of any extensible class, G2 provides three specialized types of definitions:

- An object-definition creates a subclass of object.
- A connection-definition creates a subclass of connection.
- A message-definition creates a subclass of message.

These specialized definitions are supported to provide compatibility with previous versions of G2, which did not provide a generic class definition capability. This section briefly describes the use of specialized definitions for those who encounter them in existing G2 applications.

The specialized definitions are very similar to class-definitions. The only difference is that each one provides as ordinary table attributes the attributes characteristic of every class of its type. In a class-definition, these attributes appear as initializable-system-attributes after the direct-superior-classes are specified. The grammar for specifying their values is the same in either case.

For example, the following shows a class-definition and an equivalent object-definition. Note the two techniques for initializing the stubs attribute.

FILTER, a class-definition

Notes	OK
Authors	ghw (5 Jun 2000 4:11 p.m.)
Change log	0 entries
Item configuration	none
Class name	filter
Direct superior classes	object
Class specific attributes	none
Instance configuration	none
Change	none
Instantiate	yes
Include in menus	yes
Class inheritance path	filter, object, item
Inherited attributes	none
Initializable system attributes	attribute-displays, stubs
Attribute initializations	none
Icon description	inherited

FILTER, an object-definition

Notes	OK
Authors	ghw (5 Jun 2000 4:12 p.m.)
Change log	0 entries
Item configuration	none
Class name	filter
Direct superior classes	object
Class specific attributes	none
Instance configuration	none
Change	none
Instantiate	yes
Include in menus	yes
Class inheritance path	filter, object, item
Inherited attributes	none
Attribute initializations	none
Icon description	inherited
Attribute displays	inherited
Stubs	inherited

Class Inheritance and Class Definition Types

A class defined on one type of specialized class definition cannot be a direct-superior class on another type of specialized class definition. For example, a message class cannot be a direct superior for a class defined on an object-definition because object-definitions define only subclasses of object. A message class and an object class are considered disjoint classes because they do not have a inferior/superior class relationship.

A class defined on a class-definition can be a direct superior to a class defined on a specialized class definition if its inheritance is compatible with the classes supported by the specialized class definition. For example, an integer-variable class defined on a class-definition can be a direct superior for a class defined on an object-definition because integer-variable inherits from object. A class defined on any of the specialized class definitions can be a direct superior for a class defined on a class-definition because all class types can be defined on a class-definition.

However, when specifying more than one direct superior for a class, the direct-superior classes must be compatible; that is, they must have a inferior/superior class inheritance. The G2 compiler will always give you a reason when it rejects a

value you have entered in the direct-superior-classes attribute of a class definition.

Creating an Object Definition

An object-definition can define a subclass of any object class.

To create an object-definition:

1 Select KB Workspace > New Definition > class-definition > object-definition.

2 Click to place the new definition on a workspace:

See Storing Definitions on Workspaces on page 538 for related information.

The attributes of a connection-definition are included in the table in Class Definition Attributes on page 539, and the general considerations listed in that section apply.

3 Carry out the instructions in Configuring Class Definitions on page 542, omitting the two sections that pertain to initializable-system-attributes:

- Determining the Initializable System Attributes on page 546.
- Specifying Default Values of Initializable System-Defined Attributes on page 558.

The primary direct superior *must* be object or a subclass of object.

4 Carry out the instructions under Specifying Instantiability on page 560.

5 Optionally, provide or obtain values for the following attributes as described in the sections indicated. To provide a value, edit the value cell of the attribute, as with any table attribute.

Attribute	Section
attribute-displays	Specifying Attribute Displays on page 566.
stubs	Specifying Connection Stubs on page 567.
icon-description	Specifying an Icon on page 562.

Creating a Connection Definition

A connection-definition can define a subclass of any connection class.

To create a connection-definition:

1 Select KB Workspace > New Definition > class-definition > connection-definition.

2 Click to place the new definition on a workspace:

See Storing Definitions on Workspaces on page 538 for related information.

The attributes of a connection-definition are included in the table under Class Definition Attributes on page 539, and the general considerations listed in that section apply.

3 Carry out the instructions under Configuring Class Definitions on page 542, omitting the two sections that pertain to initializable-system-attributes:

- Determining the Initializable System Attributes on page 546.
- Specifying Default Values of Initializable System-Defined Attributes on page 558.

The primary direct superior *must* be connection or a subclass of connection.

4 Optionally, provide or obtain values for the following attributes as described in the sections indicated. To provide a value, edit the value cell of the attribute, as with any table attribute.

Attribute	Section
cross-section-pattern	Defining Connection Regions on page 578.
stub-length	Specifying a Stub Length on page 579.
junction-block	Defining the Junction Block to Use on page 579.

Creating a Message Definition

A message-definition can define a subclass of any message class.

To create a message-definition:

1 Select KB Workspace > New Definition > class-definition > message-definition.

2 Click to place the new definition on a workspace:

See Storing Definitions on Workspaces on page 538 for related information.

The attributes of a connection-definition are included in the table under Class Definition Attributes on page 539, and the general considerations listed in that section apply.

3 Carry out the instructions under Configuring Class Definitions on page 542, omitting the two sections that pertain to initializable-system-attributes:

- Determining the Initializable System Attributes on page 546.
- Specifying Default Values of Initializable System-Defined Attributes on page 558.

The primary direct superior *must* be message or a subclass of message.

4 Carry out the instructions under Specifying Instantiability on page 560.

5 Optionally, provide or obtain values for the following attribute as described in the section indicated. To provide a value, edit the value cell of the attribute, as with any table attribute.

Attribute	Section
default-message-properties	Specifying Default Message Properties on page 584

Customizing Definition Classes

Definitions are the means by which the G2 class hierarchy is extended. This extensibility extends to the definition classes themselves. You can:

- Enter a class-definition class in the direct-superior-classes attribute of an instance of a class-definition.
- Give the customized definition user-defined attributes.

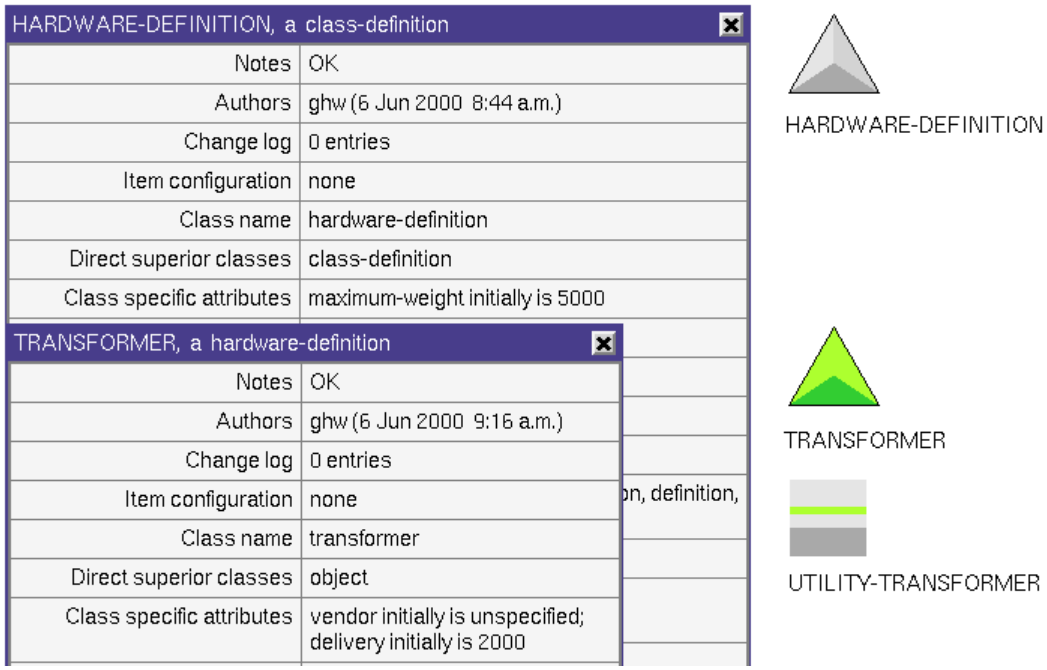
An instance of the customized definition is itself a definition, so it can be used to define a new class. Any instance of that class can then reference that definition to obtain the value of the definition's user-defined attribute(s) using the grammar:

the *attribute* of the definition named by the class of *item*

This capability is useful when an attribute relevant to a class has the same value for all instances of a class at any given time. Storing the attribute in the definition:

- Saves space by not storing a redundant copy of the attribute and its value in each instance.
- Saves time when the value changes by avoiding the need to update every instance to reflect the new value.
- Is more modular, and provides faster access, than using a freestanding variable or parameter to factor out the attribute.

For example, the next figure shows a user-defined class-definition class, hardware-definition. The class transformer is defined on an instance of hardware-definition, and utility-transformer is an instance of transformer:



This code accesses the value of the maximum-weight attribute on the class-definition that defines transformer:

the maximum-weight of the definition named by the class of utility-transformer

Creating New Classes Programmatically

You can create new definitions interactively, by instantiating and completing a definition as described previously in this chapter, or programmatically, by writing procedures that use some combination of these actions and capabilities:

- create (page 786)
- make permanent (page 801)
- transfer (page 821)
- conclude (page 783)

To create a definition programmatically:

1 Execute this action:

create a *definition C* [by cloning *existing-definition*]

where:

definition is one of class-definition, object-definition, connection-definition, or message-definition.

C is a local name.

existing-definition is a definition whose type matches *definition*.

When you clone an existing definition, the clone has the same attribute values as the original except for the class-name, which reverts to none.

2 Transfer the cloned definition to the workspace on which you want it to reside.

3 Make the definition permanent.

4 Use conclude as needed to give the new definition the name, class-specific attributes, and default values that you need.

By default, any items created with the create action are transient. Definitions must be permanent items before they define classes and can have any instances or subclasses.

Changing Definitions

You can change definitions two ways:

- Interactively, using the change attribute of a definition, as described in the next section.
- Programmatically, using the conclude action, as described under Changing Definitions with the Conclude Action on page 596.

Using the Change Attribute

The change attribute lets you change user- and system-defined attribute values, including connection specifications. This attribute exists only in definition classes, and is unique within G2.

The attribute functions as a command interpreter to let you manipulate instantiated classes, their attributes and behavior, from a single location while G2 is running. The value of the change attribute is none and remains so even after you perform a change. G2 executes the change command upon completing the statement within the Text Editor and the attribute value reverts to none.

The things you can do with the change attribute depend on the foundation class of the definition to be changed. Some changes are possible irrespective of the foundation class; others are possible only for subclasses of object, connection, or message. The possible changes are shown in the following table:

	Foundation Class of Definition			
Change Attribute Option	**Any**	**Object**	**Connection**	**Message**
Add a connection stub		✓		
Change the attribute	✓	✓	✓	✓
Change a connection stub		✓		
Copy inherited icon description		✓		
Delete a connection		✓		
Merge all instances and subclasses	✓	✓	✓	✓
Move attribute	✓	✓	✓	✓
Move connection		✓		
Rename attribute	✓	✓	✓	✓
Update each instance per attribute-displays		✓		
Update each instance per default-message-properties				✓

The following sections describe each of these options, noting whether the option is applicable to only one definition. For further information on the effects of changing a definition, see Effect on Subclasses and Instances on page 597 and Effect on Procedure Statements and Other Items on page 601.

Adding a Connection Stub to an Object Class Definition

You can add a connection stub of a particular *connection-class*, optionally using a *portname* and a stub location, by using this syntax:

add {a | an} *connection-class* [*portname*] located at
{top | bottom | left | right} *integer*

Here is an example of adding a connection:

add a movement-connection out-port located at right 10

This example adds a connection of the movement-connection class, using an out-port portname, located at right 10 on the icon.

Changing an Attribute to its Default Value in Instances

You can change the value of an attribute to its default value for all instances using this syntax:

change the attribute *attribute* of each instance to the default value
[, preserving non-default values when switching to or from values
given by a variable-or-parameter]

Here is an example:

change the attribute volume of each instance to the default value

When the attribute whose value you are changing is given by a variable or a parameter, you can use this optional statement:

, preserving non-default values when switching to or from values
given by a variable-or-parameter

This option lets you preserve any non-default initial values you may have provided the variable or parameter as you switch to the default value.

Changing Stubs in Object Class Definitions

Changes the stub specification. You can change a connection stub in three different ways:

- Connection class
- Stub direction
- Portname

To change the connection class, use this syntax:

change the connection class of the connection {at *portname* |
located at {top | bottom | left | right} *integer* } to *connection-class*

Here is an example:

```
change the connection class of the connection located at right 5
to movement-connection
```

Hint When specifying the location of a connection, you can refer to it either by its *portname* or by its location and position (as an integer) on one side of the icon (located at right 5). The changes affect all instances of the class.

To change the stub direction, use this syntax:

```
change the direction of the connection {at portname |
  located at {top | bottom | left | right} integer } to {none | input | output}
```

Here is an example:

```
change the direction of the connection at out-port to output
```

To change the portname of the connection, use this syntax:

```
change the portname of the connection {at portname |
  located at {top | bottom | left | right} integer } to portname
```

Here is an example:

```
change the portname of the connection located at right 5 to out-port
```

Copying an Inherited Icon-Description in Object Class Definitions

Use this option to copy the default icon description for an object. The default icon description is the first explicitly defined icon description that G2 finds on the class inheritance path. When copying an inherited icon description, if the class that defines the icon specifies stubs, then the stubs are inherited also. Enter the command as follows:

```
copy inherited icon description
```

This statement changes these attributes back to their default values:

- stubs (inherited)
- icon-description (the description of whatever is the inherited icon for this class appears as the value for this attribute)

Any changes you may have made to the icon-description or stubs attributes, revert back to the inherited descriptions. For example, while you cannot change the icon description of an object instance, you can change the icon's position, size, and rotation. After you complete this change option, instances reflect the appearance of the inherited icon.

Deleting Connections in Object Class Definitions

To delete a connection from a stubs attribute, and from all instances, use this syntax:

```
delete the connection {at portname |
  located at {top | bottom | left | right} integer}
```

Here is an example:

```
delete the connection located at right 5
```

If the stub had a portname, you can specify the connection's location by its portname rather than its location as shown above (located at right 5).

Merging All Instances and Subclasses into a Definition

This topic is described under Merging Classes on page 603.

Moving Attributes from One Class to Another

Use the move option to move an attribute from one definition to another.

You can move an attribute to either a superior or an inferior definition (the target class) within the class hierarchy. The target class cannot include a class-specific attribute of the same name, but it can have one or more same-name attributes through inheritance. The move option preserves non-default attribute values in class instances of both superior and inferior classes.

To move an attribute, use this syntax:

```
move attribute attribute-name to class
```

where *attribute-name* is the name of the attribute to move and *class* is the target class to which you are moving the attribute. Here is an example:

```
move attribute year to car-object
```

Moving Connections

To move a stub from one location to another, use this syntax:

```
move the connection {at portname |
  located at {top | bottom | left | right} by integer } to integer
```

Here is an example:

```
move the connection located at right 5 to 10
```

You can specify the current position of the stub either by its location on the icon (as shown here), or by its portname. You can specify that the connection be moved *by* an integer amount or *to* a positive-integer location elsewhere on the same side of the icon.

Renaming an Attribute

Use the rename option to rename an attribute like this:

```
rename attribute-number-of-doors to door-total
```

You cannot use the change attribute to change the name of any system-defined attributes.

Updating the Attribute Displays of All instances of a Class

You use the update option to update the attribute displays for all existing instances of a class.

To update the attribute displays of all instances of a class:

1. Enter the attribute displays you want to change instances to in the attribute-displays attribute. For example, if you want all instances to display the class-inheritance-attribute at the standard position, enter this:

   ```
   class-inheritance-attribute at standard position
   ```

2. Enter the update option of the change attribute like this:

   ```
   update each instance per attribute-displays
   ```

This command immediately updates all instances with the new attribute display, discarding any interactively created attribute displays, or manual changes to existing displays.

Updating the Default Message Properties of All Class Instances

Use the update option to update the default-message-properties for all instances of a message class like this:

```
update each instance per default-message-properties
```

This change substitutes the current value of the default-message-properties attribute for whatever message properties are in existence in each instance of the message class.

Changing Definitions with the Conclude Action

You can use the conclude action to change all of the editable attributes of a definition, *without* pausing or resetting the knowledge base (KB) for:

- An instantiated (or non-instantiated) class.
- A permanent or transient definition.

For details on using conclude for this purpose, see Concluding Attribute Values on page 783. For further information on the effects of changing a definition, see the next section and Effect on Procedure Statements and Other Items on page 601.

Caution G2 permits you to change an instantiated definition while the KB is running, but the consequences can be extensive and severe. Carefully consider the impact of changing any instantiated definition *before* continuing with the change.

Effect on Subclasses and Instances

Changing definition attributes, whether the KB is reset or not, has the following effects. The effects are the same for every type of definition.

Attribute	Type of Change	Effect on Subclasses and Instances
class-name	New class name of any name. You cannot rename a superior class to one of its subclasses, or a subclass to its superior class.	Existing instances become instances of the new class name. After changing a class name with an action that refers to the definition by name, the action includes a note that the (original) item no longer exists.
direct-superior-classes	Cannot change the foundation class in this attribute if instances exist. Cannot change to any other prohibited superior class specification.	The inheritance of subclasses and instances changes.
class-specific-attributes	Adding attributes Deleting attributes	Adds attribute to existing classes. Removes attribute from existing classes.

Attribute	Type of Change	Effect on Subclasses and Instances
change	Renaming attribute	Changes the name in all instances.
	Changing attribute to default value	Gives each instance the default specified in the definition. Can cause G2 to remove the value of an inherited attribute from existing instances if, for example, the default value was none and an instance contains a value.
	Changing the portname, direction, or connection class of a connection	Updates all instances as directed.
	Merging all subclasses and instances	All subclasses and instances become subclasses and instances of the class into which you are merging another class.
	Moving an attribute from one class to another	G2 updates the subclass (or superior class) with the new attribute. Maintains non-default attribute values of all instances.
	Adding, deleting, or moving an input or output connection.	Updates all instances as directed.
	Copying inherited icon description	Updates all instances with the first explicitly defined icon description that G2 locates on the class inheritance path.
	Updating each instance per attribute-displays.	Updates all instances to any specified attribute-display.

Attribute	Type of Change	Effect on Subclasses and Instances
instantiate	Changing default option	Updates menu choice to either add or delete the class from applicable G2 menus.
include-in-menus	Changing default option	Updates menu choice to either add or delete the class from applicable G2 menus.
attribute-initializations	Changing the default value of an inherited attribute	Updates all instances with the new default value, unless the previous default value of the instances has been changed.
	Changing the *data type, initial value, and history-keeping spec* for a variable or a parameter, or changing the *validity interval*, *supply simulation subtable,* and *default update interval* for a variable The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices on page 2169.	Updates all instances with the new value, unless the instance has a non-default value.
	Changing the *options for* attribute for a parameter or variable (forward chaining, etc.)	Updates all instances. New instances have the new value.
	Changing the element-type of an item-list or an item-array	Updates all instances with the new value. You cannot change the element-type of a g2-array without resetting the KB and when the array has instances.

Attribute	Type of Change	Effect on Subclasses and Instances
	Changing whether a list allows duplicate elements	Updates all instances with the new setting. You cannot change a g2-list without resetting the KB and when the list has instances. When changing from allow-duplicate-elements to do-not-allow-duplicate elements, G2 does not delete duplicate elements, but prevents you from adding duplicate items from after the change occurs.
	Changing/specifying the array length for an array	Updates existing instances, whose length value has not been changed. Instances with changed length values maintain their values. New instances will contain the new length as specified. When changing the length of a populated array, G2 increases or decreases the length, and all elements receive the default value for that array.
attribute-displays	Displaying an attribute	Does not update existing instances, only new instances. To update all instances, use the Change attribute, with *update each instance per attribute-displays*, as the previous table describes.
stubs	Adding new stub or changing a current stub	Does not update instances, only new instances created after you add or change a stub.

Attribute	Type of Change	Effect on Subclasses and Instances
icon-description	Change description or specify Inherited	Changes all instances. Adding an image file is reflected in all instances.
junction-block (connection classes only)	Changing the junction-block name	Does not update current instances, only new instances that use the junction block.

Note You cannot programmatically change either the class-name or direct-superior-classes attribute of a definition that has been declared as stable for dependent compilation and which already has a class name. For more information on declaring stable definitions, see Chapter 53, Profiling and KB Performance on page 1811.

Effect on Procedure Statements and Other Items

Changing certain definition attributes can affect executing procedures and other items that include expressions, along with arrays, lists, and relations. The effects happen because you can change the origination of attributes and instances to another class, or delete them entirely. The definition attributes that can affect procedure statements, rules, formulas, arrays, lists, and relations are:

- class-name
- direct-superior-classes
- change attribute, specifically:
  - Merge all instances and subclasses
  - Change the connection class of the connection
  - Change or delete a connection
  - Change attribute to default value
- class-specific-attributes (if the change removes an existing attribute)
- attribute-initializations

Changing Instantiated Classes

The effect of changing an instantiated class upon executing procedures (and other items with expressions) depends on the type of change being made and on when the change occurs. For instance, a procedure may refer to an instance of some definition, with a statement such as:

```
T: class tank = the tank upon this workspace
```

Changing an attribute such as the class-name of the definition for tank removes the value from the local name of any executing procedures. Any statement that thereafter refers to the local name's value gets a procedure error.

If a statement, such as a procedure for statement containing a reference to the original class name, has begun, but is incomplete while a change is made to the definition, the statement continues executing with the values of the definition *before* the change, unless the change to the definition deletes items that the statement references. When changing a definition deletes items, the for statement does not evaluate those items.

Changing an instantiated class can affect lists, arrays, and relations. For instance, an item list or array can have elements that contain instances of the tank class. A relation can exist between two tank instances.

Whenever you change one of the definition attributes listed earlier in this section, G2 can verify the contents of all list, arrays and relation items to see if they still contain items that do not conflict with their defined relation classes or element-type. If a list or array element or a relation instance is no longer valid, G2 removes it from the array, list, or relation.

G2 validates executing procedures, rules, generic formulas, and generic simulation formulas, using the items that its local names reference. If a change either deletes that item or changes its class inheritance from what the statement expects, G2 clears the local name.

The G2 Simulator, which can use generic simulation formulas, is a superseded capability. For more information, see Appendix F, Superseded Practices on page 2169.

If you change an item that is part of a rule's antecedent, the scheduler dismisses the rule invocation if the item is invalid.

Note If a KB save operation is taking place while a definition change is occurring, G2 saves the KB with the affected definitions and their instances as they were *before* the change to the definition is complete.

Merging Classes

Merging classes gives you a way to simplify the class hierarchy while retaining all the subclasses and instances of the class you want to eliminate. When you merge two classes, the instances and subclasses of one of the classes become instances and subclasses of the other class. You can merge only user-defined classes.

The merge option of the change action merges two definitions into a single definition. One of these is called the **primary definition**. The class that is merged into it is the **secondary definition**.

Merging Definitions of the Same Type

When you merge two class-definitions, object-definitions, connection-definitions, or message-definitions, the merged definitions must be identical except for the class name. For classes to be identical, the class-specific attributes and their values must be the same, and the order in which the attributes appear in the definition must correspond.

For example, if you try to merge object-definitions that have two class-specific attributes with the same name and value, but which appear in a different order, G2 views the definitions as different, even though the attribute names and values are equivalent.

Before merging definitions of the same type, you must change the definitions as needed to make them identical. You can then enter the merge option of the change attribute in the definition table of the secondary class to merge it into the primary class. For complete information about the Change attribute, see Using the Change Attribute on page 592.

To merge definitions of the same type:

1. Display the table of the primary definition, that is, the one you want to remain after the merge action.
2. Display the table of the secondary definition, that is, the one you want to eliminate.
3. If the two tables are not identical, apart from the class-name, edit the definitions as needed to eliminate all differences.
4. In the secondary definition, enter this merge command in the change attribute:

   merge all instances and subclasses into definition for *primary-definition*

   where *primary-definition* is the primary class definition, into which you want to merge the secondary definition.

Merging Classes Defined on Definitions of Different Types

You can merge a class defined on an object-, connection-, or message-definition into a class defined on a class-definition, but you *cannot* merge a class defined on a class-definition into a class defined on an object-, connection-, or message-definition. That is, the class defined on the class-definition must always be the primary definition when definitions of different types are merged.

No object-, connection-, or message-definition can be identical with a class-definition. Consequently, requiring strict identity when merging into a class-definition would preclude upgrading existing object-, connection-, and connection-definitions to be class-definitions.

Therefore, G2 does not require strict identity when merging into a class defined on a class-definition. Such a merge is possible whenever the foundation classes of the merged classes are the same. (A foundation class is any extensible system-defined class except a mixin). When such a merge occurs, every difference between the primary and secondary definitions is resolved in favor of the primary definition:

- Attributes defined in the secondary definition but not the primary definition disappear from subclasses and instances of the secondary definition.
- Attributes defined in the primary definition but not the secondary definition are added to subclasses and instances of the secondary definition.
- Attributes that exist in both definitions but have different properties use the properties in the primary definition. Subclasses and instances of the secondary definition change accordingly.

To merge classes defined on definitions of different types:

1. Be sure that the two classes inherit the same foundation class(es).
2. In the secondary definition, enter this merge command in the change attribute:

   merge all instances and subclasses into definition for *class*

   where *class* is the primary class, into which you want to merge the secondary definition.

Completing a Merge

After G2 merges the two classes, it displays a message on the Operator Logbook, stating that the two classes are merged. All subclasses and instances of the secondary class whose definition you are editing are now subclasses and instances of the primary class.

The definition for the secondary class still exists and can safely be deleted. If the definition has a subworkspace, be sure to transfer any needed information from the subworkspace to another workspace before you delete the definition.

Deleting a Definition

You delete a definition by selecting delete from its menu. When you delete a definition, you automatically delete:

- The subworkspace (if any) of the definition.
- All class and subclass instances.
- Any subworkspaces of those instances.

G2 does not delete definitions that depend on a deleted superior class, but such definitions become incomplete because a direct superior class does not exist. G2 changes their notes status to incomplete, and notes that one (or more) direct superior classes is not defined.

Instance deletion occurs because, to exist, an instance requires a complete set of information from every superior class within its class hierarchy. When you delete any superior class within an item's class hierarchy, essential information no longer exists, and thus, neither can the item.

Once you confirm that you want to delete a definition, you cannot reverse the delete command. Therefore, use caution *before* deleting any definition that may have instances crucial to other workspaces. Since an error can be quite costly, it is good practice to save your KB before deleting any definitions.

15

Variables and Parameters

Describes variables and parameters and how to use them within a KB.

Introduction

Variables and parameters are items of the g2-variable or parameter class of objects, which you use as placeholders for values of specific data types.

Variables and parameters are designed to represent a data value. You can use displays such as charts and trend charts to display the values of each, and use button items, such as radio buttons and sliders, to change their values interactively. You can create a variable or a parameter to represent any simple G2 type.

Comparing Variables and Parameters

You use variables and parameters in your knowledge base (KB) to represent changing data and as a means of maintaining a history of important values. Within a KB, variables and parameters can exist as autonomous objects, to hold the attribute values of user-defined objects, or as superior classes of your own user-defined variable and parameter classes.

Because all variables and parameters represent a value, you can refer to either in an expression as a value, rather than as an item. For example, this expression, referring to the quantitative variable Q1:

```
conclude that Q1 = 200
```

replaces the value of Q1 with 200, since G2 evaluates the variable as a value.

Within a KB, you can use variables and parameters to represent external values. For example, in a KB monitoring a nuclear power plant, variables could represent temperature readings. You use variables or parameters to represent such knowledge because they:

- Are the only items in G2 that include history keeping capabilities, which lets you maintain a record of changing values.
- Can trigger other events when you use them in conjunction with rules.

For more about the interaction with rules, see Variables, Parameters, and Rules.

Parameter Features

In effect, a parameter is a simple variable. Parameters provide a storage space in memory for an inconstant data value that can be updated as often as necessary. In addition to storing a data value, a parameter has these capabilities:

- History keeping
- Forward chaining

History keeping permits the parameter to save its values over a specified period of time. Once historical data exists, you can display, evaluate, store, and retrieve its contents.

Forward chaining allows the parameter to trigger other events when it receives a value, by invoking rules that refer to it.

A parameter is guaranteed to have a value, and does not require data seeking to obtain one. Parameters supply reliable data sources that all parts of G2 can access.

Variable Features

In contrast, a variable can be thought of as a complex parameter. A variable also stores a data value, keeps history, and can cause forward chaining to rules. Additionally, a variable can:

- Specify a particular source, called a **data server**, through which it will obtain a value.
- Include a specific formula for evaluating a value.
- Require that G2 generates a value at certain intervals.
- Define an expiration time for its value.
- Cause data seeking by backward chaining to a rule.

A variable does not always have a value, and its value can expire.

All variables are capable of having two values concurrently: a value obtained through a data server, and a simulation value. You can save a history of one or both of those values.

The G2 Simulator, which can provide simulation values, is a superseded capability. For more information, see Appendix F, Superseded Practices.

Memory Considerations

As with all items, variables and parameters maintain their capabilities within their attributes, such as last-recorded-value and validity-interval.

The chaining capabilities of both items are an exception. Although you specify which chaining options to use in an attribute of a parameter or a variable, the capability is actually a behavioral aspect of the inference engine. As such, it does not severely impact the memory requirements of a parameter or variable.

Since variables have more capabilities than parameters, they require more memory. In addition, the history-keeping-spec attribute for both variables and parameters can require potentially large amounts of storage, depending on its value.

Summary of Variable and Parameter Differences

The following table summarizes the differences between parameters and variables:

Characteristic	Variable	Parameter
Always has a value		✓
Can have a validity interval	✓	
Can receive values from a conclude action	✓	✓
Can receive values from a generic formula	✓	
Can receive values from an external data source	✓	
Can receive values from a specific formula	✓	
Can be referenced directly by a procedure		✓
Can have a history	✓	✓
Has a data server	✓	
Must have an initial value		✓
Can cause forward chaining	✓	✓
Can cause backward chaining or other data seeking	✓	
Can be the value of an object attribute	✓	✓

Consider these issues when deciding whether to use a variable or a parameter:

If the data value...	Then choose a...
Requires a specific data server, such as G2 Gateway, for obtaining values outside of G2	Variable, since you cannot specify a data server in a parameter.
Is time dependent so that the item updates on a regular basis and can expire	Variable, since parameters do not support expiration times.
Requires a value at all times, and does *not* require a specific data server	Parameter, since a parameter is guaranteed to have a current value and does not have a data server.

Variables, Parameters, and Rules

Parameters and variables are tightly coupled with rules. A common use of rules is to test whether a variable or parameter has a certain value, and, if so, to take some subsequent action, possibly involving another variable or parameter.

For both parameters and variables, the relationship with rules involves **forward chaining**. Each can either:

- *Cause* forward chaining to a rule when either receives a value.
- *Receive* a value from the consequent of a rule invoked from another variable or parameter that forward chained to the rule.

Forward chaining is the process of invoking a rule when at least one of the conditions in its antecedent is true. Consider the following rule as an example.

```
if the tank-volume of auto-1 is low then conclude that auto-needs-gas is true
```

Whenever the tank-volume variable for auto-1 obtains the value low from some source, that occurrence *causes* forward chaining to the antecedent of this rule. The rule's consequent concludes a new value for the auto-needs-gas parameter. When the auto-needs-gas parameter *receive*s a value as the effect of the variable update forward chaining to the rule, it, in turn could cause forward chaining to another rule containing that parameter in its antecedent. For a complete description of forward chaining, see Forward Chaining.

In summary, a new value for a variable or parameter can *cause* forward chaining to the antecedent of one or more rules.

For variables, the relationship with rules involves both forward chaining and backward chaining, described next.

Obtaining Values for Variables

The purpose of variables is to hold a value and, optionally, to maintain a record of values through history keeping.

Updating a variable value can take several forms. A variable can receive an unrequested value or one that is requested. Both types of update have effects on the KB.

Obtaining Unrequested Values

When variables receive unrequested values, the new value can cause forward chaining which, in turn, can conclude values for other variables. A variable can receive an unrequested value from:

- A conclude action from a procedure, method, or rule.
- An external data server such as G2, G2 Gateway, or GFI.
- A user event, such as a check-box or a type-in box.
- Activation, if the variable has an initial value.

For information on GFI, a superseded capability, see Appendix F, Superseded Practices. For a description of using initial values, see Specifying an Initial Value.

Obtaining Requested Values

The events that create a request for a new variable value are:

- The variable's default-update-interval attribute.
- An update interval in a display item that refers to a variable.
- A rule that references the variable.
- A local name declaration, a collect data statement, or a wait until statement in a procedure that references the variable.
- An update action that references the variable.

When G2 receives a request for a variable value, it first determines whether the last-recorded-value is still valid by checking the validity-interval attribute. The validity interval of a variable is described in Specifying a Validity Interval. If the value has not expired, G2 uses it and does not seek a new value. G2 seeks a value for a variable only when an item needs a value for the variable *and* the variable's value has expired. The attempt to obtain a new value for a variable is called **data seeking**.

Value expiration does not itself cause G2 to data seek for a value. If a variable value expires and no outstanding request for that value exists, G2 does not try to obtain a value automatically.

G2 begins data seeking by determining which data server the variable specifies in its data-server attribute. A data server is the process or task that tries to obtain a value for the variable. The variable data servers are:

- The inference engine
- The G2 Simulator
- Some other data server, such as G2 Gateway or GFI

GFI and the G2 Simulator are superseded capabilities. For more information, see Appendix F, Superseded Practices.

When the data server is other than the inference engine, G2 can receive both requested and unrequested values from that data server. The rest of this section shows how each of the data servers gets a value for the variable. Variable failures are described in Handling a Variable Failure.

Inference Engine Data Server

The inference engine can obtain values from rules or formulas.

To obtain a value for a variable, the inference engine:

1. Checks to see whether the variable has a specific formula in its formula attribute.

   If a specific formula exists, G2 uses it to compute a value for the variable.

2. Checks to see whether there is a generic formula. If one exists, G2 uses that.

3. When neither a specific nor a generic formula exists for the variable, and **backward chaining** is permitted, G2 backward chains to a rule to find a value.

   Backward chaining is the process of invoking one or more rules capable of inferring a value for the variable. For a description of backward chaining, see Backward Chaining.

4. If neither a specific nor a generic formula exists, and backward chaining does not produce a variable, G2 fails to get a value.

Forward and backward chaining are controlled by the options attribute of both rules and variables. For variables to be able to backward chain to a rule, and for variables and parameters to forward chain to a rule, both items must have their options attribute set appropriately.

G2-Simulator Data Server

The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices.

G2 or GFI Data Server

If the variable is using one of the other data servers, G2 sends a request to that data server for a value. G2 then performs other tasks until it receives a value.

Note The G2 inference engine timeout-for-variables attribute setting does not apply when G2 is obtaining updates from an external data server, such as G2 Gateway.

If G2 does not receive a value within the time specified by the Inference Engine Parameters system table attribute, timeout-for-variables, G2 fails to get a value for the variable.

GFI is a superseded capability. For more information, see Appendix F, Superseded Practices.

Handling a Variable Failure

A variable fails to receive a value if G2 requests a value and *one* of the following is true:

- The variable does not receive a value within the time period specified in the timeout-for-variables attribute in the Inference Engine Parameters system table.
- G2 attempts to obtain a value as the sections describing each of the data servers explain, and all possible methods fail.

A variable is not considered failed if G2 withdraws all requests for the variable's value *before* the timeout expires, or before it tries all avenues for finding the value.

When a variable fails, G2 does two things:

1. It invokes any whenever rules that check for a failure in the variable.
2. It schedules a task to try again to get a value for the variable.

Invoking Whenever Rules for Failed Variables

G2 invokes whenever rules automatically when at least one statement in the antecedent is true.

Two forms of whenever rules that you can write to test variable values are:

- whenever *variable* receives a value
- whenever *variable* fails to receive a value

You can use whenever rules to initiate specific action in the event of a failed variable. For example, a rule such as this:

```
whenever any temperature-variable V fails to receive a value
    then start value-test (V)
```

could start a procedure whenever G2 detected a variable failure. Using whenever rules this way is analogous to writing on error statements in procedures. Such whenever rules are invoked once after each time the variable fails to receive a value, not after each retry task.

Retrying the Variable

As long as there are pending requests for the variable's value, G2 retries the variable once every retry period, which is specified in the retry-interval-after-

timeout attribute in the Inference Engine Parameters system table. When no more requests for the value of the variable exist, G2 stops retrying the variable, but the variable is still considered failed.

A variable remains failed until it receives a value.

Obtaining Values for Parameters

Parameters receive values from expressions within procedures, rules, and user-interface items, such as action buttons and sliders.

Parameters are guaranteed to have a current value. If you do not assign an initial value to a parameter, G2 supplies an initial value corresponding to its data type:

Parameter Type	Initial Value
quantitative	0.0
float	0.0
integer	0
long	0L
logical	false
text	""
symbolic	g2

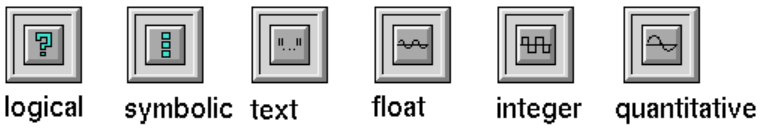
Creating Variables and Parameters

To create a variable or a parameter:

➔ Select KB Workspace > New Object > *variable-or-parameter*.

where *variable-or-parameter* is either variable or parameter from the New Object menu. After you select either choice, G2 displays a secondary menu from which to choose a particular type.

Variables and parameters share similar icons, a separate icon for each type:



Specifying Forward and Backward Chaining

For parameters and variables, the options attribute controls whether a new value causes forward chaining. For variables only, the attribute also controls whether G2 can data seek and use backward chaining. If backward chaining is in use, the attribute specifies the type to use (breadth first- or depth first-backward chaining).

The default settings for the different kinds of variables and parameters are:

This type...	Has this default setting...
Quantitative and text parameters	do not forward chain
Symbolic and logical parameters	do forward chain
Quantitative and text variables	do not forward chain, breadth first backward chain
Symbolic and logical	do forward chain, breadth first backward chain

The allowable values for each type of variable or parameter are:

This item...	Can have these settings...
Parameters	do [not] forward chain
Variables	do [not] forward chain {do not {seek data | backward chain} | , depth first backward chain | breadth first backward chain}

Use caution when setting or changing the value of the options attribute. Whatever you specify in this attribute, or whatever default value it provides, should correspond directly with any rule that references the parameter or variable.

For example, if you want a parameter to cause forward chaining to a rule when it receives a value, the default option for the parameter should be:

```
do forward chain
```

This setting indicates that the parameter can forward chain to any rules that reference it. If the rule that references the parameter is *not* itself invocable via forward chaining, the chaining process comes to a halt.

Forward Chaining on Unchanged Variables and Parameters

By default, G2 performs forward chaining on variables and parameters only when the value changes. G2 can also perform forward chaining on variables and parameters even if the received value is the same as the old value. To enable this behavior, add even for same value after do forward chain in the options attribute of a variable or parameter. For example, on a variable, you would specify:

```
do forward chain even for same value, breadth first backward chain.
```

Note this is only applicable to forward chaining and does not affect backward chaining.

The value of the options attribute contains an optional attribute in the structure, named forward-chain-even-for-same-value, which is a truth-value, whose default value is false. Thus, the value for the above set of options looks like this:

```
structure
(forward-chain: true,
forward-chain-even-for-same-value: true,
backward-chain: the symbol breadth-first)
```

Defining Debugging and Tracing

The tracing-and-breakpoints attribute defines the level of tracing and breakpoints for the variable or parameter. This attribute and its values are described in Debugging and Tracing.

The tracing-and-breakpoints-enabled? attribute of the system table must have a value of yes for this attribute to take effect.

Specifying the Type

The data-type attribute specifies the type for the variable or parameter. You cannot edit this attribute for symbolic, logical, and text variables and parameters. These are the allowable values for each variable and parameter.

This variable or parameter type...	Can have this value...
quantitative	{unit-of-measure | interval | quantity | float | integer | time-stamp | pure number
float	{unit-of-measure | float | time-stamp | pure number}

This variable or parameter type...	Can have this value...
integer	{unit-of-measure | interval | integer | pure number}
long	long
symbolic	symbol
logical	truth-value
text	text

Specifying an Initial Value

The initial-value attribute specifies an initial value for the variable or parameter, which is dependent on the type. If you do not specify an initial value for a variable, G2 supplies the value none. Here is the default initial value for each type of parameter:

This item...	Has this initial value...
Quantitative parameter	0.0
Float parameter	0.0
Integer parameter	0
Long parameter	0L
Symbolic parameter	g2
Logical parameter	false
Text parameter	""

Hint If you are maintaining historical data on a parameter or variable (using the history-keeping-spec attribute), the value you provide in the initial-value attribute is used as an initial historical data point.

You can supply initial values in individual instances of variables or parameters, or in user-defined object definitions.

Variables with an initial value behave differently from parameters upon activation. In this context, activation refers to starting or restarting the KB, or activating the workspace of the variable or parameter after KB startup. For a

complete description of KB run states and their effect upon activating items, see Operating the Current KB.

For variables, G2 concludes the initial value as the value of the variable, with its collection time set to the activation time. Activating a variable with an initial value, therefore, causes forward chaining. If the variable is keeping history, the initial value is inserted into the history as if it had been collected from a data server.

Note When using a variable as an external output, an activate event does not have the same effect as a set action. While a set action can transmit a value to the external facility, G2 does not transmit the initial value when an activate event occurs.

Parameters have an initial value by default, even if you do not provide one, because they are guaranteed to have a value at all times. When a KB is reset, parameters revert to their initial value, and do nothing upon activation. Thus, parameters do not forward chain from activation. If the parameter is keeping history, activating the parameter inserts the initial value into history as the first collected value.

Obtaining the Last Recorded Value

The last-recorded-value attribute displays the value that G2 last concluded, collected, calculated, inferred, or received for the variable or parameter. You cannot change this attribute.

If the variable specifies a validity-interval, the last-recorded-value may not be current. If the value has expired, it is has an expired suffix, and if it is current, it is suffixed by the word expires, followed by the expiration time.

If no current value exists, an asterisk appears to the right of the value. If the last recorded value reads no value, G2 has not yet found a value for the variable.

The variable-or-parameter class defines a read-only, hidden attribute named last-recorded-value-text, which shows the last-recorded-value of the variable or parameter in text format as it appears in a readout table. If the value of a variable has expired, the hidden attribute appends an asterisk to the value (*). If the variable has no value, the hidden attribute value is "****". These features only apply to variables, not parameters. This attribute does not cause data seeking.

Use the format-numeric-text function to format the value of the last-recorded-value-text hidden attribute of a variable. For details, see Format-Numeric-Text Function.

Tip Expiration of a variable's value does not cause data seeking. G2 seeks for a value only upon request when the value has expired. For the causes of data seeking, see Obtaining Values for Variables.

Specifying Whether to Keep a History of Values

The history-keeping-spec attribute indicates whether to keep a history of values for the variable or parameter. If the item is keeping a history of values, the attribute can also specify the maximum number of values to keep, and how long to keep each value.

For more information about this attribute, see History Keeping in G2.

All variables have these additional attributes:

- validity-interval
- formula
- simulation-details
- initial-value-for-simulation
- data-server
- default-update-interval

The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices.

Specifying a Validity Interval

For variables, the validity-interval attribute specifies the length of time that the value of the last-recorded-value remains current. Enter a value for this attribute as follows:

Value	Description
time-expression	Defines a specific amount of time, such as 10 seconds. You cannot use a subsecond interval for the validity interval.
indefinite	Indicates that the last recorded value is valid indefinitely.
supplied	Specifies that the inference engine computes the validity interval. When supplied is the value for this attribute, inference engine is the only valid data server.

Caution If you are using a default-update-interval for the variable and you set the validity-interval to less than the update interval, the variable will constantly expire.

Effect of Validity Interval on Expiration Time Stamp

When a value is concluded into a variable, G2 stores two timestamps:

- The collection time, which is either the time at which the conclude occurred or an explicitly supplied time in a conclude action.
- The expiration time, which is computed as the collection time plus the value of the variable's validity-interval attribute.

A variable value expires when the current time is greater than the expiration time.

You can detect interactively that a value has expired by examining an attribute table showing the variable value. G2 updates attribute tables when a variable expires, so you can leave the table on display and watch for changes.

You can detect programmatically that a value variable has expired with the statement:

if *variable* has a current value

which returns the current expiration status of the variable. The expression:

if *variable* has a value

attempts to invoke data seeking to find a value, and only returns false if the variable has failed to receive a value or if the expression has timed out and is making its final evaluation attempt.

Note You can override the validity-interval of a variable by explicitly providing an expiration time as part of a conclude action.

This is how the value of the validity-interval attribute affects the variable's expiration timestamp:

If validity-interval is...	Then...
indefinite	The expiration time stamp becomes the symbol never.
supplied	Values concluded from rules and formulas receive an expiration time stamp that is the minimum of the expirations of the values used to execute the rule or formula. If a value is concluded from a procedure, the expiration time stamp is never. This occurs because procedures can only refer to variable values within a collect data or wait until statement, not within a conclude action.

Setting the validity-interval attribute also has a direct effect on the variable's forward chaining capabilities, if the variable permits forward chaining.

Using a Specific Interval

A specific validity interval, indicating that a current value expires at some point, *always* causes forward chaining to occur when the variable receives a value.

Using an Indefinite Interval

An indefinite validity interval causes forward chaining *only* if the new value received is different than the previous value. Forward chaining then becomes a response *only* to new information.

If your KB requires forward chaining to occur whether the variable's value is new or not, but needs to use an indefinite validity interval, create a whenever rule with a statement such as:

```
whenever variable receives a value...
```

G2 invokes this rule even if the new value is the same as the current value. G2 does *not* invoke the rule if the time stamp of the new value is either the same as or predates the current value.

The reason for this is that G2 only invokes a whenever *variable* receives a value rule when the time stamp of any new value is later than that of the current value. A value whose time stamp is identical to or earlier than that of the current value is not considered new. If the variable is maintaining history, G2 stores older time stamp values in history, but does not invoke the whenever rule.

Using a Supplied Interval

A supplied validity interval is applicable when G2 computes the value of the current variable using other variables. The data server for a supplied validity interval must be inference engine.

When the validity interval is supplied, the inference engine obtains a validity interval from the variables used to calculate or infer a value. Specifically, the inference engine uses the shortest available value. For example, consider that the volume of a tank-1 object is given by a float variable with a supplied validity interval, and G2 uses this expression to compute a value for volume:

```
the area of tank-1 * the level of tank-1
```

To obtain a validity interval for the volume variable, G2 checks the intervals of the area and level variables, and then uses the shorter interval of the two.

If a variable has a supplied validity interval, but is *not* getting its value from other variables, G2 infers that the validity interval is indefinite.

An exception to this is when an update source for a variable with a supplied validity interval includes a conclude statement specifying a validity interval, using a with expiration statement such as:

```
conclude that level = 100 with expiration 10 seconds
```

Other possible sources for obtaining a variable value include rules, formulas, and actions buttons.

Creating a Specific Formula

For variables only, the formula attribute includes a specific formula that G2 uses to compute a value for the variable. When a variable includes a formula, G2 calculates it only when the variable needs a new value.

Note The variable must use the inference engine as its data server for G2 to evaluate a specific formula.

Specifying Simulation Details

The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices.

Determining the Initial Simulation Value

The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices.

Specifying a Data Server

For variables only, the data-server attribute specifies which data server G2 uses to obtain a value for the variable. If you are using a formula, you must use inference-engine as the data-server.

Data Server	Description
data-server-alias	Is an alias for a particular data server. You specify the data server that each alias name implies in the Data Server Parameters system table by modifying the data-service-alias attribute.
inference engine	Tells G2 to get a value from a formula, a generic formula, or to data seek for a value using backward chaining in that order.

Data Server	Description
GSI data server	Indicates that G2 Gateway (GSI) is the data server. This is the data server used for bridges and other external resources. Refer to the *G2 Gateway Bridge Developer's Guide* for more information.
GFI data server	GFI is a superseded capability. For more information, see Appendix F, Superseded Practices.
G2 meter	This data server is only applicable for a variable that you have created as a g2-meter. For a description of using g2-meters, see G2-Meters.
G2 simulator	The G2 Simulator is a superseded capability. For further information, see Appendix F, Superseded Practices.
G2 data server	Tells G2 to get a value from another G2 used during a G2-to-G2 connection. For information about such connections, see G2-to-G2 Interface.

Specifying a Default Update Interval

For variables only, the Default-update-interval attribute directs G2 to obtain a value for a variable at regular time intervals. For example, if the variable's Default update interval is 2 minutes, G2 obtains a value for the variable every two minutes, independent of whether it needs to for some other reason.

You can enter a subsecond value as a default update interval. A subsecond update interval is limited by the scheduler's update interval. For example, if the update interval for the scheduler is set to.5 seconds, setting a variable's update interval to less than that will not have any effect.

If a variable does not have a validity interval, G2 obtains a value for the variable only when necessary as Obtaining Requested Values describes.

History Keeping in G2

The history-keeping-spec attribute determines how to keep history for a variable or a parameter. When the value of this attribute is changed from the default do not keep history, G2 keeps a record of the variable or parameter values.

Storing and Accessing History Values

The reason for saving history is typically to:

- View historical information.
- Reason about history values.
- Store historical data.
- Retrieve historical data.

Using a G2 display such as a trend chart, you can monitor historical data as G2 collects it, or refer to the data in expressions and formulas after collecting history. History values are the basis for computing averages, minimum values, maximum values, rates of change, and so on, for a parameter or a variable. Thus, you can write rules and other statements that require knowledge of a history of events. Trend charts can display history values. For information about trend chart display items, Trend Charts.

The G2 system procedure g2-snapshot saves the current state of a KB with all of its transient data, including variable and parameter histories. For information about g2-snapshot, see Saving Permanent and Transient Data in Snapshot KBs. The saved KB can then be loaded later and warmbooted. For more information, see Warmbooting a KB Snapshot File.

The way you specify history keeping is identical in variables, parameters, and simulation variables.

The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices.

Collection Time

G2 stores history as a series of **data points**. A data point consists of the variable or parameter value and its corresponding **collection time**. Data points are stored as part of an item's transient knowledge.

In general terms, collection time refers to the time at which a variable or a parameter receives a value. The collection time is either:

- The time of collection as furnished to the variable from a source outside of G2, which can be G2 Gateway, GFI, or the G2 Simulator.

  GFI and the G2 Simulator are superseded capabilities. For more information, see Appendix F, Superseded Practices.

- The explicit collection time argument of a conclude action statement.

  The conclude action lets you conclude a value for a variable or a parameter with or without a specific collection time.

In the absence of an explicitly provided value, the collection time is the current subsecond time at the moment that the variable received a value.

For variables with an external data server, the scheduling interval has no effect. External data servers control the rate of updating the variable and thus its collection time in history values.

G2 returns the collection time as an integer if the time does not include a subsecond part, or a float if it does. Assign a collection-time value to a quantity local variable; or force the collection time value to an integer by using one of these functions: floor, ceiling, or truncate.

You can keep history data points in two different ways:

- A finite number of data points, for example:

  keep history with maximum number of data points = 100

- An unspecified number of data points limited by a period of time, which determines how long to keep the history values, for example:

  keep history with maximum age of data points = 1 day

To keep history values:

➔ keep history with maximum
{number of data points = *integer-expression* |
age of data points = *time-expression*}
[, with minimum interval between data points = *time-expression*]

Data points are zero based, 0 being the most recent and possibly the current value. To refer to the oldest history value, use the total number specified, less one. For example, if you define the maximum number of data points to keep as 20, you can refer to them as 0 – 19.

Saving a Maximum Number of Data Points

To keep history using a maximum number of data points:

➔ keep history with maximum number of data points = *integer-expression*

For example:

keep history with maximum number of data points = 100

Specifying history saving in this way causes G2 to allocate a finite amount of memory to accommodate the total number of data points to collect.

Saving Data Points over a Maximum Time Period

To keep history over a maximum period of time:

➔ keep history with maximum age of data points = *time-expression*

For example:

```
keep history with maximum age of data points = 2 days
```

The *time-expression* cannot be a subsecond interval.

When keeping history for a maximum time period, once the maximum number of data points are received within the specified interval, G2 does not remove older history values until a new value is added within the specified time. For example, if you keep history for 15 seconds, updating every 5 seconds, the total number of data points varies from 1 to 3 during the first 15 seconds. After the first 15 seconds, the number of data points remains steady.

Caution Keeping historical data can be costly in terms of memory usage. When you specify a maximum age of data points, G2 can consume indefinite amounts of storage space, until it either reaches the maximum age, or runs out of space. Running out of storage space could abort G2.

Saving a Maximum Number of Data Points over a Specific Time Period

To keep history combining both a finite number of data points and a specific time period:

➔ keep history with maximum number of data points = *integer-expression*
and maximum age of data points = *time-expression*

Be sure to specify the number of data points first, and then the time period. For example:

```
keep history with maximum number of data points = 100
and maximum age of data points = 2 days
```

Tip Changing the history keeping method from a maximum number of data points to a maximum period of time may not affect current history values until the next value is concluded into the variable or parameter. When new history values are collected *after* you change the history keeping method, the history is adjusted to the new interval.

Specifying a Minimum Interval between History Data Points

In addition to keeping history values as a specific number of data points or over a certain time period, you can append a minimum interval between data points statement to specify the amount of time between when G2 saves one data point

and the next. G2 still collects data points between those points, but the minimal interval determines which data points are saved.

The default minimum interval between history data points is one second. If more than one data point is received within a one-second interval, G2 keeps only the last one. If you do not include the minimum interval between history data points statement in the history-keeping-spec attribute, this default behavior remains. A one-second interval is the default.

By specifying the minimum interval between data points in different ways, G2 adjusts the collection times to the minimal interval. To keep only the last data point received, the collection time is canonicalized by the minimum interval specification. This provides history collection with a unique timestamp as its target for each specified time interval. For example, specifying the minimal interval between data points as 1 hour keeps only the last data point received in any 60 minute period, regardless of how many data points are collected.

Use the minimum interval between data points statement to keep history:

- On an event basis, maintaining all data points.
- As a single data point per interval, which could be subsecond or not.

Event-Based History

Setting the minimum interval between data points to 0 seconds directs G2 to save every data point collected (within the specified maximum number or age of data points). Using this setting provides the finest granularity for keeping history values.

Keeping a Single Data Point per Interval

To keep a single data point per interval, use the minimum interval between data points statement with any value other than zero (0). Whenever the interval is greater than zero, and more than one value is collected in the given interval, G2 keeps only the last value within an interval.

To keep subsecond history values, enter a value that divides evenly into a second. If you enter a value that does not divide evenly, G2 adjusts the value to an even interval. For instance, while .1,.2, and .5 are not rounded, entering a value of .4 seconds causes G2 to round the figure to .5 seconds, as .6, .7, .8, and .9 seconds will be round up to a 1 second interval. The minimum allowable value is 0.05 seconds and the maximum value is 6 days.

The subsecond value you enter in this part of the history-keeping-spec attribute actually determines the number of data points saved during a 1-second interval. The maximum number of data points that you can save per second is 20, the minimum is 1.

```
keep history with maximum age of data points = 1 minute and 10 seconds,
    with minimum interval between data points = 2 seconds
```

Values above 1 second must be integral second intervals, such as 2 seconds, 2 minutes and 5 seconds, or 1 hour.

For parameters, and variables whose data server is the inference engine, the scheduling interval can affect the granularity between collection times if it is set to an interval *greater* than the minimum interval between data points value.

For example, if the scheduler's minimum-scheduling-interval is set to .5 seconds, setting the history-keeping-spec attribute of a variable or a parameter to include a minimum interval between data points of .2 seconds will not have any effect. Instead, the minimum interval will be .5 seconds, corresponding to the scheduler interval.

Working with History Keeping Using Attribute Access

Using the attribute access facility, you can:

- View the history of a variable or parameter dynamically.
- Create or change the history keeping specification programmatically.

Displaying History Values Dynamically

To see a variable's history values and their collection times, use a Readout-table.

1. Create a readout-table item by selecting KB Workspace > New Display > readout-table > readout-table.
2. Edit the table of the readout-table.
3. In the expression-to-display attribute, enter the attribute of the item whose value you want to see, for example:

   ```
   the history of float-variable
   ```

The history values appear when the readout-display is updated:

History:	sequence(structure(history-value: 118.0, history-collection-time: 1363.0), structure(history-value: 132.0, history-collection-time: 1367.0), structure(history-value: 185.0, history-collection-time: 1371.0), structure(history-value: 164.0, history-collection-time: 1375.0), structure(history-value: 164.0, history-collection-time: 1379.0), structure(history-value: 133.0, history-collection-time: 1383.0), structure(history-value: 140.0, history-collection-time: 1387.0), structure(history-value: 159.0, history-collection-time: 1391.0), structure(history-value: 159.0, history-collection-time: 1395.0), structure(history-value: 136.0, history-collection-time: 1399.0))

Specifying History-Keeping Programmatically

Creating a variable programmatically and specifying its history-keeping-spec attribute requires knowledge of the attribute type. The history-keeping-spec attribute type is a symbol or structure. This example:

- Creates a float-variable, transfers it, makes it permanent, and concludes its name to be float-var.
- Changes the history-keeping-spec attribute from its default value do not keep history to a history keeping specification using the structure ( ) function.

```
conclude that the history-keeping-spec of float-var =
    structure(maximum-number-of-data-points: 10,
                minimum-interval-between-data-points: 1 minute)
```

Changing the History-Keeping Specification

You can change the value of the History-keeping-spec in several ways, two of which are shown next.

To conclude a new value into the subattribute directly:

➔ Use a conclude action, for example:

```
conclude that the maximum-number-of-data-points
    of the history-keeping-spec of float-var = 100
```

or

➔ Use the change-attribute function to create a new structure to conclude directly, for example:

```
conclude that the history-keeping-spec of V1 =
    change-attribute((the history-keeping-spec of V1),
                     maximum-number-of-data-points, 100)
```

See Structure Functions for a description of the change-attribute function.

Removing History Keeping

You can remove history keeping and the current history from either a variable or a parameter in two ways programmatically by:

- Concluding that the attribute has no value.
- Using the g2-clear-histories system procedure.

You can clear all history values from a parameter and to set the initial value and collection time by using the g2-initialize-parameter system procedure.

To remove the history specification of a variable or parameter:

➔ conclude that the history-keeping-spec of float-var has no value

For information about system procedures, see the *G2 System Procedures Reference Manual*.

History Expressions

G2 provides many expressions that reason about a variable or parameter history values.

Most history expressions require you to specify a *time-expression*, consisting of time units (seconds, minutes, hours, days, or weeks) or an arithmetic expression. If the time interval is an arithmetic expression, G2 interprets its result as a number of seconds. Some examples of valid time intervals are:

```
7 weeks, 6 days, 5 hours, 4 minutes, and 3 seconds
10 minutes
16 days and 2 hours
the current time + 10
3*4 seconds
```

Note When using the between *time-expression* ago and *time-expression* ago phrase in these expressions, you must specify the longer time interval before the shorter time interval.

Each of these expressions calculates a statistic, using the history values stored in a quantitative variable (or its subclasses) or a quantitative parameter (or its subclasses).

Note To create a rule that is invoked when the value of a history expression changes, use a whenever rule. For efficiency reasons, G2 does not invoke an if rule that is configured to forward chain when the rule antecedent refers to a history expression.

Obtaining a History Value

To obtain a history value:

➔ the value of {*variable* | *parameter*}
{as of *time-expression* ago | as of *integer-expression* datapoints ago}
-> {*integer* | *float* | *symbol* | *text* | *truth-value*}

For a variable or parameter that is keeping history values, this expression produces the value that the variable or parameter had at the specified time interval ago or at a number of datapoints ago.

Two examples are:

```
conclude that Q1 = the value of F1 as of 1 hour ago
```

```
conclude that A1 = the value of F1 as of 10 datapoints ago
```

Specifying a time interval causes G2 to return the value from the specified time. If G2 did not collect a value then, it returns the value collected immediately prior to the specified time-interval. For instance, in the example shown here (as of 1 hour ago), if no value were collected at that time, G2 would return the value collected before that time, perhaps 1 hour and 1 minute ago. If no previous time existed, G2 returns no value, expressed as **** (asterisks). Note that this behavior differs from the behavior of the interpolated value of expression.

If you specify a number of data points ago, as in the second example, G2 returns the value at that data point. History data points are zero-based, meaning that if 10 data points are collected, the number of data points ranges from 0 – 9, and specifying 10 is invalid.

If G2 did not collect a value at the specified data point, it returns no value, expressed as **** (asterisks).

If you specify a number of history values ago, G2 produces that history value. For example:

```
the value of the temperature of tank-3 as of 36 datapoints ago
```

In this example, if G2 did not collect a value at the temperature attribute's 36th history value, the expression produces a *no value* condition.

Computing the Number of History Datapoints

To obtain the total number of history datapoints within, and on, the boundaries of the time interval you specify:

➔ the number of history datapoints in {*variable* | *parameter*}
{during the last *time-expression* |
between *time-expression* ago and *time-expression* ago }
-> *integer*

This expression produces the number of history values contained in the specified variable or parameter, or the number of history values received during the specified time interval. G2 counts the values within and on the boundaries of the specified time interval.

Here are two examples:

```
conclude that Q1 =
    the number of history datapoints in F1 during the last 12 hours

conclude that Q1 =
    the number of history datapoints in F1
    between 1 hour ago and 1 minute ago
```

Computing the Average History Value

To compute the average history value over a specified period of time:

➔ the average value of {*quantitative-variable* | *quantitative-parameter*}
{during the last *time-expression* |
between *time-expression* ago and *time-expression* ago}
-> *float*

If the specified variable or parameter contains at least one history value for the specified time interval, this expression produces a value that G2 calculates based on those history values. If the specified variable or parameter does not contain at least one history value, this expression produces a *no value* condition.

The calculated average is the sum of the variable's or parameter's values divided by the number of values. G2 does not calculate a time-weighted average, in which the distribution of values over time can produce differing results.

Here are two examples:

```
conclude that Q1 =
    the average value of F1 during the last 10 minutes

conclude that Q1 =
    the average value of F1 between 5 minutes ago and 10 seconds ago
```

Tip G2 does not compute a time-weighted average where the distribution produces differing results. This average value is simply the sum of the variable or parameter's values divided by the number of values.

Computing the Sum of Values in Histories

To compute the sum of values in histories:

➔ the sum of the values of *quantitative-var-or-param*
during the last *time-expression*

➔ the sum of the values of *quantitative-var-or-param* between
time-expression ago and *time-expression* ago

For example, this procedure sums the values of the parameter named param during the last 1 hour:

```
sum-of-values()
q1: float;
begin
    q1 = the sum of the values of param during the last 1 hour;
post "[q1]";
end
```

This procedure sums the values of param between 30 and 10 seconds ago:

```
sum-of-values-over-interval()
q1: float;
begin
    q1 = the sum of the values of param between 30 seconds ago and 10 seconds
ago;
post "[q1]";
end
```

Computing the Integral

To compute the integral over a selected period of time:

➔ the integral in {seconds | minutes | hours | days | weeks}
of {*quantitative-variable* | *quantitative-parameter*}
{during the last *time-interval* |
between *time-interval* ago and *time-interval* ago}
-> *float*

If the specified variable or parameter contains at least one history value for the specified time interval, this expression produces a value that G2 calculates based on those history values. If the specified variable or parameter does not contain at least one history value, this expression produces a *no value* condition.

Two examples are:

```
conclude that Q1 =
    the integral in seconds of F1 during the last 10 seconds

conclude that Q1 =
    the integral in minutes of F1 between 1 hour ago and 1 minute ago
```

G2 performs a true integration over time of the most recently received value for the variable or parameter at each instant of the specified time interval. For all history values falling into the specific time interval, suppose we have the value series as $v_1, v_2, v_3, \ldots v_n$ with correspond time series as $t_1, t_2, t_3, \ldots t_n$, then G2 uses following formula to compute the integral:

$$\left[\sum_{i=1}^{n} v_{i-1} \cdot (t_i - t_{i-1}) \right] / u$$

In above formula, t_0 is the start time of the specific time interval, v_0 is always 0.0, and *u* is the time unit in seconds. For example, using in seconds of we have the unit as 1, while in minutes of has the unit of 60, because one minute has 60 seconds.

Computing the Interpolated Value

To compute the interpolated value of a variable or a parameter:

➔ the interpolated value of {*quantitative-variable* | *quantitative-parameter*}
    as of *time-interval* ago
    -> *float*

If the specified variable or parameter contains at least two history values for the specified time interval, this expression produces an interpolated value that G2 calculates based on those history values. If the specified variable or parameter does not contain at least two history values, this expression produces a *no value* condition.

If G2 did not collect a value for the variable or parameter at the time interval you specify, it performs straight-line interpolation of the values that were collected immediately before and after that time interval. For instance, if the expression states 1 hour ago and collections occurred at 50 minutes and 1 hour 5 minutes ago, G2 interpolates those two values. In this way, interpolation differs from the value of expression, which returns the value received *prior* to the specified time interval, if no value was received as of the specified time interval.

Here is an example:

```
conclude that Q1 =
    the interpolated value of F1 as of 1 hour ago
```

Note When obtaining interpolated values at a maximum time period value, it is possible for the value to not be returned. For example, if you keep history for 30 seconds, and then try to obtain the interpolated value of the variable as of 30 seconds ago, G2 returns no value.

Computing Maximum and Minimum Values

To compute the maximum or minimum value of a variable or a parameter over a period of time:

➔ the {maximum | minimum} value of
{*quantitative-variable* | *quantitative-parameter*}
{during the last *time-interval* |
between *time-interval* ago and *time-interval* ago}
-> {*integer* | *float*}

If the specified variable or parameter contains at least one history value for the specified time interval, this expression produces the maximum or minimum of the history values received within the specified time interval. If the specified variable or parameter does not contain at least one history value within the specified time interval, this expression produces a *no value* condition.

Two examples are:

```
conclude that Q1 =
    the maximum value of F1 during the last 1 day

conclude that Q1 =
    the minimum value of F1 between 5 minutes ago and 5 seconds ago
```

Computing the Rate of Change

To compute the rate of change within a particular time unit over a time period:

➔ the rate of change per {second | minute | hour | day | week}
of {*quantitative-variable* | *quantitative-parameter*}
{during the last *time-interval* |
between *time-interval* ago and *time-interval* ago}
-> *float*

If the specified variable or parameter contains at least one history value for the specified time interval, this expression produces a rate of change statistic that G2 calculates based on those history values. If the specified variable or parameter does not contain at least one history value, this expression produces a *no value* condition.

The expression calculates the rate of change per second, minute, hour, day, or week of the specified variable or parameter over the specified time interval. G2

calculates the rate of change by dividing the total accumulated change during the specified time interval by the length of the time interval.

Two examples are:

```
conclude that Q1 =
    the rate of change per minute of F1 during the last 2 hours

conclude that Q1 =
    the rate of change per hour of F1 between 48 hours ago and 24 hours ago
```

G2 computes the rate of change by dividing the total accumulated change during the time period by the length of the time period. For all history values falling into the specific time interval, suppose we have the value series as $v_1, v_2, v_3, \ldots v_n$ with correspond time series as $t_1, t_2, t_3, \ldots t_n$, G2 actually use only the first two and the last two values to get the rate of change. If all four values were available, G2 uses follow formula to get the result:

$$\frac{v_b - v_a}{t_b - t_a} \cdot u$$

in which

$$v_a = \frac{v_1 + v_2}{2}, v_b = \frac{v_{n-1} + v_n}{2}, t_a = \frac{t_1 + t_2}{2}, t_b = \frac{t_{n-1} + t_n}{2}$$

and *u* is the time unit in seconds. For example, using per second we have the unit as 1, while per minute has the unit of 60, because one minute has 60 seconds.

If there's just three history values, then above formula will be simplified into following form:

$$\frac{v_3 - v_1}{t_3 - t_1} \cdot u$$

With just two history values, it will be

$$\frac{v_2 - v_1}{t_2 - t_1} \cdot u$$

And if there's just one history value falling into the time interval, the result is 0.

Computing the Standard Deviation

To compute the standard deviation of the value over a selected period of time:

➔ the standard deviation of {*variable* | *parameter*}
    {during the last *time-interval* |

between *time-interval* ago and *time-interval* ago}
-> *float*

If the specified variable or parameter contains at least one history value for the specified time interval, this expression produces a standard deviation statistic that G2 calculates based on those history values. If the specified variable or parameter does not contain at least one history value within the specified time interval, this expression produces a *no value* condition.

Two examples are:

```
conclude that Q1 =
    the standard deviation of F1 during the last 1 minute
```

```
conclude that Q1 =
    the standard deviation of F1 between 1 minute ago and .5 seconds ago
```

For all history values falling into the specific time interval, suppose we have the value series as $v_1, v_2, v_3, \ldots v_n$, then G2 uses following formula to compute the standard deviation:

$$\sqrt{\left|\frac{\sum_{i=1}^{n} v_i^2}{n} - \bar{v}^2\right|}$$

in which

$$\bar{v} = \frac{\sum_{i=1}^{n} v_i}{n}$$

Concluding the History Directly

To conclude the history of a variable or parameter directly:

➔ the {history | history-using-unix-time} of {*variable* | *parameter*}

This example shows how to conclude the value of the history hidden attribute of one variable or parameter into the history hidden attribute of another variable or parameter. When concluding histories this way, the value of the history-collection-time subattribute of the history structure is based on the G2 start time. For example:

```
conclude that the history of var-1 = the history of var-2
```

You can also conclude the history directly, where the history-collection-time is based on the current UNIX time, as follows:

```
conclude that the history-using-unix-time of var-1 =
the history-using-unix-time of var-2
```

For information on the subattributes of the history and history-using-unix-time hidden attribute structures, see the *G2 Class Reference Manual*.

Actions to Use with Variables and Parameters

Several actions are specifically for use with variables and/or parameters. For a complete discussion of all G2 actions, see Actions.

Concluding an Attribute Variable to Have No Value

To conclude that an attribute variable has no value:

➔ conclude that the *attribute* of *item* has no value

For example:

```
conclude that the velocity of signal-load has no value
```

where velocity is an object attribute given by a float-variable.

Concluding Values for Variables and Parameters

Use the conclude action to change the value of any variable or parameter. The variable or parameter must be active at the time the conclude action is executed.

Concluding values for variables or parameters may cause forward chaining, for example by triggering an event that invokes a whenever rule awaiting a value.

You use identical statements to conclude values for variables or parameters, except that variables can include the with expiration statement. Both variables and parameters can include the with collection time statement.

Use this phrase...	To specify...
with expiration	An expiration time for a value. G2 then sets the expiration time of the variable to the value of the expression. If you do not specify an expiration time, G2 sets the variable's expiration time according to its validity-interval attribute.
with collection time	The collection time for a variable or parameter. If the collection time you specify is earlier than the most recent collection time established by G2, G2 inserts the value into the history. If you do not specify a collection time, the collection time for the variable is the time at which G2 executes this action.

For a variable, you can optionally specify both an expiration time and a collection time, which effectively changes the validity interval of that value. This has no effect on the value of the validity-interval attribute for the variable.

Concluding a Logical Value

To conclude a value for a logical variable or parameter:

➔ conclude that {*logical-variable* | *logical-parameter*}
{= *truth-value-expression* | is {true | false} }

This action concludes a variable or parameter value of true, false, or what the *truth-value* evaluates to. As a shortcut, you can also set the value of a local variable or parameter to true or false as follows:

conclude that [not] {*logical-variable* | *logical-parameter*}

This grammar sets the variable or parameter to true unless not is specified, in which case it sets the variable or parameter to false. Despite its appearance, specifying not does not invert the value of the variable or parameter; it just sets the value to false.

Some examples are:

conclude that the there-is-enough-heat of heat-exchanger is true

conclude that not the there-is-enough-heat of heat-exchanger-2

conclude that the there-is-enough-heat of heat-exchanger-2 = false

The third example is preferable to the second, because its meaning is self-evident.

Concluding a Quantitative Value

To conclude a value for a quantitative variable or parameter:

➔ conclude that {*variable* | *parameter*} = *value-expression*

This action concludes that the value of a quantitative, integer, or float variable or parameter is the value of *value-expression*. The quantitative variable or quantitative parameter can be a subclass of either, or a user-defined attribute containing the same.

Two examples are:

```
conclude that the interior-temperature of tank-1 =
    the exterior-temperature of tank-1 + 30

conclude that the area of floor-5 =
    the length of floor-5 * the width of floor-5
```

Concluding a Symbolic Value

To conclude a value for a symbolic variable or parameter:

➔ conclude that {*symbolic-variable* | *symbolic-parameter*}
{ = *symbolic-expression* | is *symbol* }

This action concludes the value of the *symbolic-expression* into the symbolic variable or parameter, or any attribute that receives its value from a symbolic variable or parameter. This example replaces the value of the symbolic variable that represents the tank attribute with a new symbol, filling.

```
conclude that the tank of the gas-tank connected to oil-tanker is filling
```

Concluding a Text Value

To conclude a value for a text variable or parameter:

➔ conclude that {*text-variable* | *text-parameter*} = *text-expression*

This action changes the value of the *text-variable-or-parameter* to the value of the *text-expression*. The *text-variable* or *text-parameter* is any variable or parameter, or subclass of either, or any attribute that receives its value from a text variable or text parameter.

Two examples are:

```
conclude that the comments of gas-tank = "Remember to replace the valve!"

conclude that name-response = last-name
```

Concluding That a Variable Does Not Have a Value

To conclude that a variable does not have a value or a current value:

➔ conclude that *variable* has no [current] value

The *variable* can be any variable or any attribute that receives its value from a variable. If current appears, this action concludes that the value of *variable* is expired. If current is omitted, the action concludes the *variable* has no value at all, expired or otherwise.

Using the has no value statement causes the variable's last-recorded-value attribute to show no-value, rather than an expired value. The variable maintains its history, if it has one. Using the has no current value statement causes the variable's last-recorded-value attribute to display the value with an asterisk beside it, indicating that the value is expired. If the value of the variable is already expired, G2 does nothing.

Variable and Parameter Rules

The three whenever rules for use with variables and parameters are:

Whenever a Variable or Parameter Receives a Value

G2 can detect whenever a variable or parameter receives a value with a whenever rule as follows:

whenever {*variable* | *parameter* | *attribute*} receives a value}

An indefinite validity interval for a variable affects how and when G2 invokes a receives a value whenever rule. For more information, see Using an Indefinite Interval.

Whenever a Variable Fails to Receive a Value

G2 can detect when a variable fails to receive a value with a whenever rule as follows:

whenever *variable* fails to receive a value

Whenever a Variable Loses Its Value

G2 can detect when a variable loses its value with a whenever rule as follows:

whenever *variable* loses its value

Variable and Parameter Expressions

These expressions refer to the knowledge associated with variables and parameters.

Expressions that refer only to the valid current value of a variable cause G2 to perform data seeking to obtain a new current value if the specified variable has never had a value or if its current value has expired.

A reference to a variable or parameter produces the same value everywhere it is used in the same expression. Also, a reference to a variable or parameter produces the same value everywhere it is used within the same transaction scope.

Directly Referring to a Variable or Parameter

To refer directly to a variable or a parameter:

➔ {*quantitative-variable* | *quantitative-parameter*}
-> *integer* | *float*}

➔ {*integer-variable* | *integer-parameter*}
-> *integer*

➔ {*float-variable* | *float-parameter*}
-> *float*

➔ {*symbolic-variable* | *symbolic-parameter*}
-> *symbol*

➔ {*text-variable* | *text-parameter*}
-> *text*

➔ {*logical-variable* | *logical-parameter*}
-> *truth-value*

Refer to a variable or parameter to produce its current value in a larger expression. If a variable has no current value, in some contexts, such as in rules and the collect data statement in procedures, G2 performs data seeking to attempt to obtain a new current value. See the discussion of data seeking in Obtaining Values for Variables.

For example, suppose you have this rule:

```
for any tank T upon this workspace
    if the temperature of T > 100 then
    conclude that the latest-temperature of master-display = the temperature of T
```

In this case, the temperature attribute for the user-defined tank class is defined as given by a variable. If the temperature of tank-1 has a current value, the conclude action in the rule's consequent uses that value as the new value of the latest-temperature attribute of master-display. If the temperature of tank-1 does not have a current value, referencing it causes G2 to perform data seeking.

Since variables do not always have a value, you cannot refer to them directly in procedures as you can a parameter. Attempting to do so causes G2 to signal an error. Instead, you can access the value of a variable within a procedure by using a collect data statement or within a the current value of expression.

A collect data statement causes data seeking if the variable does not have current value. For more information about how G2 performs a collect data operation, see collect data.

Using the Value of Expression

To refer to the value of a variable:

➔ the value of *variable*
-> {*integer* | *float* | *symbol* | *text* | *truth-value*}

This expression produces a valid current value of the specified variable. If its current value has expired, G2 performs data seeking to obtain a new current value. If an attempt at data seeking fails for any reason, evaluating this expression produces a *no value* condition.

The expiration time interval of this expression is the expiration time interval of the current value of the specified variable. For example:

```
the value of the temperature of tank-1
```

Using the Has a Value Expression

To determine if a variable has a value:

➔ *variable* has {a | no} value
-> *truth-value*

This expression produces a truth-value that indicates whether a variable has a valid current value or whether G2's most recent attempt at obtaining a new current value succeeded. For example:

```
the temperature of tank-1 has a value
```

In this case, if the temperature of tank-1 has a valid current value, this expression produces the truth-value true. If the temperature of tank-1 has never had a value, or if its current value has expired, this expression causes G2 to perform data seeking to obtain a new current value. If G2 obtains a value from that data seeking, the expression produces the truth-value true; otherwise, the expression produces the truth-value false.

Using Current Value Expressions

A variable has a current value when it has a value that has not yet expired. Current value expressions do *not* cause data seeking, even if the specified variable has no current value.

When a variable receives a new current value, G2 determines an expiration time for that value based on whether it was explicitly specified (via the conclude ...

with expiration action) or by combining the collection time of the new value and the variable's validity interval.

To obtain a variable's current value:

➔ the current value of *variable*
-> {*integer* | *float* | *symbol* | *text* | *truth-value*}

This expression refers to a variable's current value, but does *not* cause data seeking if the variable has never had a value or if its current value has expired. If the specified variable has a current value, this expression produces it. Otherwise, the expression produces a *no value* condition. For example:

```
the current value of the temperature of tank-1
```

If this expression produces a value, the expiration time of the expression is the expiration time of the current value. If this expression produces a *no value* condition, the expiration time is also a *no value* condition.

To determine whether a variable has a current value:

➔ *variable* has {a | no} current value
-> *truth-value*

This expression produces a truth-value that indicates whether a variable has a current value. For example:

```
the temperature of tank-1 has a current value
```

In this case, if the temperature attribute of tank-1 has a current value, this expression produces the truth-value true. If the temperature attribute of tank-1 has never had a value, or if its current value has expired, this expression produces the truth-value false.

If this expression produces the truth-value true, the expiration time of this expression is the expiration time of the current value. If this expression produces the truth-value false, the expiration time is indefinite.

If a has a value expression is nested within a has a current value expression, G2 evaluates the nested expression no differently than a has a value expression. G2 performs data seeking, if necessary, to obtain a new current value for the variable referenced in the nested has a value expression.

Obtaining the Simulated Value of a Variable or Parameter

The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices.

Obtaining the Collection Time for a Variable or Parameter

To obtain the collection time for a variable or parameter:

➔ the collection time of {*variable* | *parameter*}
[as of *integer-expression* datapoints ago]
-> *float*

This expression produces a float value that represents the time in seconds when the specified variable or parameter received its most recent value. This time value is based on the G2 clock, so it might not represent the passage of real time.

If you specify this expression without the optional as of *integer-expression* datapoints ago phrase, for the expression to produce a value, the specified variable or parameter need not have a current value, but must have received at least one history value. Otherwise, this expression produces a *no value* condition. For example:

```
the collection time of custom-variable
```

In this case, if the variable custom-variable has received at least one history value, this expression produces a float value that represents the time in seconds at which the most recent history value was received.

If you specify this expression with the optional as of *integer-expression* datapoints ago phrase, then for the expression to produce a value, the specified variable or parameter need not have a current value, but must have received at least the specified number of history values plus one.

Otherwise, this expression produces a *no value* condition. The most recently received history value is zero (0) history values ago. For example:

```
the collection time of custom-variable as of 10 datapoints ago
```

If the variable custom-variable has received at least 11 history values, this expression produces a float value that represents the time in seconds at which the value 10 history values ago was received.

The precision of the collection time depends on the specification of the minimum interval between data points of the history-keeping-spec. For details, see Specifying a Minimum Interval between History Data Points.

Obtaining the Expiration Time for a Variable

To obtain a variable's expiration time:

➔ the expiration time of *variable*
-> *integer*

This expression produces an integer value that represents the expiration time of the specified variable's most recently received history value. If the specified

variable has never received a value, or if its validity-interval attribute is set to indefinite, this expression produces a *no value* condition. For example:

```
the expiration time of custom-variable
```

In this case, if the variable custom-variable has received at least one history value, this expression produces an integer value that represents the time in seconds at which the variable's current value expires.

Referring to a Variable or Parameter That Gives the Value of an Attribute

To refer to a variable or parameter giving the value of an attribute:

➔ the {*variable* | *parameter*} giving
the *attribute-name* of {*object* | *connection* | *message*}
-> {*variable* | *parameter*}

This expression produces the child variable or parameter that gives the value of an attribute of an object, connection, or message of a user-defined class. This expression refers to the child variable or parameter as an item, rather than to the value of that variable or parameter. For example:

```
if the status of the temperature-variable TV
    giving the temperature of tank-1 is broken
    then post "The temperature sensor [the public-name of TV] is broken."
```

This expression refers to the status attribute of the temperature-variable item that gives the value of the temperature attribute of tank-1.

Referring to a Time Interval Ending with the Collection Time

To refer to a time interval ending with the collection time:

➔ *history-expression* of *variable-or-parameter* during the *time-expression*
ending with the collection time

where:

- *history-expression* is one of the following:

  ```
  the value of
  the number of history datapoints in
  the average value of
  the integral in {seconds | minutes | hours | days | weeks} of
  the interpolated value of
  the {maximum | minimum} value of
  the rate of change per {seconds | minutes | hours | days | weeks} of
  ```

the standard deviation of
the sum of the values of

- *variable-or-parameter* is a quantitative variable or parameter, except when using the number of history datapoints in, in which case it can be any type of variable or parameter.

For example, the following procedure computes the average value of a parameter during the 30 minutes ending with the collection time:

```
compute-average()
q1: float;
begin
    q1 = the average value of param during the 30 minutes ending with
         the collection time;
    post "[q1]";
end
```

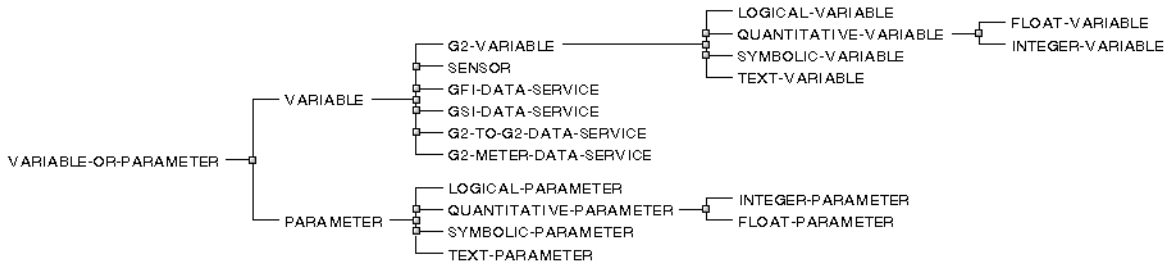
The Variable and Parameter Classes

Within the G2 system-defined class hierarchy, variable and parameter are subclasses of the variable-or-parameter class of objects. The variable and parameter classes are reserved, abstract classes, as is the g2-variable class from which each of the variable type classes stem.

The variable and parameter type classes are:

G2-Variable Subclasses	Parameter Subclasses
logical-variable	logical-parameter
quantitative-variable	quantitative-parameter
float-variable	float-parameter
integer-variable	integer-parameter
long-variable	long-parameter
symbolic-variable	symbolic-parameter
text-variable	text-parameter

You can create subclasses from any of those classes.



In addition to the value type subclasses, the variable hierarchy includes these other classes:

- gfi-data-service
- gsi-data-service
- g2-to-g2-data-service
- g2-meter-data-service
- sensor

GFI is a superseded capability. For more information, see Appendix F, Superseded Practices.

The sensor class has been replaced by the g2-variable subclasses. While G2 supports sensors for use in existing KBs, we recommend that the sensor class not be used in new KB development.

Each of the data service classes is a mixin class that adds attributes to G2 variable items to receive their value from another process. which adds class attributes necessary for G2 variable data For a description of mixin classes and how to use them, see Using Mixin Classes.

Note While G2 permits the use of parameters and variables of the non-specific type quantitative, for efficiency, we recommend that you use an item of specific data type, integer or float, whenever possible.

Variables and parameters have these class-specific attributes:

Attribute	Variable	Parameter
options	✓	✓
tracing-and-breakpoints	✓	✓
data-type	✓	✓
initial-value	✓	✓

Attribute	Variable	Parameter
last-recorded-value	✓	✓
history-keeping-spec	✓	✓
validity-interval	✓	
formula	✓	
simulation-details	✓	
initial-value-for-simulation	✓	
data-server	✓	
default-update-interval	✓	

The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices.

Common Attributes

The common attributes of variables and parameters are:

Attribute	Description
options	Controls whether or not G2 can backward or forward chain or data seek from the variable, or forward chain from the parameter.
Allowable values:	See Specifying Forward and Backward Chaining
Default value:	See Specifying Forward and Backward Chaining
tracing-and-breakpoints	Sets tracing and breakpoints, overriding the tracing-message-level and breakpoint-level attributes in the Debugging Parameters system table.
Allowable values:	See the description in Debugging Parameters
Default value:	default

Attribute	Description
data-type	Indicates the variable's data type or, for numeric variables, the units of measure in use.
Allowable values:	See Specifying the Type
Default value:	See Specifying the Type
initial-value	The initial value (if any)
Allowable values:	Any valid value of an applicable data type for the item.
Default value:	See Specifying an Initial Value
last-recorded-value	The value G2 obtained for the variable or parameter
Allowable values:	Any valid value of an applicable data type for the item
Default value:	For variables: no value For parameters: See Specifying an Initial Value
history-keeping-spec	Describes whether to maintain a history of the values for this item and if so, in which manner.
Allowable values:	See History Keeping in G2
Default value:	do not keep history

Variable-Specific Attributes

The class-specific attributes of variables are:

Attribute	Description
validity-interval	The length of time that the last-recorded-value remains current.
Allowable values:	any *time-interval* indefinite supplied
Default value:	supplied
formula	A formula through which to calculate the variable value.
Allowable values:	Any logical-, arithmetic-, symbolic-, or text-expression
Default value:	none
simulation-details	Specifies what simulation is in effect.
Allowable values:	See Appendix F, Superseded Practices
Default value:	no simulation formula yet
Notes:	The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices.
initial-value-for-simulation	The initial value for simulated variables that use discrete-state and continuous-state simulation formulas.
Allowable values:	any number default
Default value:	default
Notes:	The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices.

Attribute	Description
data-server	The data server for this variable.
Allowable values:	See Specifying a Data Server
Default value:	inference engine
default-update-interval	The interval at which G2 should collect a value for the variable.
Allowable values:	any non-negative number none
Default value:	none

Value-Structure and History Hidden Attributes

The variable-or-parameter class defines a hidden attribute called value-structure, which indicates the current value and collection-time:

structure
(data-point-value: *quantity*,
data-point-collection-time: *quantity*)

It also defines a hidden attribute called history, which is a sequence of structures, where each structure indicates the value and collection-time for each update in the history:

structure
(history-value: *quantity*,
history-collection-time: *quantity*)

When a variable or parameter keeps a history, the value-structure is simply a copy of the latest update to its history hidden attribute.

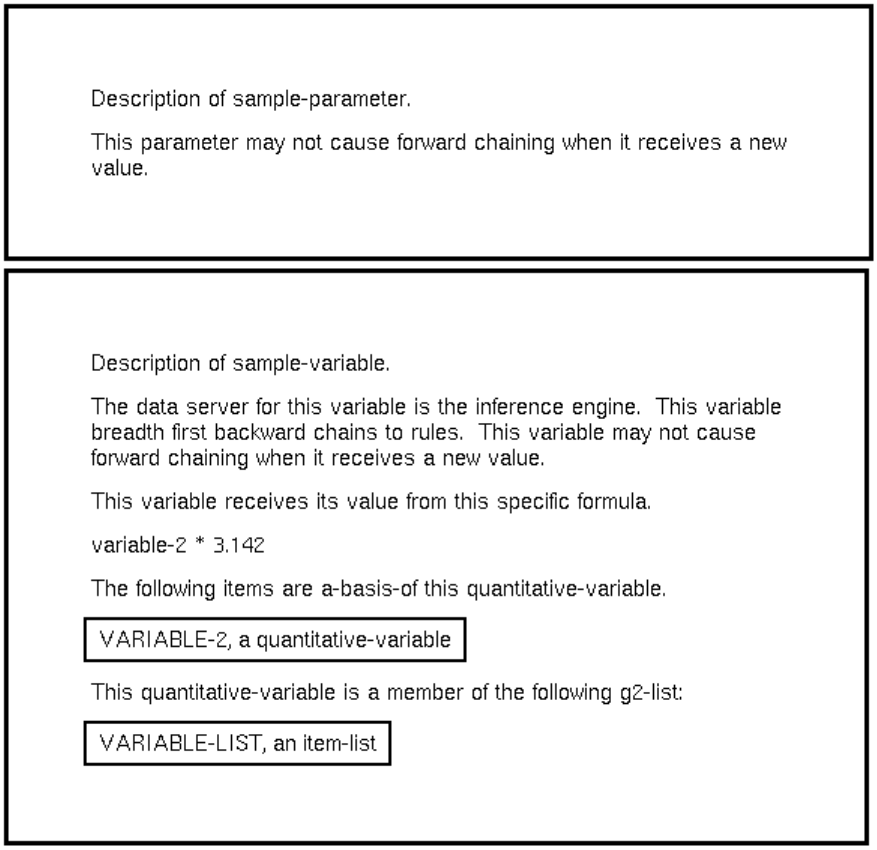
When a variable or parameter does not keep a history, the value-structure provides a way of keeping track of the latest value and collection-time of the variable or parameter.

Describing Variables and Parameters

When you select describe for a variable or a parameter, G2 displays a temporary workspace containing the system-defined describe information for any item, as well as additional information that applies only to variables and parameters:

- Forward chaining: Whether forward chaining to rules from the variable or parameter is selected.
- Generic formula: Any generic formula from which the parameter receives a value.
- Specific formula (variables only): The variable may either contain the specific formula within its attribute table or be referenced by it.
- Rules it forward chains to: Any rules to which G2 could forward chain from the variable or the parameter (if forward chaining is enabled).
- Data server (variables only): The data server of the variable.

Here are describe workspaces of a parameter and a variable. The description of sample-variable contains a short reference to variable-2 and a short reference to variable-list. Short references are shortcuts to access items related to the variable in some way.



16

Lists and Arrays

Describes how to use lists and arrays.

Introduction

Lists and arrays are instances of the g2-list and g2-array classes, respectively. They consist of a series of **elements** whose data types depend on the class of the list or array. Lists and arrays can consist of:

- Items only.
- Values of a single simple data type, such as logical, quantity, float, integer, or text.
- Values of different data types, such as simple and composite value data types.
- Both items and values, using the item-or-value data type.

You can reference list and array elements positionally, using an integer **element index**. The index of the first element of a list or array is zero.

KB Saving of Permanent Lists and Arrays

G2 lists and arrays can have item elements that do not reside in the same module as the list or array item. When G2 saves a permanent list or array with the KB, it saves the UUID of any cross-module item element with the list or array item so that the item element can be reinserted in the list or array at KB load time.

For the successful saving and reloading of permanent lists and arrays that contain references to items in other modules, G2 relies on the item element UUIDs being unique and stable across all the modules in your KB. Therefore, it is essential to refrain from changing item UUID values when only a subset of your KB modules are resident in G2 in order to prevent the failure of array and list element insertion at KB load time.

For more information on UUID attributes, see Using Universal Unique Identifiers.

Lists and Sequences

The sequence value type is a list-like entity whose elements can consist of items and values, including other sequences and structures. Using a sequence to store item references and values you can:

- Reason about its elements.
- Save the sequence as permanent knowledge in your KB.

The use of sequences is described in more detail in Attribute Access Facility.

Note Be careful not to confuse the sequence data type with the general concept of a sequence of elements kept in a list or array, as described in this chapter.

Comparing Lists and Arrays

You create lists and arrays as placeholders for items and values in your KB. The two items have similar behavior but are best suited for different purposes. The choice of whether to use a list or an array can be made easily once the differences between both items are clear. This section offers a brief description of lists and arrays, and concludes with a table of their differences.

Choosing Lists

Lists are objects that contain a group of one or more elements. Lists have no fixed length. A list grows or shrinks in length as you insert or remove elements, and the numeric position (or index) of any element changes as G2 inserts or removes elements closer to the beginning of the list. Use lists to implement:

- Queues.
- Push-down stacks.
- Any sequence of items in which the number of elements changes frequently with insertions and deletions.

G2 provides expressions to reference certain list elements, using the position near the beginning (first and second) or the end (next-to-last, last) of the list, or adjacent to another element (before, after).

You can also access list elements by using an element index into the list. However, unlike an array index operation, which evaluates to a position and thus accesses the array element directly, a list element index causes G2 to perform a linear search from the beginning of the list to the specified *item-or-value* referenced by the index.

Choosing Arrays

Arrays are objects that contain a sequence of zero or more elements. Arrays have a fixed length, which you declare in the array-length attribute of the array. You can change the array length interactively by editing the attribute, or programmatically by using the change action.

Changing the length of an array interactively or programmatically using the conclude grammar is a permanent change. Changing the length programmatically using the change grammar is temporary: resetting the knowledge base causes the array length to revert to its original value.

When you shorten the length of an array while it is active, the high-end contents are lost. When you lengthen an array while it is active, G2 provides initial values for the new elements.

Arrays are more efficient than lists for referencing an indexed element. In contrast to lists, arrays do not require a linear search for their elements. G2 uses an array

index to locate an array element directly. Use arrays when you need quick access to a set number of elements.

Using Nested Arrays

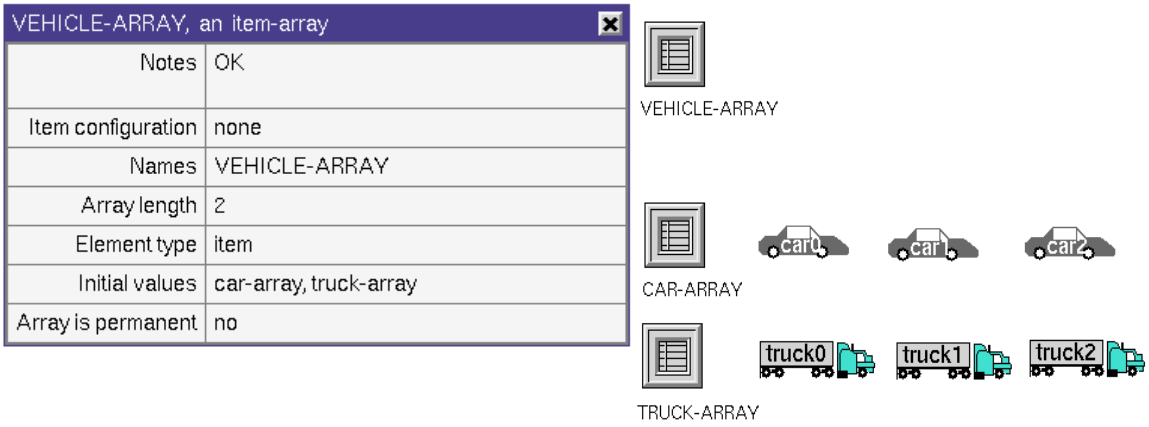
While arrays in G2 are one-dimensional, you can nest them so that they function as multi-dimensional arrays. For example, you can create an array of arrays and reference its elements using a double index. The array of arrays then functions as a **two-dimensional array**.

The next diagram illustrates an item array, vehicle-array, whose two elements are item arrays: car-array is element 0 and holds three cars and truck-array is element 1 and holds three trucks.

To reference the second car in that array, use the expression:

```
vehicle-array [ 0 ] [ 1 ]
```

where the first index ([0]) references car-array, and the second index ([1]) references car1.



List or Array Contents

When a list or array is populated with item elements, the item elements consist of item references. The items are not a part of the list or array. Deleting an item from a list or array does not delete the item, but removes the item reference. Because item elements are references, a single item can be a member of several lists or arrays, and can appear more than once in the same list. In contrast, value list and array elements are not references, and exist as part of the list or array.

By default, neither item references nor value elements are a permanent part of a list or array: lists are cleared upon reactivation, and arrays are re-initialized. A list or an array is also cleared when its workspace is disabled or it is transferred to a disabled workspace. Such a list or array is called a **transient-membership** list or array.

You can optionally set a list or array to retain references and elements as permanent knowledge. Such a list or array is called a **permanent-membership** list or array. For details, see Using Permanent-Membership Lists and Arrays.

Alternatively, you can save the elements of transient-membership lists and arrays by using the g2-snapshot system procedure, and then load the snapshot KB, using the warmboot option. A snapshot of a KB saves all existing data, transient and permanent, which you can restore later during a warmboot load. For information about saving and loading a KB this way, see Saving Permanent and Transient Data in Snapshot KBs and Warmbooting a KB Snapshot File.

Effect of Run States on Lists and Arrays

As with most G2 items, you can reference lists and arrays only when they are active. This means that expressions and actions involving lists and arrays execute only when the list or array is active. In addition, items must be active before you can add them to a list or array. For a description of G2 run states, see Operating the Current KB.

With transient membership...	With permanent membership...
Inactive lists are empty. Inactive arrays contain their initial values.	Inactive lists retain their elements after they have been populated. Inactive arrays contain their initial values, if they have not been previously populated, or their elements if they have.

Activation and deactivation affects lists and arrays in this way:

This action...	In a list or array with transient membership...	In a list or array with permanent membership...
Deactivating list or array	Removes all elements.	Retains all elements.
Deactivating an item which is an element	In a list, removes item. In an array, renders the item inaccessible until active.	Retains the item.

This action...	In a list or array with transient membership...	In a list or array with permanent membership...
Deleting an item	In a list, removes item. In an array, removes item so that element has no value.	In a list, removes item. In an array, removes item so that element has no value.
Activating list or array	Removes all elements. Populates array with initial values.	Populates list with previous elements. Populates array with previous elements, or with initial values if not previously populated.

When workspace items are list or array elements, their behavior differs from other elements. In G2, deactivated workspaces are the only deactivated items that you can refer to within expressions. Similarly, deactivated workspaces can exist as list or array elements.

Summary of List and Array Differences

The following table summarizes the differences between lists and arrays:

Characteristic	Arrays	Lists
Increases or decreases size dynamically		✓
Has a set length	✓	
Allocates memory at creation time	✓	
Contains mixed data types	✓	✓
Contains elements of values and items	✓	✓
Can have initial values	✓	

Creating Lists and Arrays

To create a list or array class:

1 Select KB Workspace > New Object > g2-list or g2-array.

2 Select a class of list or array from the choose a class submenu.

For example:

KB Workspace > New Object > g2-list > item-list

Setting the Array Length

The array-length attribute lets you specify how many elements the array contains. The default value is 0.

The array length should correspond to the number of initial values that you provide. If the number of initial values differs from the length of the array, G2 populates the array with the initial value for the array type.

Defining the Element Type

The element-type attribute defines the class or type of the list or array. You cannot change the value of this attribute, unless you modify the attribute-initializations attribute of user-defined class of item-list or item-array.

The valid types or classes for lists and arrays are listed in The List and Array Classes.

Allowing Duplicate List Elements

The allow-duplicate-elements? list attribute determines whether a list permits duplicate elements.

Changing the default yes value to no disallows duplicate list elements in the list.

If you allow duplicate elements in a list, and then change the attribute, G2 does not indicate whether duplicate elements already exist, but prevents you from subsequently adding any duplicates.

Note Inserting elements into a list that does not allow duplicate elements takes longer than into a list that allows duplicates.

For information on the backward compatibility feature for inserting duplicate elements into lists, see Ignoring Duplicate List Element Error.

Providing Initial Values for Array Elements

The initial-values attribute specifies an initial value for each array element. The number of initial values should correspond with the array length or should contain one initial value.

You can use this attribute to populate an array as Populating an Array describes. For g2-arrays and item-arrays, you can specify item names as the initial values.

Elements in arrays must always have a value, except for item-arrays and g2-arrays, which can have an initial value of none. If you do not specify initial values for the array, G2 provides initial values as follows.

Array Type	Default
g2-array	none
value-array	0.0
item-array	none
symbol-array	g2
text-array	""
truth-value-array	false
quantity-array	0.0
float-array	0.0
integer-array	0
long-array	0L

G2-Array Initial Values Conflict

You can set the initial values of a g2-array subclass in the attribute-initializations attribute of its class definition, as follows:

- If the initial-values attribute specifies only one value, all elements receive that value.
- If either the number of elements or the element type of an instance differs from the values specified in the class definition, G2 posts a warning about the discrepancy in the notes attribute of the instance.
- If the initial-values attribute has a count that does not match the array-length attribute, G2 assigns the initial value for that array class to each element.
- If any of the initial values of an item-array conflict with the element-type of the array, G2 assigns no-item to the element in conflict.

Specifying Symbolic Initial Values in Arrays

Three types of arrays can contain symbolic values:

- g2-array
- value-array
- symbol-array

When you enter symbolic values in the attribute-initializations attribute of a class definition, the syntax differs slightly depending on the array class.

For g2-array and value-array classes, precede symbolic names with the symbol. For symbol-arrays, enter only the symbolic name, as follows:

For this array class...	Enter symbolic values like this...
g2-array and value-array	the symbol cold, the symbol warm, the symbol hot
symbol-array	cold, warm, hot

Using Permanent-Membership Lists and Arrays

If the list-is-permanent attribute of a list, or the array-is-permanent attribute of an array is no (the default), membership in the list or array is transient:

- Resetting G2 causes a transient-membership list to be cleared, and a transient-membership array to be set to zero length.
- Starting G2 causes a transient-membership array to be set to its initial values.
- Membership information for a transient-membership list or array is lost during a KB save and load operation.

You can make list and array element knowledge permanent by changing the value of the list-is-permanent or array-is-permanent attribute to yes. A permanent-membership list or array is unaffected by resetting or restarting G2, and by saving and loading the KB, provided that all member items have the properties described in Complying to Permanent Membership.

Levels of Permanency in Lists and Arrays

The list-is-permanent and array-is-permanent attributes *do not* determine whether the list or array item is itself permanent, and *do not* determine whether any items referenced as elements are permanent. The attributes control only whether *membership* is permanent.

- If a list or array itself is transient, resetting G2 or saving the KB will delete it whether or not it specifies permanent membership.
- If an item referenced in a list or array is transient, resetting G2 or saving the KB will delete the item whether or not the list or array specifies permanent membership.

Thus, three levels of permanency apply to lists and arrays:

- Whether the list or array is itself permanent.
- Whether membership in the list or array is permanent.

- Whether an item that belongs to a list or array is permanent.

Be careful not to confuse these levels of permanency when you work with lists and arrays.

To ensure all levels of permanency, you must:

- Set the list-is-permanent or array-is-permanent attributes to true.
- Use the make permanent action:
  - After creating the list or array.
  - Each time you use the change action to add or remove elements to or from the list or array; you do not need to use the make permanent action if you add or remove elements, using the conclude action.
  - For each element of the list or array.

Initial Values of Arrays

Unlike lists, arrays can have initial values. If you do not provide initial values for an array, G2 populates each element with its initial value, which is type dependent. For example, for an integer-array, G2 supplies an initial value of 0; for a symbol array, G2 supplies the symbol g2.

When a permanent-membership array is first activated, G2 initializes it with either its default or user-provided initial values. The array then maintains initial or changed values throughout KB Restart and Reset operations, no further initialization takes place, and the values are saved in the KB.

Complying to Permanent Membership

For an item to remain in a permanent-membership list or array when G2 is reset, the item must be permanent. Otherwise, resetting G2 will delete it, as with any transient item, precluding its continued membership in the list or array.

Failure of an item to comply to permanent membership does not preclude its inclusion in a permanent-membership list or array. G2 does not signal an error if you populate permanent-membership list and array elements with non-compliant items, and does not post any messages if their non-compliance causes G2 to remove them from the list or array.

Maintaining Permanent-Membership Lists and Arrays

It is the KB developer's responsibility to monitor and maintain the integrity of permanent-membership lists and arrays. After adding items to them, G2 does not monitor members for conformance to permanent membership, nor does it inform you if it is unable to save any list or array item as a permanent member during a KB save or reset operation.

For items participating in permanent-membership lists and arrays, G2 does not signal an error if you:

- Conclude or change a list element to a transient item.
- Change the value of the list-is-permanent or array-is-permanent attribute to no.

Restoring Permanent-Membership Lists and Arrays

After successfully saving permanent-membership lists and arrays that contain items, G2 attempts to restore each item to membership when you load the KB. The inability to restore a previously saved list or array item member is called a **rendezvous failure**. Such a failures occurs if G2 is unable to locate one or more of the item members at KB load time.

If the UUID reference in a list or array does not correspond to any item because the item has been deleted or its UUID has changed, G2 replaces the item reference with no value.

Note Single-module KB saving is not compromised by inter-module item references in permanent lists and arrays.

Populating a List

Use the insert action to populate an empty list. The insert action causes G2 to insert a list element. The syntax is:

insert *item-or-value* {at the {beginning | end} of} |
{ {before | after} *item-or-value* in} *g2-list*

When you specify an insert action, by using the before or after phrase:

- G2 evaluates the existing element as a value, rather than as a specific location in the list.
- The element before or after the element you are inserting must already exist.

Several ways to insert list elements are:

```
insert 100 at the beginning of sample-list

insert 99 before 100 in sample-list

insert 101 after sample-list [1] in sample-list

insert 101 after the second integer in sample-list in sample-list
```

Notice that when you specify an insert *item-or-value* after the second *type* action, the list is noted twice: once to specify the list element and again to specify the list.

Note If the allow-duplicate-elements? attribute of any list is set to no and you attempt to insert a duplicate element, G2 signals an error.

Inserting Based on Element Location

To insert a list element based on element location:

➔ insert *item-or-value* {before | after} element *integer-expression* of *g2-list*

For example:

```
insert start-action-button after element 9 of button-list
```

Inserting at the Beginning or End of a List

To insert an element at the beginning or end of a list:

➔ insert *item-or-value* at the {beginning | end} of *g2-list*

This action inserts *item-or-value* at the beginning or end of the specified g2-*list*. You must use the insert action to add elements to an empty list.

For example:

```
insert 100 at the beginning of sample-list
```

Inserting Before or After an Existing Element

To insert an element before or after an existing list element:

➔ insert *item-or-value* {before | after} *item-or-value* in *g2-list*

This action inserts *item-or-value* either before or after the existing element stated by *item-or-value* of a list.

For example:

```
insert 99 before 100 in sample-list
```

```
insert temp-variable at the beginning of items-on-ws-list and insert s1
    after temp-variable in items-on-ws-list
```

Note When you use the location before or after, the element before or after the element you are inserting must already exist; otherwise, G2 signals an error.

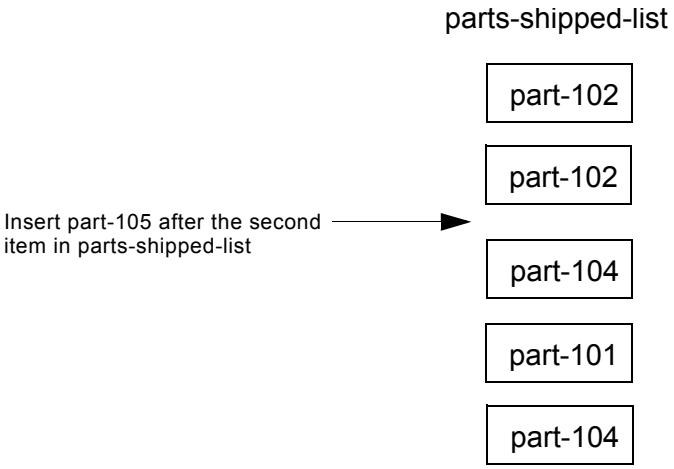
Inserting into Lists with Duplicate Elements

When a list allows duplicate elements, using the insert and remove actions may produce unexpected results if you use either the second or next to last phrase, both of which evaluate to a value rather than a location.

Consider the next example, in which an item list with duplicate elements contains several part numbers, and this expression indicates an insertion after the second element in the list:

```
insert part-no-105 after the second item in parts-shipped-list
in parts-shipped-list
```

The pointer in this diagram shows the desired position in the list:



When G2 evaluates this expression, it first obtains the *value* of the second item in the list, part-no-102 (this could also be an element index such as parts-shipped-list [1], for this example). G2 then searches the list from the first element to the last, and inserts the new element after the first *occurrence* of the given value. In this example, G2 inserts part-no-105 after the *first* element in the list (the first part-no-102) since it is the first *occurrence* of the second element value.

G2 guarantees to insert list elements after the first occurrence of an element value. Setting the allow-duplicate-elements? attribute to no prevents positional ambiguity when inserting new list elements.

Removing List Elements

To remove an existing element from a list, use the remove action:

➔ remove {*item-or-value* | element *integer-expression* |
{the {first | last} *type*} from *g2-list*

Two examples are:

```
remove the last float from float-list

remove the first quantity from q-list
```

The remove action removes an item, value, or list type from a list. List type is one of the following valid g2-list types: quantity, integer, float, symbol, text, or truth-value.

Removing a Particular List Element

To remove a particular list element:

➔ remove element *integer-expression* from *g2-list*

For example:

```
remove element 200 from new-autos
```

Removing Using an Element Index

To remove a list element:

➔ remove *item-or-value* from *g2-list*

The *item-or-value* must be present in the list. In this example, the first element with this *item-or-value*, starting from the beginning of the list, is removed.

```
remove temp-variable from items-on-ws-list
```

Removing a Type of List Element

To remove a type of list element:

➔ remove the {first | last} *type* from *g2-list*

The *type* can be one of the following valid data types for g2-lists: quantity, integer, float, symbol, text, truth-value, item-or-value, value, item, or a class.

Here are two examples:

```
remove the first integer from int-list

remove the last integer from int-list
```

If you attempt to remove a non-existent element or remove an element from an empty list, G2 signals an error.

Populating an Array

The two ways of populating an array are:

- Changing the initial-values attribute.
- Iterating over the array to change its elements.

Changing the Initial Values of an Array

You can change the initial-values attribute, interactively or programmatically, to populate the contents of each array element. Each time the array becomes active, G2 populates it with the initial values in the attribute table.

- If you specify only one initial value, G2 assigns that value to every array element.
- If you specify multiple initial values, but a total number of values less than or greater than the array length, all of the elements receive the default initial value for that array class and a warning appears in the notes attribute.

The following procedure code is an example of populating a newly created array, using the attribute access facility to conclude directly into the initial-values attribute:

```
create an integer-array NewArray;
conclude that the array-length of NewArray to 3;
conclude that the initial-values of NewArray = sequence (100, 101, 102);
transfer NewArray to WS at (50, 50);
make NewArray permanent;
conclude that the array-is-permanent of NewArray = true;
```

Note these things about the code:

- Because the value of the initial-values attribute is none by default, and does not consist of a sequence, you must conclude an entire sequence of elements into the attribute.
- The array-is-permanent attribute values are yes and no on the attribute table. Since the type of this attribute is truth-value, however, you conclude its value as true or false programmatically.

Iterating over an Array

You can populate an array by iterating over its elements and inserting values or items. When you do this, be careful that concurrent processes are not allowed to affect the results of the iteration. For details see Allowing Other Processing During List and Array Iteration.

The next example shows how to populate an array with the elements of a list, first changing the array length to the number of elements in the list, and then iterating through the list to insert element values into the array:

```
conclude that the array-length of int-array = the number of elements in int-list;
index = 0;
for int-value = each integer in int-list
    do
        conclude that int-array [index] = int-value;
        index = index + 1;
    end
```

Using an Attribute File

Attribute files are a superseded capability. For further information, see Appendix F, Superseded Practices.

Replacing List and Array Elements

Use the change and conclude actions to replace items and values.

Using Change

To change an element of an array or list:

➔ change {*g2-array* | *g2-list*} [*integer-expression*] = *item-or-value*

This action places an item or value into an array or list, replacing the existing item or value, if any. Lists must already have an item or value at the specified location, which is replaced by the change action. Arrays do not require an existing value at the specified location, but if one exists, it is replaced.

An example is:

```
change list-of-names[10] = the name of new-patient-10
```

Using Conclude

To replace a list or array element with a value:

➔ conclude that {*g2-array* | *g2-list*} [*integer-expression*]
{= *value-expression* | is *symbolic-expression*}

If you try to conclude a value into an element that does not contain a value, a variable, or a parameter, G2 signals an error. To conclude a list element, the list must already have an item or value at that location.

The behavior of this action varies depending on whether the current element is an item or a value, as follows:

If the element at the given position...	Then the conclude action...
Is a value	Replaces the existing value with the new one. This has the same behavior as the change action described under change.
Is a variable or parameter item	Concludes a new value for the variable or parameter.
Is an item that is not a variable or parameter	Causes G2 to signal an error.

You cannot conclude a new item into lists and arrays with item elements. Use the change action to place a new item into an existing element of a list or array of items.

Two examples are:

```
change parts-list[4] = 191.1
```

```
conclude that auto-array[100] = new-auto-10
```

If the array or list contains values, no difference exists between using the change or conclude actions to replace an element.

Altering the Length of an Array

To change the length of an array:

➔ conclude that the array-length of *g2-array* = *quantity-expression*

The *g2-array* refers to any array. If the quantity expression you enter makes the array longer, this action also re-initializes the new elements to the values of the default initial-values attribute. For more information about this attribute, see Providing Initial Values for Array Elements.

For example:

```
conclude that the array-length of my-array = 20
```

Changing Elements to Have No Values

Use the change action with the syntax described below to designate that an element of an item-list or an item-array or a g2-list or a g2-array does not have a value. Value arrays always have an initial value, so you cannot change or conclude a value element not to have a value.

To change an array or list element to have no value, use this syntax:

➔ change {*g2-array* | *g2-list*} [*integer-expression*] to have no value

For example:

```
change project-list[100] to have no value
```

Data Seeking and Event Updating

Changing lists and arrays has this effect on data seeking and event updating within your KB:

- Adding or removing list or array elements does not cause event updating.
- G2 does not data seek to add elements to an array or list.
- G2 does not forward chain to array element expressions.
- G2 does forward chain to some list element expressions, when the element is an item. These expressions are first, second, next to last, last, and type in g2-list.

Iterating over Lists and Arrays

You can iterate over lists and arrays to perform various computations, including populating the list or array, adding elements, reasoning about elements, and so forth.

To iterate over the elements in a list:

➔ for *local-name* = each *type* in *list* do

For example:

```
for int-value = each integer in my-list
    do
        change int-array[index] = int value;
        index = index + 1
    end
```

This is an efficient method for list iteration because G2 locates each list element by simply incrementing a pointer at the beginning of each iteration of the loop.

Using counter syntax such as for i = 0 to 15 do change my-array[i] = my-list[i] requires unnecessary processing because it causes G2 to locate each list element by starting at the beginning of the list for each loop iteration.

Here is a procedure that iterates over an item list of task times, using the insert action:

```
enqueue-task (new-task: class task, task-queue: class item-list)
task-1: class task;
task-time-of-new-task: float = the start-time of new-task;

begin
    for task-1 = each task in task-queue
        do
            {If task is time dependent, insert at beginning}
            if the start-time of task-1 > task-time-of-new-task then
                begin
                    insert new-task before task-1 in task-queue;
                    return;
                end;
        end;
            {If not time dependent, add to end of queue.}
            insert new-task at the end of task-queue;
end
```

In the example, the enqueue-task procedure adds new tasks to a queue in the correct time sequence.

Iterating According to Element Type

G2 can iterate over the current elements of a list or array to reference a specific element type.

To iterate according to element type:

➔ the {*class-name* | *type*} [*local-name*] in {*g2-array* | *g2-list*}
-> {*item* | *integer* | *float* | *symbol* | *text* | *truth-value*}

➔ the *class-name* [*local-name*] in {*item-array* | *item-list*}
-> *item*

➔ the *type* [*local-name*] in {*value-array* | *value-list*}
-> {*integer* | *float* | *symbol* | *text* | *truth-value*}

These generic reference expressions produce the item or items of the specified class, or the value or values of the specified type, that is referenced in an item-array, item-list, value-array, or value-list.

For example:

```
if any float in task-list < the current time then post "Task is late."
```

Tip Referring to an item that is a member of a list or array, is an *indirect* reference to that item.

Iterating over Lists For a Particular Item

To iterate over every list that contains a particular item:

➔ *g2-list* that contains *item*

For example:

```
for x = each item-list that contains my-object do ...
```

Specifying a Relative List Position

To iterate over each list element specifying a relative position in the list:

➔ the {first | second | next to last | last} {*class-name* | *type*} [*local-name*] in *g2-list*
-> {*item* | *integer* | *float* | *symbol* | *text* | *truth-value*}

These expressions produce the item or value that is referenced in the specified element in a list. An example:

```
the first float in grades-list
```

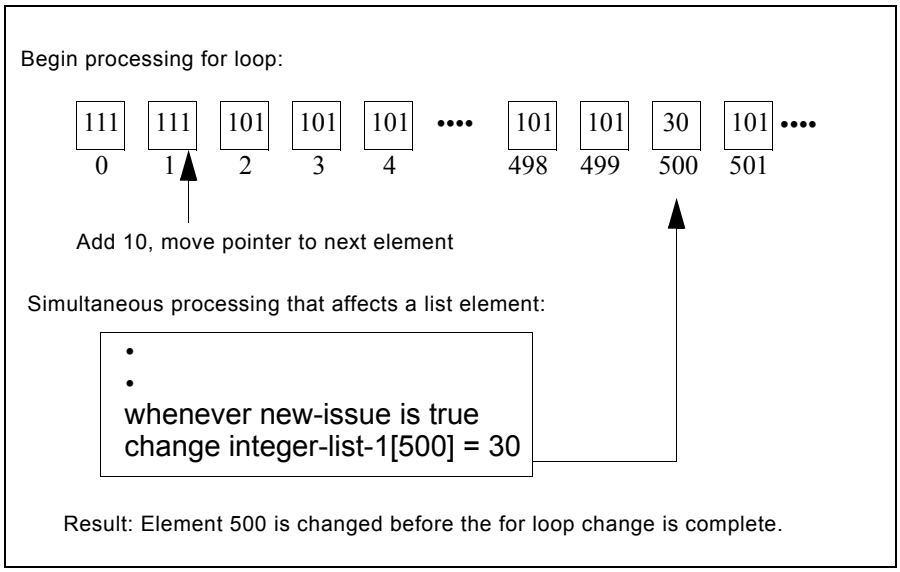
Allowing Other Processing During List and Array Iteration

Allowing other processing to occur during list or array iteration can cause unpredictable results. For example, consider the following loop code that allows other processing while iterating over a list, adding 10 to each integer in integer-list-1, which consists of 1000 elements:

```
for int-value = each integer in integer-list-1
    do
        conclude that integer-list-1[i] = int-value + 10;
        i = i + 1;
        allow other processing;
    end
```

When a procedure allows other processing, it is possible for another statement to change the list element values during the processing of the procedure. The following diagram illustrates how an element could be changed twice during the

procedure, once from an external action (a whenever rule), and once from the procedure.



Simultaneous processing could also result in G2 adding an element to the beginning of the list while the procedure was processing the remainder of the list. In such a situation, the for loop would not update the new value.

Due to the possibility of concurrency problems, we recommend that you do not allow other processing to occur during list or array iteration unless you are certain that no concurrent processing can interfere. For more information, see Allowing Other Processing.

Using Other List and Array Expressions

You can reason about both lists and arrays and their elements by using the expressions described here. For a general discussion of G2 expressions, see Expressions.

Accessing List or Array Elements by Index

To access a list or array element with an index:

➔ {*g2-array* | *g2-list*} [*integer*]
-> {*item* | *integer* | *float* | *symbol* | *text* | *truth-value*}

This expression produces the value in, or the item referenced in, the specified element in a list or array.

An example is:

```
conclude that my-array[9] = the current time
```

When an expression refers to an element in a list, the length of time G2 requires to evaluate the expression is proportional to the position of the specified element in the list.

Performing Computations over Sets of Elements

You can compute the values of expressions over sets of elements in a list or an array.

Sum, Product, Minimum, Maximum Of

the {sum | product | minimum | maximum} over each
generic-reference-expression in {*g2-array* | *g2-list*} of (*quantity-expression*)
-> {*integer* | *float*}

This expression produces a calculated value of either type integer or float from the set of items or values specified in the generic reference expression, which are contained in the specified array or list and which meet the criterion specified in the quantity expression.

Because this expression can produce either an integer or float value, use a piece of knowledge declared as type quantity to contain the produced value.

For example, the following expression computes the sum of the flows of all valves that are connected to tank-1 and are also elements in my-valve-list:

```
the sum over each valve V connected to tank-1 in my-valve-list of
  (the flow of V)
```

Average Of

the average over each *generic-reference-expression* in {*g2-array* | *g2-list*}
of (*quantity-expression*)
-> *float*

This expression produces a calculated value of type float from the set of items or values specified in the generic reference expression, which are contained in the specified array or list and which meet the criterion specified in the quantity expression.

An example is:

```
x = the average over each task T in task-list of (the start-time of T);
post "[x] is the average of the start times of all jobs in the queue";
```

Count Of

You can also use the count of each expression to specify the elements that G2 iterates over in a set.

the count of each *generic-reference-expression* in {*g2-array* | *g2-list*}
[such that (*truth-value-expression*)]
-> *integer*

This expression produces the number of items or values specified in the generic reference expression, which are contained in the specified array or list and which meet the criterion specified in the truth-value expression. For a list, unless you are counting a subset of the list's elements, using the the number of elements in expression is faster.

For example, the following expression finds the number of elements in qlist-1 that have the value 4:

```
the count of each water-tank T in water-tank-list such that T < 200
```

Note You can optimize the execution of a the count of each expression if it references indexed attributes. See Defining an Indexed Attribute for more information about indexed attributes.

Testing for List Membership

To test whether an element is or is not a member of a list:

➔ {*class-name* | *type*} is [not] a member of *g2-list*
-> *truth-value*

When testing for membership in a specified list, G2 ignores the alphabetic case when comparing two text values and ignores the type when comparing two quantity values. For example:

- The text string "Text" is a member of the text-list that contains "text".
- The float 2.0 is a member of the quantity-list that contains the integer 2.

This expression produces a truth-value that indicates whether an item of the specified class, or a value of the specified type, is a member of the specified list.

Referring to an item that is a member of a list or array, is an *indirect* reference to that item.

An example is:

```
when task-1 is a member of task-list post "task-1 is included"
```

Obtaining the Number of List Elements

To find the number of elements in a list:

➔ the number of elements in *g2-list*
-> *integer*

This expression produces the number of elements in a list. You can also use the the count of each expression to count the elements in a list; however, using the the number of elements in expression operates more quickly.

An example is:

```
conclude that element-count = the number of elements in task-list
```

Finding the Length of an Array

To find the length of an array:

➔ the array-length of *g2-array*
-> *integer*

This expression produces the number of elements in an array.

Accessing Lists or Arrays That are Object Attributes

User-defined objects can have attributes that are instances of lists and arrays. You specify such an attribute in a class definition.

To access lists or arrays that are object attributes:

➔ *attribute-name* initially is an instance of {a | an} {*g2-list* | *g2-array*}

where *attribute-name* is the name of the attribute you are specifying, and *g2-list* or *g2-array* is any applicable g2-list or g2-array class or subclass. To distinguish lists and arrays that are object attributes, this section refers to them as *attribute lists and arrays*.

When objects with such attributes are instantiated, the lists or arrays exist as separate items within the KB. You can access a list or array that is an attribute by naming it. You can name attribute lists and arrays interactively by accessing the subtable from the object's attribute table.

To change the name of attribute lists and arrays programmatically:

➔ change the text of the names of the *array-attribute* of *object* to *text*

or

➔ change the name of the *array-attribute* of *object* to the symbol *symbol*

You can also use attribute access to reference and change most attribute values, as described in Attribute Access Facility.

Changing Attribute List and Array Elements

To provide new element values to attribute lists and arrays without names:

➔ change (the *attribute* of *object*) [*integer-expression*] = *item-or-value*

where *attribute* is the attribute name, *object* is the object, followed by the element index, and *item-or-value* is the new value to assign to that array or list element.

The conclude actions uses two similar syntactical forms to produce different results.

Concluding an Unnamed Object Attribute That is a List or an Array

To provide a new element value to an unnamed attribute list or array:

➔ conclude that (the *attribute-name* of *object*) [*integer-expression*] = *item-or-value*

Element Syntax	Description
(the *attribute-name* of *object*)	An unnamed attribute list or array whose element value you want to change.
item-or-value	The new value for the attribute list or array element.

Concluding a List or Array Element That is an Object

To change the attribute value of an object that is a list or array element:

➔ conclude that the *attribute* of {*g2-list* | *g2-array*} [*integer-expression*] = *item-or-value*

Element Syntax	Description
g2-list | *g2-array* [*integer-expression*]	An array or list whose element at the specific index consists of an object, which includes an attribute as specified by *attribute*.
attribute	The attribute name of the object referenced in the array or list.
item-or-value	The new value of *attribute*, which must be a value, a variable, or a parameter. If the array or list element is a variable or a parameter, G2 updates it with the new value.

Copying Lists and Arrays

You can copy a list to a sequence or a sequence to a list by using g2-list-sequence from the table of hidden attributes for the list.

Similarly, you can copy an array to a sequence or a sequence to an array by using g2-array-sequence from the table of hidden attributes for the array.

These attributes are accessible to you programmatically through G2's attribute access facility and are documented in the *G2 Class Reference Manual*.

Procedures for copying arrays to sequences and sequences to arrays are similar. When copying a sequence into an array the values from the sequence take precedence over any initial values the array may already have, and the array length of the array adjusts, as needed.

g2-list-sequence

Here is a procedure for copying a list into a sequence:

```
copy-list-to-seq(list: class g2-list) = (sequence)
seq: sequence;
begin
    seq = the g2-list-sequence of list;
    return seq;
end
```

Here is a procedure for copying a sequence into a list:

```
copy-seq-to-list(seq: sequence, list: class g2-list)
begin
    conclude that the g2-list-sequence of list = seq;
end
```

g2-array-sequence

Here is a procedure for copying an array into a sequence:

```
copy-array-to-seq(array: class g2-array) = (sequence)
seq: sequence;
begin
    seq = the g2-array-sequence of array;
    return seq;
end
```

Here is a procedure for copying a sequence into an array:

```
copy-seq-to-array(seq: sequence, array: class g2-array)
begin
    conclude that the g2-array-sequence of array = seq;
end
```

You can use the g2-array-sequence hidden attribute to copy the initial-values of an array to the current values of the array, as follows:

```
initialize-array ()
array: class g2-array;
begin
    conclude that the g2-array-sequence of array = the initial-values of array;
end
```

Representing Sparse Arrays

A dense array is a quantity array that can include zero values. A sparse- array representation consists of two separate arrays: a value array that holds only the non-zero values of a corresponding dense array and an integer array which holds the indexes to the non-zero elements as they would be in the corresponding dense array.

A sparse array can be more efficient because it reduces the number of elements that need processing; however, it requires the overhead of an additional array to hold the indexes.

For example, a dense array can contain the following values:

(0.0, 0.0, 0.0, 45.1, 9.2, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 35.6)

The sparse-array representation that corresponds to this dense array consists of two arrays: a value array with these values:

(45.1, 9.2, 35.6)

and an index array with these values:

(3, 4, 12)

where 3, 4, and 12 are the indexes to the three non-zero values of the corresponding dense array.

The G2 system procedure, g2-sparse-gather, converts a dense array into a sparse array; and the G2 system procedure, g2-sparse-scatter, converts a sparse array into a dense array.

See the *G2 System Procedures Reference Manual* for a complete description of these and other procedures.

Representing Matrixes with Arrays

A matrix can be represented by an item-array whose elements are also arrays. For example, you can construct a 3 x 4 matrix of integer values by defining a three-element item array and populating each element with a 4-element integer-array. Each integer array represents a row in the matrix.

To represent a matrix as an array, you must first create the individual arrays before inserting them into the slots of the top-level array.

Using System Procedures with Lists, Arrays, and Matrixes

Here is a list of system procedures specifically for use with lists, dense arrays, dense matrixes, and sparse arrays. For a complete description of these and other G2 system procedures, see the *G2 System Procedures Reference Manual*.

To do this...	Use this system procedure...
Sort a list	g2-sort-list
Sort an array	g2-sort-array
Get the element position of an item or value in an array	g2-get-position-of-element-in-array
Get the element position of an item or value in a list	g2-get-position-of-element-in-list
Copy array elements to the initial-values attribute of the array	g2-array-copy-elements-to-initial-values
Get the largest value in an array	g2-array-max
Get the smallest value in an array	g2-array-min
Get the sum of all the elements in an array	g2-array-sum
Get the sum of the absolute values of all the elements in an array	g2-array-sum-abs
Add two arrays	g2-array-add
Copy one array into another array	g2-array-copy

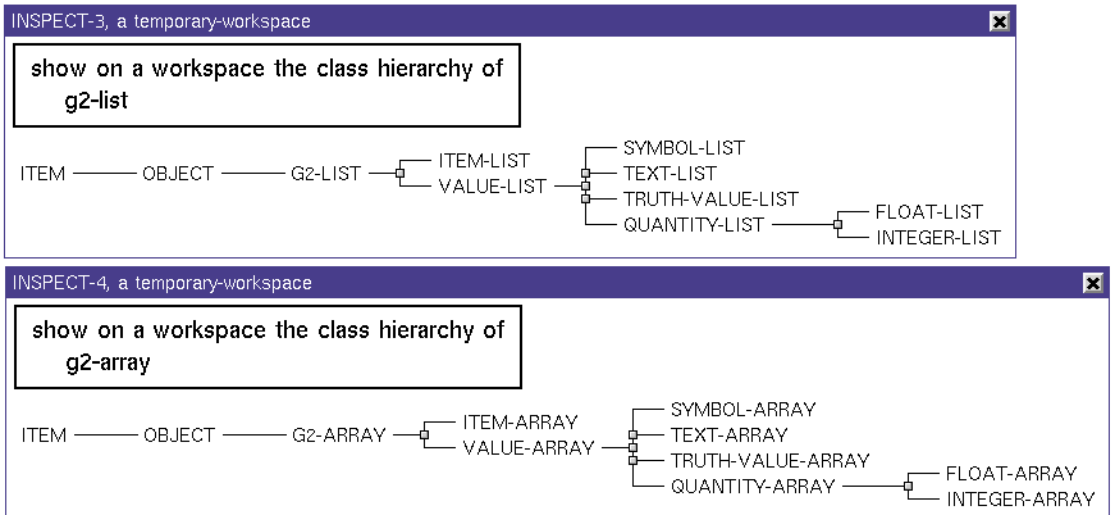
To do this...	Use this system procedure...
Determine if two arrays have the same element values	g2-array-equal
Multiply two arrays	g2-array-multiply
Subtract one array from another	g2-array-subtract
Get the dimensions of a matrix	g2-get-matrix-dimensions
Solve the set of N linear equations AX=B	g2-lu-back-substitute
Decompose a matrix using Crout's method with partial pivoting	g2-lu-decompose
Get the values that solve the equation: x-array = a-matrix * x-array = b-array	g2-lu-solve
Multiply a matrix by another matrix or an array	g2-matrix-multiply
Multiply an array by a scalar	g2-scalar-multiply
Transpose the rows and columns of a matrix	g2-transpose
Get the result of the operation X = X + alpha * Y, where X and Y are sparse arrays	g2-sparse-add
Convert a dense array into a sparse array	g2-sparse-gather
Get the value of an element in a sparse array	g2-sparse-get
Get the product of multiplying two sparse arrays	g2-sparse-multiply
Convert a sparse array into a dense array	g2-sparse-scatter
Set a sparse array element to a value	g2-sparse-set

The List and Array Classes

Within the system-defined class hierarchy, nine system-defined parallel subclasses for lists and arrays exist, each capable of containing these elements:

These list and array classes...	Can have these elements...
g2-list or g2-array	Items and values
value-list or value-array	Values of any G2 type, but no items
item-list or item-array	Items
symbol-list or symbol-array	Symbolic values
text-list or text-array	Text values
truth-value-list or truth-value-array	Truth values
quantity-list or quantity-array	Float or integer values
float-list or float-array	Float values
integer-list or integer-array	Integer values

The class hierarchies of lists and arrays are:



Creating Subclasses of Lists and Arrays

You can create subclasses directly from g2-list and g2-array, and from any of their subclasses as you would any other user-defined class.

When creating a new list or array class, always use the most specific type possible for your requirements, for example:

- If a list class is to contain only float elements, use float-list as the superior class, not quantity-list.
- If an array should contain items, specify item-array as the superior class rather than g2-array.

You can also create your own set of list or array classes in which membership is permanent by default, by specifying the array-is-permanent or list-is-permanent as yes in the attribute-initializations attribute of the class definition.

For classes with value-array as a direct superior class, you can include values of any type (float, integer, symbol, text, and truth-value, but not items), as the following attribute-initializations value indicates:

```
array-length for g2-array: 3;
initial values for value-array: the symbol ME, the symbol YOU, the symbol US
```

Notice that you must precede symbolic values (me, you and us in this example) with the statement the symbol, as described in Specifying Symbolic Initial Values in Arrays.

For new classes with item-list or item-array as a direct superior class, the elements of the list or array can be generic or specific to a particular item class. To create a list or array of a specific item class, use an class definition to create a subclass of item-list or item-array.

Within the class definition, you can use the attribute-initializations attribute to specify a particular class of items for the element-type attribute. When you specify a class name for the element-type attribute, all elements within the list or array must be instances of that class.

For a complete description of creating subclasses, see Creating Object Classes.

Class-Specific Attributes

These are the class-specific attributes of lists and arrays:

Attribute	Description
array-length	Arrays only. Specifies the number of elements that the array contains.
Allowable values:	Minimum: 0 Maximum: 523,264
Default value:	0
element-type	Specifies the type or class of the list or array elements.
Allowable values:	For a list of allowable type or classes for lists and arrays, see The List and Array Classes.
Default value:	For a list of type or classes for lists and arrays, see The List and Array Classes.
allow-duplicate-elements?	Lists only. Controls whether duplicates of the same element can exist in a list.
Allowable values:	{yes | no}
Default value:	yes
initial-values	Arrays only. Specifies the initial-value for each array element.
Allowable values:	See Providing Initial Values for Array Elements
Default value:	See Providing Initial Values for Array Elements
array-is-permanent	Arrays only. Specifies whether the elements of the array are maintained as permanent KB knowledge.
Allowable values:	yes or no

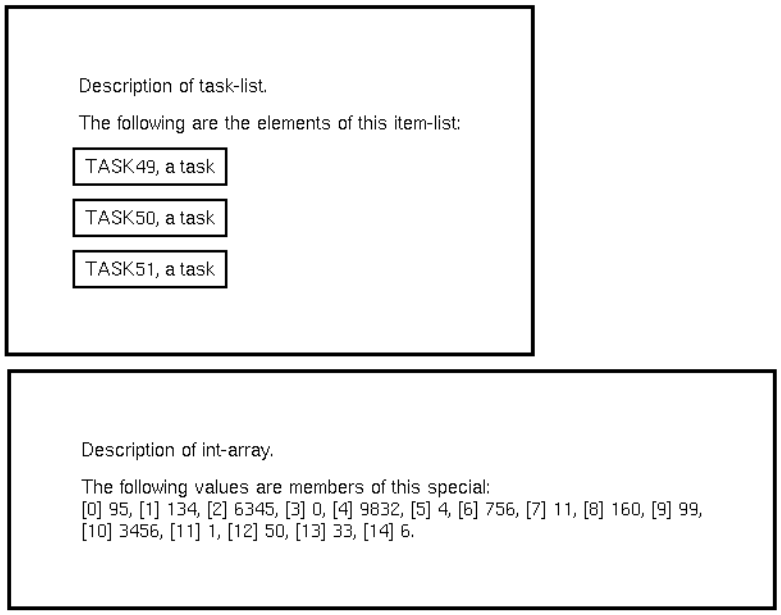
Attribute	Description
Default value:	no
list-is-permanent	Lists only. Specifies whether the elements of the lists are maintained as permanent KB knowledge.
Allowable values:	yes or no
Default value	no

Note G2 permits the use of lists and arrays of the non-specific type quantity. For efficiency, we recommend that you use an item of specific type (integer or float) whenever possible.

Describing Lists and Arrays

Once a list or array is populated, you can use the describe menu choice to display its elements, or to display the lists and arrays to which an item belongs. The Describe facility includes a list or array index for every value-array element.

The next diagram shows an example of the Describe facility for an array and a list. The description of the list contains short references to the tasks that are its elements.



When you select describe from the menu of a list or an array, G2 displays a workspace containing information about every datum or item in the list or array. Click on the short reference of the element item to access the attribute table of the original, go to the original item, and so on.

The Describe display does not update dynamically as list or array elements change.

Note We do not recommend using the Describe facility for very long lists or arrays with many elements, because it uses a large amount of storage space.

17

Hash Tables and Priority Queues

Describes how to use hash tables and priority queues.

Introduction

G2 provides two data structures for use in a wide variety of programming contexts:

- hash-table — A collection of key-value pairs, where the key and the value can be any G2 item-or-value. Hash tables provide fast lookups for various types of data, regardless of the number of entries in the table, where the lookup time is proportional to the log of the number of key-value pairs in the table.
- priority-queue — A collection of items, each with an associated priority. For example, you could use a priority queue as the core of an event-based simulator, where the time an event should occur is used as the priority.

While sequences and structures also provide the ability to define key-value pairs, structures require that the keys be symbols. Hash tables do not have this restriction; the key and the value can be any G2 item or value. Also, when the number of key-value pairs in a sequence or structure becomes very large, finding elements can be very slow. Thus, when all you require is a set of key-value pairs, we recommend that you use hash tables rather than sequences or structures. If you require the ability to parse the key-value pairs sequentially, you should use a sequence or a list.

G2 provides hash tables and priority queues as objects, which you can create from the KB Workspace > New Object menu or programmatically, using the create action. G2 provides a number of procedures for adding and removing elements, accessing values, and dynamically changing values in a hash table, and for changing the priority of an item, and get the object with the highest priority from a priority queue.

Note G2 does not save the contents of a hash table or priority queue in a KB when it is saved. When a new KB is loaded, all hash tables and priority queues are emptied. Also, disabling a hash table or priority queue empties their contents.

Hash-Table Class

The hash-table class provides a data structure for fast lookup of a value, based on a key. The key and value can be any item-or-value. When specifying keys as text, the text is case sensitive.

G2 provides procedures for getting and setting values given a key, clearing individual values from the table given a key, clearing all key-value pairs from the table, and converting hash tables to sequences to allow iterating over the elements.

You can subclass the hash-table class to provide application-specific behavior.

Hidden Attributes

The hash-table class defines the following hidden attributes:

Attribute	Description
g2-hash-table-sequence	A sequence of structures that defines all the key-value pairs in the hash table, where each structure has attributes ENTRY-KEY and ENTRY-VALUE. For example: sequence (structure (ENTRY-KEY: "Key 1", ENTRY-VALUE: "text value"), structure (ENTRY-KEY: "Key 2", ENTRY-VALUE: "text value"))
Allowable values:	A sequence of structures
Default value:	sequence()
Notes:	See also the description of g2-hash-table-to-sequence in the *G2 System Procedures Reference Manual*.
g2-hash-table-number-of-entries	The number of structures in the g2-hash-table-sequence. In the sequence above, the number of entries is 2.
Allowable values:	integer
Default value:	0

Application Programmer's Interface

The API procedures for hash tables appear on the g2-hash-tables workspace of G2 System Procedures.

For a description of these procedures, see the *G2 System Procedures Reference Manual*.

To display the hash table procedures:

➔ Choose Get Workspace > g2-system-procedures to display the G2 System Procedures top-level workspace, display the table of contents, and choose g2-hash-tables.

Note The g2-hash-table-to-sequence procedure is not supported and will be removed in a future release. The procedure exists for compatibility with G2 Version 8.0 Beta Rev. 0 only. Use the hash-table-sequence hidden attribute on hash-table instances instead.

Example: Hash Tables

Here is an example of manipulating hash tables. First, the procedure clears the hash table named my-hash-table, then it calls g2-get-hash-table-value on a key, which doesn't exist. The return value is false. The procedure then sets various keys and values, using different data types, and gets the resulting values for each key and posts the result. It clears a value for a particular key and attempts to get its value. Then it posts the value of the g2-hash-table-sequence. Finally, it clears all keys and values and post the empty sequence. The procedure provides three examples of getting a value from a textual key, one of which fails because it does not use the correct case.

```
test-hash-table()
result: item-or-value;
found: truth-value;
seq: sequence;

begin
    {clear the hash table}
    call g2-clear-hash-table(my-hash-table);

    {symbolic key and value with no value set}
    result, found = call g2-get-hash-table-value(my-hash-table, the symbol my-key);
    post "symbolic key my-key has value [result]";

    {symbolic key and value}
    call g2-set-hash-table-value(my-hash-table,the symbol my-key, the symbol val-1);
    result, found = call g2-get-hash-table-value(my-hash-table, the symbol my-key);
    post "symbolic key my-key has value [result]";
```

```
    {text key and value}
    call g2-set-hash-table-value(my-hash-table, "text key 1", "text value");
    result, found = call g2-get-hash-table-value(my-hash-table, "text key 1");
    post "text key @"text key 1@" has value [result]";

    {upper case text key and value}
    call g2-set-hash-table-value(my-hash-table, "Text Key 2", "text value");
    result, found = call g2-get-hash-table-value(my-hash-table, "Text Key 2");
    post "text key @"Text Key 2@" has value [result]";

    {wrong case text key and value returns false}
    result, found = call g2-get-hash-table-value(my-hash-table, "text key 2");
    post "text key @"text key 2@" has value [result]";

    {quantity key and value}
    call g2-set-hash-table-value(my-hash-table, 1, 1.0);
    result, found = call g2-get-hash-table-value(my-hash-table, 1);
    post "quantity key 1 has value [result]";

    {logical key and value}
    call g2-set-hash-table-value(my-hash-table, true, true);
    result, found = call g2-get-hash-table-value(my-hash-table, true);
    post "logical key true has value [result]";

    {item key and value}
    call g2-set-hash-table-value(my-hash-table, cp-1, cp-2);
    result, found = call g2-get-hash-table-value(my-hash-table, cp-1);
    post "item key cp-1 has value [the name of result]";

    {clear hash table value}
    call g2-clear-hash-table-value(my-hash-table, cp-1);
    result, found = call g2-get-hash-table-value(my-hash-table, cp-1);
    if found then
        post "item key cp-1 has value [the name of result]"
    else post "The key cp-1 does not exist";

    {post hash table sequence}
    seq = the g2-hash-table-sequence of my-hash-table;
    post "the hash table sequence = [seq]";

    {clear hash table and post empty sequence}
    call g2-clear-hash-table(my-hash-table);
    seq = the g2-hash-table-sequence of my-hash-table;
    post "the hash table sequence = [seq]";
end
```

Here is the hash table and the resulting Message Board when you start this procedure:

MY-HASH-TABLE

MESSAGE-BOARD

MESSAGE-BOARD

#92 9:37:55 a.m. symbolic key my-key has value false

#93 9:37:55 a.m. symbolic key my-key has value VAL-1

#94 9:37:55 a.m. text key "text key 1" has value text value

#95 9:37:55 a.m. text key "Text Key 2" has value text value

#96 9:37:55 a.m. text key "text key 2" has value false

#97 9:37:55 a.m. quantity key 1 has value 1.0

#98 9:37:55 a.m. logical key true has value true

#99 9:37:55 a.m. item key cp-1 has value CP-2

#100 9:37:55 a.m. The key cp-1 does not exist

#101 9:37:55 a.m. the hash table sequence = sequence (structure (ENTRY-KEY: "text key 1", ENTRY-VALUE: "text value"), structure (ENTRY-KEY: "Text Key 2", ENTRY-VALUE: "text value"), structure (ENTRY-KEY: 1, ENTRY-VALUE: 1.0), structure (ENTRY-KEY: the symbol MY-KEY, ENTRY-VALUE: the symbol VAL-1), structure (ENTRY-KEY: true, ENTRY-VALUE: true))

#102 9:37:55 a.m. the hash table sequence = sequence ()

Priority-Queue Class

The priority-queue class provides a data structure for associating items with a priority, which can be any float value.

G2 provides procedures for adding a new item to a queue at a given priority, removing an item from a queue, changing the priority of an existing item in a queue, getting the item with the highest priority, getting and removing the item with the highest priority, and determining if the queue is empty.

Note that if items in the priority queue have the same priority, the order in which they are retrieved from the queue is unpredictable. If you care about the order of items with the same priority, then provide a more detailed prioritization scheme, such as 1.1, 1.2, 1.3, and so on.

You can subclass the priority-queue class to provide application-specific behavior.

Hidden Attributes

The priority-queue class defines the following hidden attributes:

Attribute	Description
g2-priority-queue-sequence	A sequence of structures that defines all the items and priorities in the priority queue, where each structure has attributes ENTRY and PRIORITY. For example: sequence (structure (ENTRY: post-three, PRIORITY: 3.0), structure (ENTRY: post-five, PRIORITY: 5.0))
Allowable values:	A sequence of structures
Default value:	sequence()

Attribute	Description
g2-priority-queue-number-of-entries	The number of structures in the g2-priority-queue-sequence. In the sequence above, the number of entries is 2.
Allowable values:	integer
Default value:	0

Application Programmer's Interface

The API procedures for priority queues appear on the g2-priority-queues workspace.

For a description of these procedures, see the *G2 System Procedures Reference Manual*.

To display the priority queue procedures:

➔ Choose Get Workspace > g2-system-procedures to display the G2 System Procedures top-level workspace, display the table of contents, and choose g2-priority-queues.

Example: Priority Queue

This example performs these operations on a priority queue:

- Clears the priority queue.
- Builds a new queue by inserting items randomly into the queue.
- Posts the name of the highest priority item to the Message Board.
- Removes the top item from the queue and posts the name of the new highest priority item.
- Changes the priority of the top item and posts the name of the new highest priority item.
- Removes a specific item from the queue.
- Posts the value of the g2-priority-queue-sequence.
- Loops through the rest of the items in the queue, removing the top item until the queue is empty, then posts the empty sequence. Note that looping through the items in a priority queue, based on the priority is a fast operation, regardless of the number of items in the queue.

Here is the procedure that performs these operations:

```
reorder-items(queue: class priority-queue)
empty, result: truth-value;
itm: item-or-value;
priority: float;
cp: class connection-post;
seq: sequence;
begin
    {clear queue}
    call g2-clear-priority-queue(queue);

    {build queue}
    result = call g2-insert-in-priority-queue(queue, post-one, 1.0);
    result = call g2-insert-in-priority-queue(queue, post-three, 3.0);
    result = call g2-insert-in-priority-queue(queue, post-five, 5.0);
    result = call g2-insert-in-priority-queue(queue, post-four, 4.0);
    result = call g2-insert-in-priority-queue(queue, post-six, 6.0);
    result = call g2-insert-in-priority-queue(queue, post-two, 2.0);

    {get highest priority item}
    itm, priority = call g2-get-higheset-from-priority-queue(queue);
    post "[the name of itm] is the top item in the queue at priority [priority]";

    {remove highest priority item}
    itm, priority = call g2-remove-highest-from-priority-queue(queue);
    post "[the name of itm] removed from the queue";
    itm, priority = call g2-get-highest-from-priority-queue(queue);
    post "[the name of itm] is the top item in the queue at priority [priority]";

    {change priority of item}
    result = call g2-change-priority-in-priority-queue(queue, itm, 7.0);
    post " [the name of itm] now has priority 7.0";
    itm, priority = call g2-get-highest-from-priority-queue(queue);
    post "[the name of itm] is the top item in the queue at priority [priority]";
```

```
    {remove post-four from queue}
    result = call g2-remove-from-priority-queue(queue, post-four);
    post "@"post-four@" removed from the queue";
    itm, priority = call g2-get-highest-from-priority-queue(queue);
    post "[the name of itm] is the top item in the queue at priority [priority]";

    {post priority queue sequence}
    seq = the g2-priority-queue-sequence of queue;
    post "the priority queue sequence = [seq]";

    {remove rest of items until queue is empty}
    repeat
        itm, priority = call g2-remove-highest-from-priority-queue(queue);
        post "[the name of itm] removed from the queue at priority [priority]";
        result = call g2-priority-queue-is-empty (queue);
        exit if result is true;
    end;
    post "the queue is empty";

    {post empty sequence}
    seq = the g2-priority-queue-sequence of queue;
    post "the priority queue sequence = [seq]";

end
```

Here is the priority queue and the resulting Message Board when you start this procedure:

MY-QUEUE

MESSAGE-BOARD

MESSAGE-BOARD

#117 9:45:17 a.m. POST-ONE is the top item in the queue at priority 1.0

#118 9:45:17 a.m. POST-ONE removed from the queue

#119 9:45:17 a.m. POST-TWO is the top item in the queue at priority 2.0

#120 9:45:17 a.m. POST-TWO now has priority 7.0

#121 9:45:17 a.m. POST-THREE is the top item in the queue at priority 3.0

#122 9:45:17 a.m. "post-four" removed from the queue

#123 9:45:17 a.m. POST-THREE is the top item in the queue at priority 3.0

#124 9:45:17 a.m. the priority queue
sequence = sequence (structure (ENTRY:
POST-THREE,
PRIORITY: 3.0),
structure (ENTRY: POST-FIVE,
PRIORITY: 5.0),
structure (ENTRY: POST-SIX,
PRIORITY: 6.0),
structure (ENTRY: POST-TWO,
PRIORITY: 7.0))

#125 9:45:17 a.m. POST-THREE removed from the queue at priority 3.0

#126 9:45:17 a.m. POST-FIVE removed from the queue at priority 5.0

#127 9:45:17 a.m. POST-SIX removed from the queue at priority 6.0

#128 9:45:17 a.m. POST-TWO removed from the queue at priority 7.0

#129 9:45:17 a.m. the priority queue
sequence = sequence ()

18

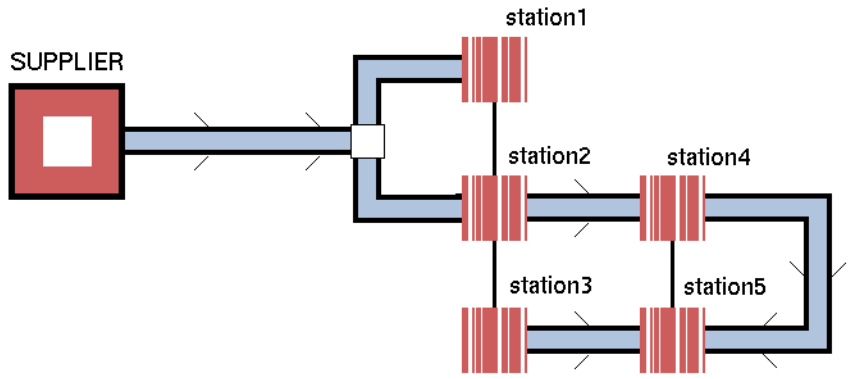
Connections

Describes connections, connection posts, and junction blocks.

Introduction

A **connection**, an item of the connection class, is a graphical item that creates a logical relationship between two or more objects. You can use connections to represent almost anything that provides a pathway or route between two or more objects. For instance, use connections in your knowledge base (KB) to represent electrical wires, pipes, roadways, or cables.

The next figure shows a simple schematic of objects and their connections.



In addition to representing physical entities such as pipes and wires, connections can represent more abstract relationships such as the cause of one event by another, or one event occurring later than another. A connection can also represent a transition from one state to another.

Note If you are using connections for abstract reasoning only and you are moving connected objects on a workspace, you can improve performance by making connections transparent or by not displaying the workspace at all.

Properties of Connections

Connections have the properties of elasticity, direction, style, line pattern, and arrowheads. When you drag an end of a connection, the connection becomes elastic, letting you lengthen or shorten it. Once a connection exists between two objects, dragging one of the objects stretches the connection. Each connection can be either non-directional or of a single direction (input or output), and be either diagonal or orthogonal in style. Connections can also have a line pattern, which can be solid, dashed, dotted, or a combination, and an arrowhead at the end.

In addition, connection definitions have a unique cross-section-pattern attribute, which lets you specify the connection width and color. By defining different cross-section patterns in your connection classes, you can create visually distinct

connections representing different flows in your KB. You can change the colors of a connection's cross-section pattern programmatically.

Using connections in your KB always involves stubs, and may also involve junction blocks and connection posts, described in Using Junction Blocks, and Using Connection Posts, respectively.

Controlling Connection Caching

When G2 caches graphical connections between objects, expressions that reference connections execute faster, but changing connections takes longer. When G2 does not cache connections, connection expressions take longer but changing connections is faster. The default behavior is not to cache them; however, either behavior may be preferable, depending on your particular application.

To change G2's connection caching behavior:

➔ Set the connection-caching-enabled? attribute of the Miscellaneous Parameters system table to yes or no.

The default setting is no, which suppresses connection caching. If your application often executes complex expressions that reference connections, and/or rarely changes connections during KB execution, performance may improve if you turn connection caching on.

Connecting to Objects

Connections are interrelated with objects: connections join objects, and objects can have connections. You add connections to an object by using stubs. A stub is a short connection located on the perimeter of an object icon with nothing on the other end.

All user-defined object classes are capable of having one or more stubs, which you define in the stubs attribute of the object definition. In the simplest case, the stubs attribute can define a system-defined connection, consisting of a single black line, one pixel wide. Alternatively, you can specify a user-defined connection class. Each stub must also include the stub location upon the object icon.

A stub can specify a direction (input or output), and include a user-defined name. Creating an object with a stub automatically creates one connection item for every stub. For a description of specifying stubs, see Specifying Connection Stubs.

While object definitions specify what connections an object has by default, connection definitions describe the connection, and allow it to be instantiated. A connection instance cannot exist without the object that uses it; an object is devoid of its specified connection unless the connection definition exists and is complete.

A connection definition specifies the class name, direct superior classes, class-specific attributes, cross-section pattern, stub length, and junction block for each instance of the connection class. For a complete description of specifying stubs for objects and creating connection definitions, see Creating Connection Classes.

Creating a Connection

While you can use the default connection class provided with G2, typically you create your own custom connection definitions and use them in your object definitions. This section presents a basic example to illustrate the process of creating a generic object with a user-defined connection.

You can create the definition for the connection and object in any order. However, if you specify a connection definition before it exists, instantiating the class will not produce a stub. For clarity, the following example creates the connection definition first.

To create a connection definition:

1 Select:

KB Workspace > New Definition > class-definition > class-definition.

or

KB Workspace > New Definition > class-definition > connection-definition

2 Open the definition table.

3 Edit the class-name attribute as red-connect.

4 Edit the direct-superior-classes attribute as connection.

5 Edit the cross-section-pattern attribute, for example:

outer = black, inner = red; 2 outer, 5 inner, 2 outer

For complete information on creating connection classes, see Creating Connection Classes.

To create an object definition:

1 Select:

KB Workspace > New Definition > class-definition > class-definition

or

KB Workspace > New Definition > class-definition > object-definition

2 Open the definition table.

3 Edit the class-name attribute as red-object.

4 Edit the direct-superior-classes attribute as object.

5 Edit the stubs attribute and enter a specification like this:

```
an input red-connect located at left 20;
an output red-connect located at right 20
```

You can edit the object icon if you wish. The example here uses the system-defined object icon, but with other colors.

For complete information on creating object classes, see Creating Object Classes.

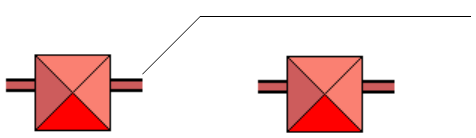
Connecting Objects

Once the object and connection definitions exist, you can use the connection. You cannot create a free-standing connection. A connection always begins as an object stub.

To use the red-connect connection:

1 Create two instances of the red-object class and place them on a workspace.

2 Click on the right-hand stub of the left object and drag the connection into the icon body of the right object, then click the pointer again to release the connection:



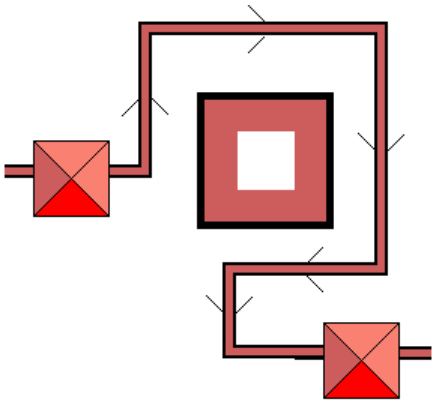
Using Connections

You form a connection by dragging a stub to another object. Click on the destination object to end the connection.

Drawing Orthogonal Connections

When you drag an orthogonal connection, it forms a right angle between the object and your pointer. To reach a destination, you may want to create additional bends along the connection. To create such angles, drag the connection with your pointer and click once at the location where you want to create an angle. Resume dragging (and optionally clicking to create angles) until you reach the destination

object, then drag the connection into the icon body and click to release the connection.



Hint If the connection seems to be stuck to the pointer and you cannot end it, double click or press Ctrl + a to release the connection.

To lengthen or shorten a connection between two objects, click on one of the objects and move it further away from or closer to the other object. The connection *stretches* or *shrinks* depending on which way you drag the object. If there are multiple bends in a connection, only the last two links shorten or stretch as you drag the connected object. To shorten an unattached connection, click on the free end and retrace the connection route towards the object. The connection *shrinks* as you drag the pointer.

Drawing Diagonal Connections

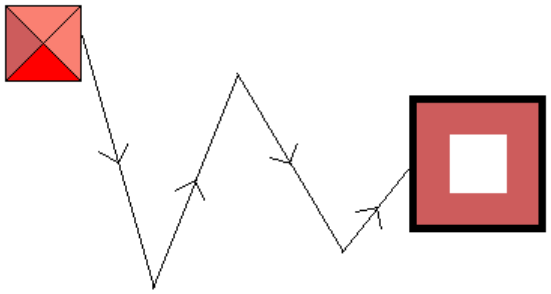
When you drag a diagonal connection, G2 draws the connection, using the specified line width and line pattern, in the color first mentioned in the cross-section specification for the connection class.

Drawing diagonal connections is slightly different than drawing orthogonal connections.

To draw a diagonal connection:

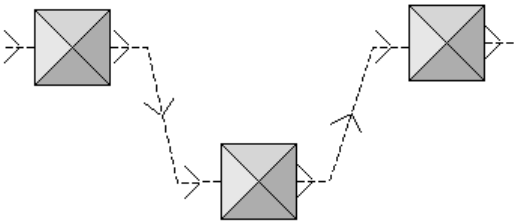
1. Click near the end of a diagonal connection, the last line segment becomes elastic.
2. Drag the connection to extend it and click where you want a bend. Repeat this step for as many bends you desire.
3. When the connection reaches a destination object, click to end the connection.

Here are some examples, using different line patterns:



Stubs an input connection located at left 25 with style diagonal and with line-pattern dash; an output connection located at right 25 with style diagonal and with line-pattern dash

CLASS-3

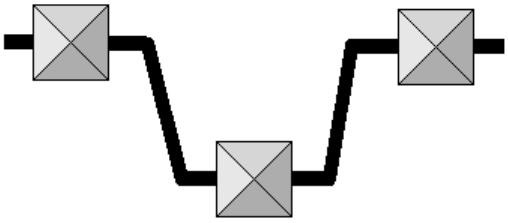


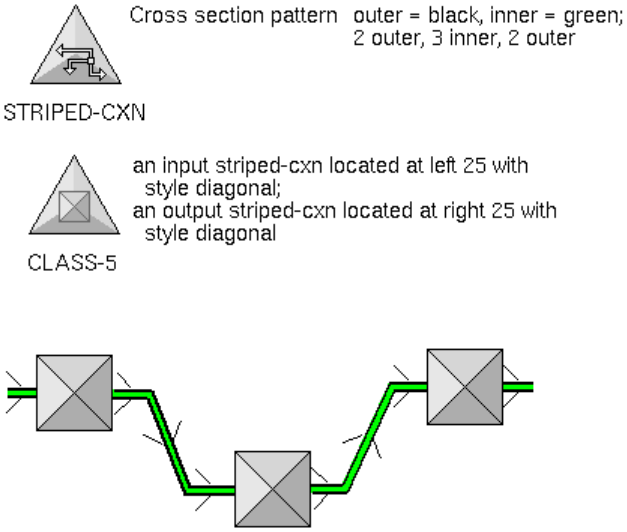
Cross section pattern color = foreground; 10 color

WIDE-CXN

Stubs a wide-cxn located at left 25 with style diagonal; a wide-cxn located at right 25 with style diagonal

CLASS-4



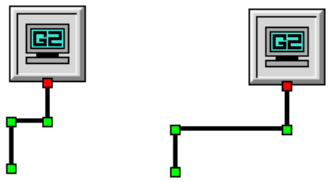


For diagonal connections, you cannot use the junction block that G2 creates automatically. You have to create a junction block subclass by using an object definition as described in Using Junction Blocks.

Note Junction blocks work exactly the same way for diagonal connections as for orthogonal connections, except that the junction blocks are not created automatically.

Changing Connection Vertices

You can interactively change the connection vertices of a connection by dragging the handles on the connection. For example:



To interactively change connection vertices, the show-selection-handles attribute in the Drawing Parameters system table must be true, the default.

Using Connection Arrowheads

You can configure the connection-arrows attribute of connection instances to have one of these values:

- default — Directional arrows along the length of the connection, which is the current behavior.
- none — No arrows anywhere on the connection.
- *arrow*[, *arrow*] . . . — A single arrow specification or a list of arrow specifications, separated by commas.

where:

- *arrow* = *shape* | *adjectives shape* | *shape place* | *adjectives shape place*
- *shape* = arrow | triangle | diamond | circle
- *adjectives* = *adjective* | *adjective adjectives*
- *adjective* = filled | open | wide | narrow | large | small | thin | thick
- *place* = at the {output | input} end | at both ends | along the length

Note The following combinations are not valid: wide circle and narrow circle.

Here are some examples:

arrow

filled triangle

large thick arrow

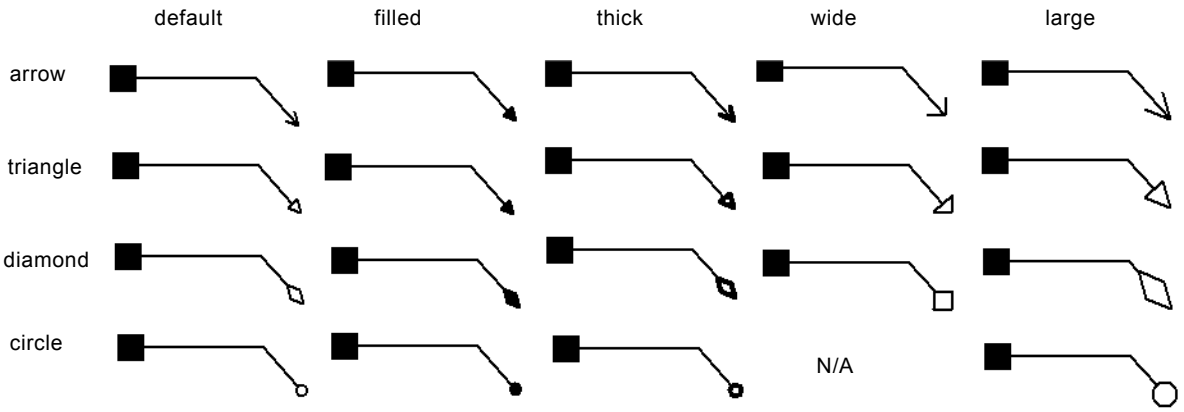
diamond at both ends

open large circle at the input end, filled small circle at the output end

Arrows scale with the width of a connection.

This feature is accessible in TW, G2, and the Workspace View ActiveX control.

This figure shows various combinations of arrowhead styles:



Connecting to Objects without Stubs

You can lengthen a stub by dragging it with the pointer and then connecting it to another item. While at least one object must have a stub to begin a connection, other items that you connect to may or may not have stubs.

For example, the next diagram shows a grouping of four objects before and after they are connected. Though one object originally has stubs, you can connect it to any one of the other objects, as the right-hand grouping illustrates.

After you connect a stub to an item that did not previously have a connection, deleting the connection leaves the stub intact.

Defining Connectedness

Objects are considered connected to one another only when a direct path exists between them, consisting of connections or junction blocks. Other objects cannot exist between two objects in a connection. Two objects are:

- **Directly connected** when a route exists between them that consists of a sequence of two or more connections linked by connection posts and/or junction blocks.
- **Indirectly connected** when a route of objects exists between them that consists of a sequence of one or more objects, in which each object are **directly connected** with the next one in the route.
- **Not connected** otherwise.

Thus connectedness is not transitive for direct connections: A connect B and B connect C does *not* imply A connect C. Note that two items could be both directly and indirectly connected if more than one route exists between them.

Notice that, for connection helpers (connections, connection posts and junctions) the definition of connectedness between them and normal objects are of different approaches. For example, follow table shows the expected results when using different type of items in system defined function items-are-connected().

Class of item1	Class of item2	items-are-connected(item1, item2)
block[a]	block	True iff[b] no other blocks between them
connection	block	True iff no other blocks between them
connection post	block	True iff no other blocks between them
junction	block	True iff no other blocks between them
connection	connection	True iff there's only one junction/CP[c] between them
block	connection	True on directly attached connections to the block
junction	connection	True on directly attached connections to the junction
connection post	connection	True on directly attached connections to the CP
connection	connection post	True on directly attached CPs to the connection
connection post	connection post	True iff no other block/CPs between them
block	connection post	True iff no other block/CPs between them
junction	connection post	True iff no other block/CPs between them

Class of item1	Class of item2	items-are-connected(item1, item2)
block	junction	True iff no other block/junctions between them
connection post	junction	True iff no other block/junctions between them
connection	junction	True iff no other block/junctions between them
junction	junction	True iff no other block/junctions between them

a. block: class or subclass of item other than class or subclass of connection, connection-post and default-junction.
b. iff: if and only if.
c. CP: connection post.

Notice that the result is not symmetrical if swapping the order of two arguments. This is reasonable, and there's a theorem between KB expression for each ... connected to and the system-defined function items-are-connected:

for each item I connected to B => items-are-connected(B, I) is true

Also, connection posts (CPs) are special: they're supposed to connect items appears on different workspaces. Given the fact that a connection post could have multiple names, two connection posts are considered as the "same" just if there's one shared name between them.

Disallowing Connections

G2 does not allow connections to items for which connections do not make sense, such as logbook pages or a readout tables.

All G2 items that permit connections can restrict other items from connecting to them by using this configuration statement: declare properties as follows : not manual connections.

Determining the Item Count for Connections

When you create objects with stubs, each stub counts as a single item in your KB. The item count changes as you connect and delete connections as follows:

- Joining two stubs changes the number of connections from two items to one (a single connection between two objects).
- Joining a stub to an item without a stub, retains one item.
- Deleting a connection from an item originally without a stub leaves the stub and thus *adds* a connection item to the KB.

Note If you create a transient item with stubs but do not transfer the item to a workspace, looping over every connection with a statement such as for each connection connected to *my-object* do, will not locate connections of any items that are not on workspaces.

Deleting Stubs and Connections Interactively

To delete a stub:

1 Click on the end of the stub and drag it inside of the object icon.

2 Click to release.

The stub is deleted.

To delete a connection:

1 Click on the connection between two objects to display the connection menu.

2 Choose delete.

The connection is deleted without confirmation.

Deleting Stubs and Connections Programmatically

To delete stubs programmatically, use the delete action, described in delete.

Connection Layering

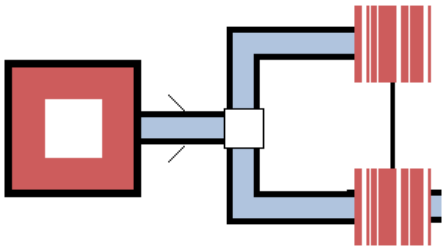
As Layering Items upon the Same Workspace describes, each item upon a workspace has an associated item-layer-position. Connections are no exception. The item-layer-position of a connection is always above the object to which it is drawn to. G2 displays connections is this manner regardless of whether the connections are drawn interactively or programmatically.

Connections are layered above the object at their input end. For non-directional connections drawn interactively, this is the end from which you drag the stub, or the *from* object for connections drawn programmatically.

Connections with the same object at their input end are layered in the order you create them.

Using Junction Blocks

While connections provide the graphical and logical connections between objects, you may also need branching connections. The item at the junction of two connections, shown next, is called a junction block.



Creating Junction Blocks

You do not have to create junction blocks manually. G2 creates a junction block for you automatically whenever you join one connection to another. The style of the junction block depends on how the connection has been created.

A junction block is an item of the default-junction class. If you are using the system-defined, default class of connection, G2 creates an instance of a default junction object whenever you join two connections.

If you create your own connection subclass, *and* specify a cross-section pattern, G2 creates a new default junction class automatically. The name of the class is the name of your connection class with the prefix junction-block-for. For example, if the name of your connection class is water-line and you have specified a cross-section pattern for it, G2 creates a default-junction subclass called junction-block-for-water-line. If you have used a connection definition to define your connection, this class name appears in your connection definition table in the junction-block attribute. The default junction class will not appear on a class definition. You can use the Inspect Facility to check for its existence.

Whenever you join two connections of the subclass, G2 creates an instance of the customized junction block automatically. You can connect two connections of different classes, as long as they have identical cross-section patterns.

Whenever possible, we recommend that you use the junction block that G2 creates dynamically for you. However, if you need a specific kind of junction block, perhaps one with additional attributes, or for use with a diagonal connection, create a new class. If you wish to name your default-junction class using the junction-block naming scheme that G2 uses, define your default-junction class before you define your connection class, otherwise the automatically created default junction will preempt the name.

Creating a Junction Block Subclass

To create a new junction block subclass:

1. Create an object definition.
2. Name the class by completing the class-names attribute.
3. In the direct-superior-classes attribute, specify default-junction as the superior class.

Once the class exists, use it whenever you need a junction block by selecting:

KB Workspace > New Object > *junction-block-name*

where *junction-block-name* is the name of the class you created. You can join the connections to the junction block as necessary.

Using Connection Posts

If you need to connect objects across workspaces, you can do so by using a connection post. A connection post is an instance of the connection-post class of objects, analogous to the connection posts found in flow diagrams and electrical schematics. Using a connection post lets you connect two or more objects on different workspaces, indicating that endpoints of connections on separate workspaces are actually joined.

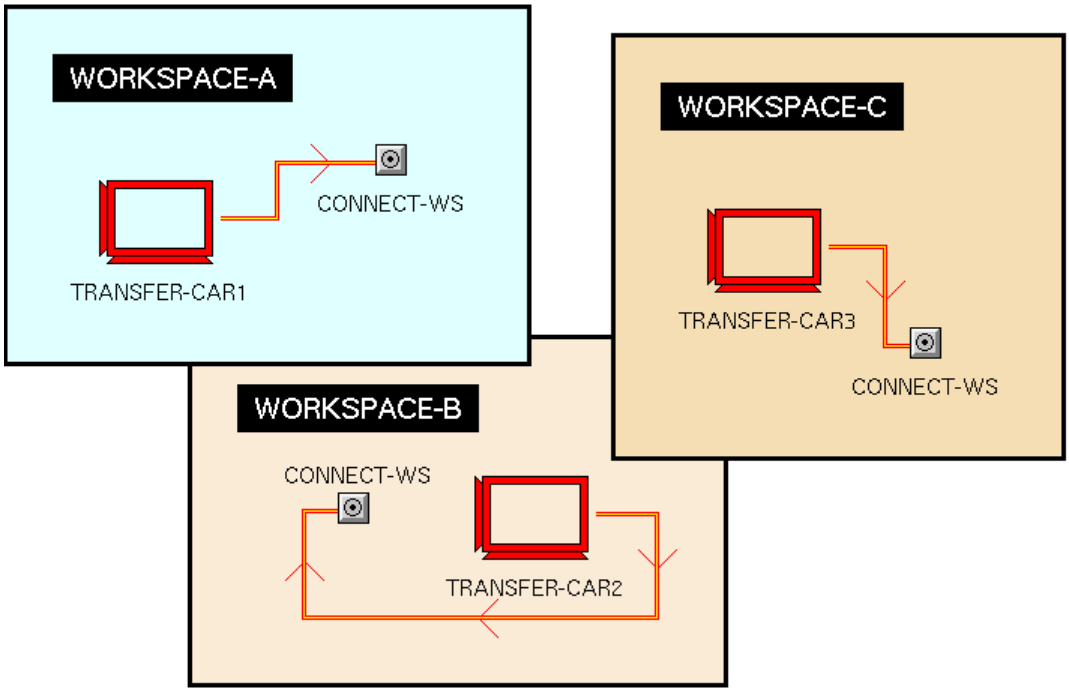
To create a connection post:

➔ Select KB Workspace > New Object > connection-post.

The names attribute specifies the name of the connection post. All connection posts of the same name are connected to each other, regardless of their locations. Connecting an object to a connection post connects it to any other object connected to that connection post, and to any object connected to any other connection post having the same name.

Note Using connection post as argument of items-are-connected is allowed. However, if there're multiple connection posts having the same name, calling items-are-connected with explicit name of connection posts may not get the correct result. That's because G2 will do conflict name resolving when compiling the related KB code involving the conflict name. For example, suppose there're two connection posts having the name CP1, when compiling the expression items-are-connected(CP1, AA), G2 will internally assign names like CP-XXX-CP1-1 to one of the connection posts and use that internal name in the expression. However, according to the asymmetry of items-are-connected, if connection posts were at the first argument place, the internal searching process will still consider all connection posts with the same original name.

For example, this figure shows three connection posts of the same name, connect-ws. All of the objects, transfer-car1, transfer-car2, and transfer-car3 are connected to each other via the connect-ws connection posts.



Hint Although connection posts are most often used to connect objects across workspaces, you can also use them to connect objects on the same workspace. Also, you can attach any number of connections to a connection post.

You cannot connect objects with opposing directions of flow.

Creating Connection Posts on Subworkspaces Automatically

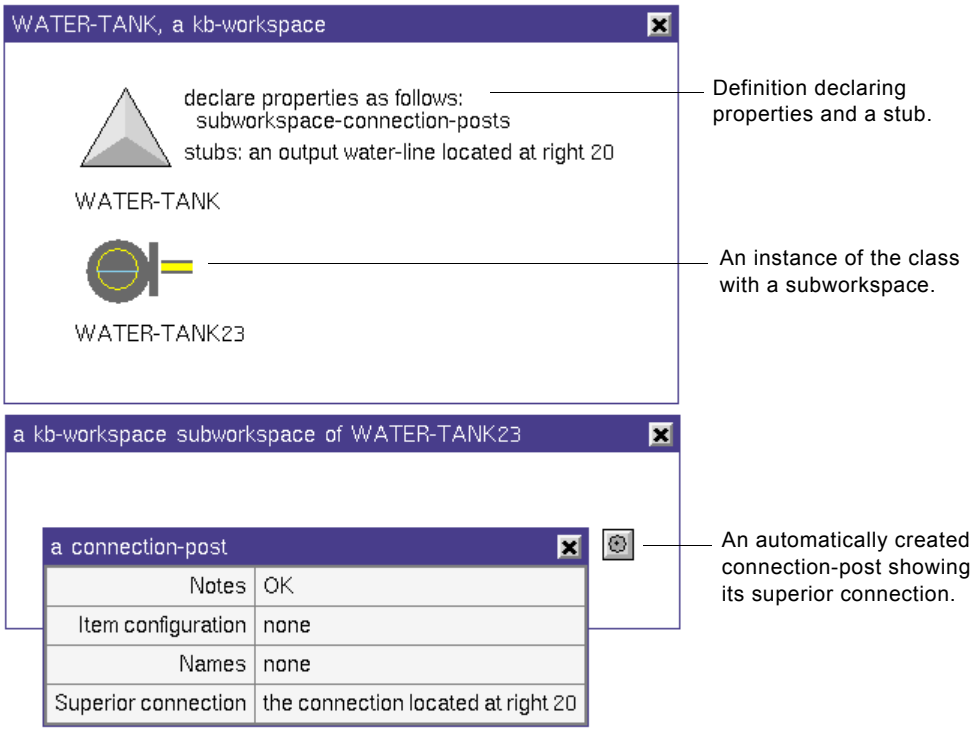
G2 provides a special configuration clause that affects how you use connection posts and how items are connected through the workspace hierarchy. The configuration clause is:

declare properties as follows : subworkspace-connection-posts

This configuration is a convenient way to create connection posts automatically upon the subworkspaces of objects. If an object definition includes this configuration statement in its instance-configuration attribute (*not* item-

configuration), whenever a subworkspace is created for that class, G2 automatically creates a connection post for each connection stub on the class icon.

For example, the next diagram shows the water-tank class, with its instance configuration declaring subworkspace connection posts. An instance of this class, water-tank23, has a subworkspace, shown below it. Upon the subworkspace is the connection post that G2 creates automatically due to the configuration clause. Further, the connection post has a completed superior-connection attribute, indicating the specific connection to which it is attached.



Creating a Connection Post Subclass

To create a connection post subclass:

1. Create an object definition.
2. Name the class by completing the class-names attribute.
3. In the direct-superior-classes attribute, specify connection-post as the superior class.

Once the class exists, use it whenever you need a connection post by choosing:

KB Workspace > New Object > *connection-post-name*

Using Connection Expressions

G2 provides a powerful language for referring to connections and objects connected to other objects. The expressions you use with connections make it possible, for example, to write generic rules that refer to any class connected to any object.

In all connection expressions, you can refer directly to the object (if appropriate), to a name on the object (the *name*), or to a direction (an input to, an input of, an output of) of the connection.

Expressions do not exist for referencing a connection as an item upon a workspace such as:

if there exists a *connection* upon *workspace-name*
the count of each *connection* upon *workspace-name*

While the Text Editor lets you enter such statements, they are ineffective and always return false, since connections cannot exist as autonomous items upon a workspace.

Note G2 ignores junction blocks when looking for connected objects unless you refer to the, any, or every junction-block (or a subclass of junction-block) connected to an object or connection within the expression.

These generic reference expressions reference items that are connected to other items or that are attached to particular connections.

Referring to Connected Items

To iterate over one or multiple items that are connected:

➔ the *class-name* [*local-name*] connected to *item*
-> *item*

With the the quantifier, this generic reference expression produces the one and only item of the specified class that is connected in any way to the specified item. With the any quantifier, this expression produces the set of items of the specified class that are connected in any way to the specified item. For example, to refer to items connected to other items:

for any file-marker F connected to page-marker1

for any file-marker F connected to any page-marker

Referring to Input or Output Stubs

Whenever stubs are defined with a direction (input, output), you can refer to them using the flow direction (input to, input of, or output of).

To refer to input or output stubs:

➔ the *class-name* [*local-name*] connected at an
{input to | input of | output of } *item*
-> *item*

With the the quantifier, this generic reference expression produces the one and only item of the specified class that is connected at any input stub or output stub of the specified item. With the any quantifier, this expression produces the set of items of the specified class that are connected at any input or output stub of the specified item. The input to and input of phrases are equivalent; use one or the other to improve the readability of your code. For example, to refer to objects connected at a particular flow direction:

```
for any file-marker F connected at an output of any page-marker
```

Input and output stubs of objects are described in Specifying Connection Stubs.

Referring to Port Names

The place at which a connection attaches to an icon is called a port. You can give these locations names by defining stubs with port names in the object definition.

To refer to port names:

➔ the *class-name* [*local-name*] connected at the {port*name* of | input to |
input of | output of } *object*
-> *item*

With the the quantifier, this generic reference expression produces the one and only item of the specified class that is connected at a named port or at any input stub or output stub of the specified object. With the any quantifier, this expression produces the set of items of the specified class that are connected at a named port or at any input or output stub of the specified object. The input to and input of phrases are equivalent; use one or the other to improve the readability of your code. For example, to refer to connected objects using a port name:

```
for any file-marker F connected at the infile-port of any page-marker
```

Another example:

```
if the status of any valve V connected at the water-input-for any tank is
    blocked then inform the operator that
    "[the public-name of V] is blocked; check it immediately!"
```

This generic if rule checks whether the status attribute contains the symbol blocked for any valve that is connected to any tank at its water-input-for port. Input and output stubs of objects are described in Specifying Connection Stubs.

Tip This example illustrates a naming convention for portnames in an object definition whereby you include a preposition as a name suffix. In the example, water-input-for is the portname. This convention makes the expression easier to read by G2 developers, knowledge engineers, and application users.

Referring to the End of a Connection

G2 can iterate over any specified object connected to the (or an) input end of, output end of, or either end of a connection.

To refer to the end of a connection:

➔ the *class-name* at {an input end | an output end | either end} of *connection*
-> *item*

With the the quantifier, this generic reference expression produces the one and only item of the specified class that is at an input end, output end, or either end of the specified connection. With the any quantifier, this expression produces the set of items of the specified class that is at an input end, output end, or either end of the specified connection. For example, to refer to objects using the connected direction only, without a connected at statement:

```
for any truck T at an input end of any km-connection
    unconditionally post "[the name of T] is connected!"
```

Referring to the Connection Class

G2 can iterate over any item that uses a specific connection class.

To refer to the connection class:

➔ the *connection-class-name* [*local-name*] connected to *item*
-> *item*

With the the quantifier, this generic reference expression produces the one and only connection class that is connected in any way to the specified item. With the any quantifier, this expression produces the set of connections of the specified class that are connected in any way to the specified items. For example, to refer to a specific connection class:

```
for any km-connection C connected to any truck T upon this workspace
    unconditionally post "[the name of C] is connected to T"
```

Iterating over Connections

You can iterate over all of the connections of an item, or you can iterate of a subset of item connections by specifying a particular connection class, a -name, or whether the connection is an input or output connection.

To iterate over connections:

➔ for *local-name* = each *connection-class connection-spec item*

The *connection-spec* phrase uses this syntax:

```
{connected to |
 connected at the -name of |
 connected at the input to |
 connected at an input to
 connected at the input of
 connected at an input of
 connected at the output of |
 connected at an output of }
```

Here is an example that iterates over all the connections of an item and deletes them:

```
for C = each connection connected to warehouse-123
    do
        delete C removing connection stubs without permanence checks
    end
```

This example constrains the iteration to pipe-line class connections connected to an output of an object:

```
for PL = each pipe-line connected at an output of station3
    do
        change the inside stripe-color of PL to yellow;
        for OBJ = each object connected to PL
            do
                post "[the name of OBJ] is connected"
            end
    end
```

Using Actions with Connections

Several actions exist that manipulate connections. Following is a description of each action, along with an example.

Changing the Stripe-Color

When you define regions for the cross-section pattern of a connection class, you can refer to those regions.

To refer to connection regions:

➔ change the *connection-region* stripe-color of *connection-class-name* to {*color-name* | *symbolic-expression*}

An example is:

```
change the electrical-wire stripe-color of km-connection to red
```

You can also provide a symbol of the form RGB*rrggbb* as a valid color name, where *rr*, *gg*, *bb*, are the 8-bit hex values for red, green, and blue. For details, see Other Literal Terms.

Creating Transient Connections

Connections can be of any connection class and may be orthogonal or diagonal, with or without direction, and located at an existing stub or in an entirely new position. G2 creates stubs for connections newly locating it at some location on an object; otherwise, G2 requires that a stub already exist at the specified position (locating it at).

You can create a connection with a name or without, by specifying with name none. You can create a connection on one object (connected to) with the other end of the connection left free, or between two objects (connected between). The create action fails if one or both objects has a configuration statement specifying not manual-connections.

Transient connections are orthogonal by default. When the connection is orthogonal, G2 places a bend at the end of each specified distance. If the connection is diagonal, G2 places a bend after each two distances, with a segment connecting bends. You can think of a diagonal connection as connecting odd-numbered bends in an imaginary orthogonal connection.

To create a connection using the create action:

➔ create a connection [*local name*] [of {class *connection-class-name* | the class named by *symbolic-expression*}] connected {*between-spec* | *to-spec*} *connection-spec* [, ...]

The *between-spec* and *to-spec* uses this syntax:

{between *item from-position* and *item to-position* | to *end-position*}

Element	Description
from-position	The item from which the connection is being drawn.
to-position	The item to which the connection is connecting.
end-position	See the following syntax description.

The *end-position* uses this syntax:

{ [at *-name*] [[newly] locating it at
{ {left | top | right | bottom} | the side named by *side-expression*} } {integer |
at the position given by *quantity-expression*} [at {*-name* |
the named by *-name-expression*}]]

The *connection-spec* uses this syntax:

{with style {diagonal | orthogonal} |
with the style named by *symbolic-expression* | with vertices *integer* [,...] |
with the vertices given by *integer-g2-list* |
with direction {input | output} |
with the direction named by *direction-expression* }

Here is an example that creates a connection on one object:

```
create a connection connected to d1 newly locating it at right 15
```

Here is an example that creates a connection between objects:

```
create a connection of class km-connection connected between d1
    locating it at right 15 and d2 locating it at left 20 with direction input
```

Here is an example that creates a connection with vertices:

```
create a connection of class km-connection connected to d1
    newly locating it at right 20 with direction output, with vertices 20 20 30 40
```

The next sections illustrate several ways of creating connections and explain the statements.

Creating a Connection on One Side of an Object

This action creates a transient system-defined connection on the right side of the D1 object.

```
create a connection connected to d1 newly locating it at right 15
```

Note To create a connection without direction, omit the with direction statement completely, as in this example. You cannot use the statement with direction none.

The newly locating it at statement indicates that no stub currently exists at the given location. If you omit this statement in any create action by stating only the position (at right 15), G2 requires that a stub exist at the location you specify.

Creating a Directional Connection

This action connects two objects with a transient connection located at existing stub locations of both items:

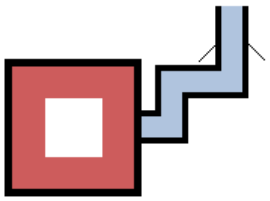
```
create a connection of class km-connection connected between d1
    locating it at right 15 and d2 locating it at left 20 with direction input
```

In this case, the statement also specifies a user-defined connection class, km-connection, to use and a direction (input).

You can make transient connections only to transient stubs, not to permanent stubs. If you create a connection to a stub, and the stub already has a connection on it with the other end free, G2 deletes both the old connection and the old stub, and creates a new stub.

Creating a Connection with Vertices

This action creates a connection, with specific vertices, on one side of an object, quarters, as the example illustrates:



```
create a connection of class pipe-line connected to quarters
    newly locating it at right 45 with direction output, with vertices 20 30 40 50
```

Specifying Connection Vertices

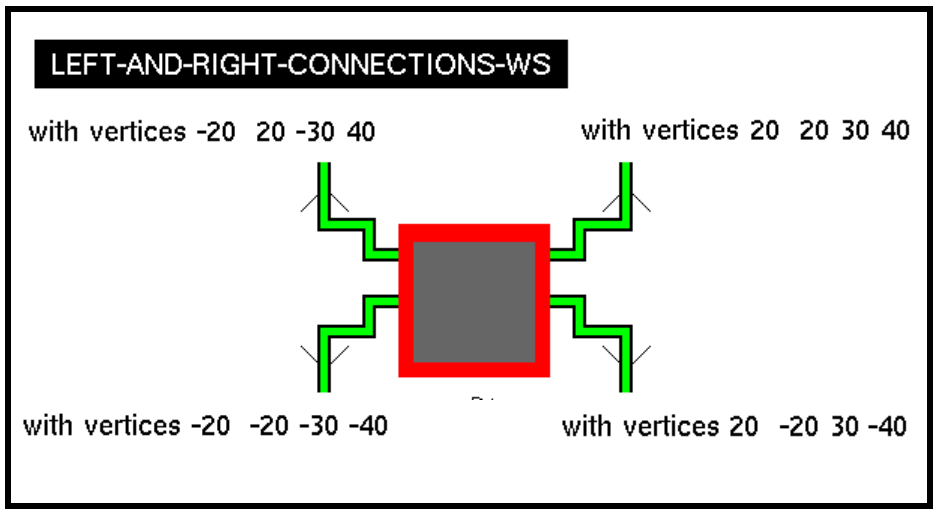
You specify vertices (bends) for a connection with a list of numbers, which determine how G2 draws the segments and vertices of a connection. Separate vertex numbers with a space character, not a comma (,).

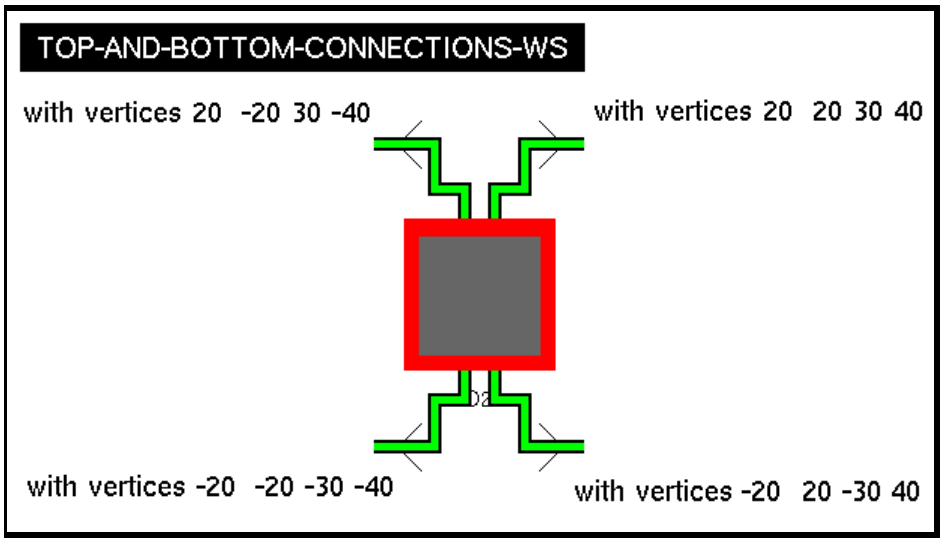
Vertex numbers are relative to the starting point of the connection (top, bottom, right, or left). Positive numbers extend upwards and to the right of an icon, negative number extend downwards or to the left. The second number in a vertex specification determines in which direction the connection continues. After the second vertex, the pairs of values determine how G2 draws the remaining segments and vertices of the connection.

For example, in the previous diagram, the first number specifies the distance (20 workspace units) that G2 extends the connection from the icon before creating the first vertex. Both right and top connections require a positive first number to extend away from the object. Connections beginning at the left and bottom of an icon require a negative value first number to extend outwards from the object.

For connections that begin on the left or right side of an icon, the odd number vertices (1, 3, 5, etc.) always indicate a horizontal part of the connection, while even number vertices (2, 4, 6, etc.) specify the vertical connections. This rule reverses for connections that begin at the top or bottom of an icon — odd number vertices specify vertical connections, even numbers horizontal directions.

To illustrate this, the next two diagrams show how to specify the vertices for the same connection, starting first on the left and right sides of an icon, and starting second at the top and bottom of an icon, extending in different directions.





Creating an Existing Connection Programmatically

A common requirement in KBs is to obtain the connections of an existing item and create them elsewhere on other items.

G2 provides the g2-get-connection-vertices system procedure for this purpose. The system procedure populates an integer list with the lengths of the connection segments and vertices of an existing connection. Positive values specify a segment extending upwards or to the right of an object or vertex, and negative numbers specify a segment extending downwards or to the left. Once populated, you can

then use the list as the vertices specification for the create connection action and its given by *integer-list* grammar.

The integer list that the system procedure populates contains a minimum number of connection segments and vertices, typically one or two less than those of the original connection. When creating a new connection between two objects, the create action uses the vertices contained in the list, and then determines the last one or two vertices based on the location of the item to which the newly created connection is being attached.

Note The g2-get-connection-vertices system procedure has changed in recent G2 releases and currently returns the minimum number of vertices that the create action requires, rather than the exact number of vertices from the original connection. For KBs that may have relied on the previous behavior, a backward compatibility option exists to revert the system procedure to its previous behavior. For information about that option, see Changing the Backward Compatibility.

The following procedure illustrates one way to use the system procedure in conjunction with the create connection action. This code creates a new object with connections identical to those of an existing item, which is passed to the procedure as an argument.

```
gds-get-connection-vertices(from-item: class object)
IL: class integer-list;
New Object: class red-object;
C: class connection;

begin
    create an integer-list IL;
    change the name of IL to the symbol vertices-list;
    transfer vertices-list to this workspace at (50, -20);
    C = the connection connected to from-item;
    call g2-get-connection-vertices(from item, C, vertices-list);
    create a red-object NewObject;
    transfer New Object to this workspace at (50, 50);
    create a connection C of class connection connected to NewObject
        locating it at right 20 with the vertices given by vertices-list;
    make NewObject permanent
end
```

Making a Transient Connection Permanent

Use the make action to make a transient connection permanent. The next example appends the make permanent action to the create action:

```
in order
create a connection C of class km-connection connected to d1
    newly locating it at right 10 with direction output and make C permanent
```

Deleting a Connection

The delete action deletes a transient connection. You can delete a permanent connection by using the without permanence checks grammar. Deleting a connection leaves the stubs, unless you specify the optional removing connection stubs phrase, shown here:

```
delete pipe-connection57 without permanence checks removing connection stubs
```

Making an item with connections transient, and then deleting that item, automatically deletes the connection stubs.

Hint Transient connections and their stubs are deleted whenever you reset the KB.

Detecting Connection and Disconnection Events

Several rules can detect changes in any connection, or in a direct connection only. You can constrain the execution of the body of a rule that detects a connection/disconnection event by making such execution conditional on one of the functions described under Detecting Connectedness.

You cannot use rules to detect changes in the connection status of default-junction and connection-post items because those items are considered transparent to the connection search.

Generic Connection and Disconnection Events

A whenever rule can detect the establishment or breaking of a connection between two items irrespective of:

- Whether the connection is direct or indirect.
- The class(es) of the connection(s) that connect the items.
- The direction(s) of the (s) to which the connections attach.

To detect generic connection and disconnection events:

➔ whenever *item* is connected to *item*

➔ whenever *item* is disconnected from *item*

For example:

```
whenever any temp-control-monitor C
is connected to any outer-thermometer T
    then start check-temp-progress (C, T)
```

Direct Connection and Disconnection Events

A whenever rule can detect the establishment or breaking of a direct connection between two items. The rule can fire for any direct connection, or only if the connection is:

- Of a specified connection class.
- Has a specified name.
- Has a specified direction.

To detect direction and disconnection events:

➔ whenever a connection [of class *class*]
is directly {connected to | disconnected from}
[an input of | an output of | the *name* of] *item*

When referring to a connecting that is directly connected to an item, G2 permits the use of a local variable. You cannot use a local variable when referring to a connection being directly disconnected *from* an item. Not using a local variable for a disconnected connection prevents the possibility of attempting to refer to a deleted item in the rule consequent.

Here are some examples:

whenever a connection of class pipe-connect is directly disconnected
from any tank T then start check-connection (T)

whenever a connection X is directly connected to
inflow- of any tank T then start -inflow-rate (T, X)

whenever a connection of class my-connect is
directly connected to an input to any tank T
then conclude that the input-connection of T is true

System Procedures for Connections

G2 provides two system procedures for use with connections:

g2-get-connection-vertices
(*connected-item*: class item, *connection*: class connection,
integer-list: class integer-list)

This procedure accepts a single item with a connection, and populates an integer list with the connection segments and vertices, as described in Creating an Existing Connection Programmatically.

g2-get-items-connected-to-port
(*connection-source*: class item, *connected-class*:
symbol, *port-name*: symbol, *connected-items*: class item-list)

This procedure finds items connected to the port of any given item, where the port name is given by an expression.

Functions for Connections

Several system-defined functions provide two categories of information about connections:

- Type and location of connections.
- Existence of connectedness.

Checking Connection Information

These are the functions that check the:

- Connection direction.
- Portname of a connection.
- Connection position.
- Side of a connection.
- Connection style.

To check the connection direction:

➔ connection-direction
(*item1*, *connection1*)
-> *symbol*

Returns input, output, or none to indicate whether *connection1* connected to *item1* is directed and, if so, whether it is an input or output connection. The *connection1* must be connected directly to *item1*.

If this function returns a direction of none, you cannot specify with direction none in a connection statement. Instead, do not enter any direction statement.

To check the connection portname:

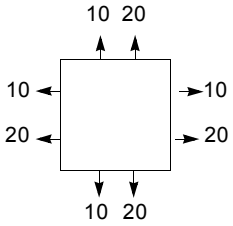
➔ connection-portname
(*item1*, *connection1*)
-> *symbol*

Returns a symbol indicating the portname at which *connection1* attaches to *item1*.

To check the connection position:

➔ connection-position
(*item1*, *connection1*)
-> *integer*

Returns an integer indicating the position along the side of *item1* at which the *connection1* is attached. The value is relative to the side of the icon. For example, if an icon is 30 workspace units square, here is how the connections at locations 10 and 20 are returned:



To check the connection side:

➔ connection-side
(*item1*, *connection1*)
-> *symbol*

Returns top, bottom, left, or right, indicating the side of *item1* to which the *connection1* is attached.

To check the connection style:

➔ connection-style
(*item1*, *connection1*)
-> *symbol*

Returns either diagonal or orthogonal as the style of *connection1*.

Detecting Connectedness

These functions are predicates that check:

- Whether two items are directly connected in any way.
- Whether two items are directly connected in a specified direction.
- Whether two items are connected via specified (s).

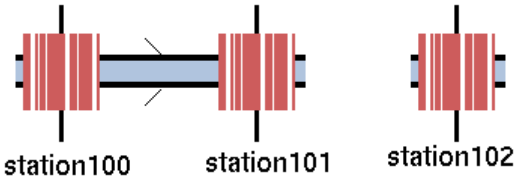
For a description of what constitutes connectedness, see Defining Connectedness.

You can constrain the execution of the body of a rule that detects a connection/disconnection event (described under Detecting Connection and Disconnection Events) by making such execution conditional on one of these functions.

To check whether two items are directly connected in any way:

➔ items-are-connected
(*item1*: class item, *item2*: class item)
-> *truth-value*

Returns true if any connection whatsoever directly connects *item1* and *item2*; else false. Example:

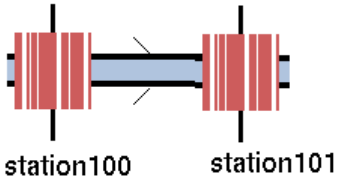


items-are-connected(station100, station101)	true
items-are-connected(station100, station102)	false

To check whether two items are directly connected in a specified direction:

➔ items-are-connected-with-direction
(*item1*: class item, *item2*: class item, *item1-direction*: symbol)
-> *truth-value*

Returns true if a connection runs from *item1* to *item2* and has the specified direction relative to *item1*; else false. Example:

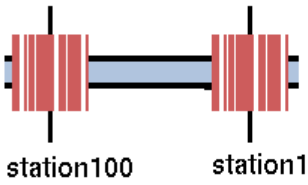


items-are-connected-with-direction(station100, station101, the symbol output)	true
items-are-connected-with-direction(station100, station101, the symbol input)	false

To check whether two items are directly connected at specified ports:

➔ items-are-connected-at-ports
(*item1*: class item, *item2*: class item,
name1: symbol, *name2*: symbol)
-> *truth-value*

Returns true if a connection runs from *item1* to *item2* and connects to *name1* on *item1* and *name2* on *item2*; else false. To specify that a name does not matter, specify any. Example:



items-are-connected-at-ports (station100, station101, the symbol right-port, the symbol left-port)	true
items-are-connected-at-ports (station100, station101, the symbol right-port, the symbol any)	true
items-are-connected-at-ports (station100, station101, the symbol any, the symbol left-port)	true
items-are-connected-at-ports (station100, station101, the symbol left-port, the symbol any)	false

Describing Connections

Use the Describe facility to provide useful information about the location and direction of connections.

Click directly on a connection to get to its item menu and select describe. If you click too close to the connection, you will get the item menu for the object, rather than the connection. Similarly, if you click too close to an unconnected end of a connection, you will extend the connection instead of getting the item menu.

G2 provides this information for a directional connection:

Description of PIPE-LINE-XXX-9.

Input end connected to SUPPLIER at right 45
Output end connected to JUNCTION-BLOCK-FOR-PIPE-LINE-XXX-8 at left 13 at port JUNCTION-BLOCK-CONNECTION-3

If a connection is non-directional and the horizontal distance between connection endpoints is greater than the vertical distance between connection endpoints, describe displays information such as the following:

```
left end connected to enet--1 at enet-out and right end
    connected to enet--2 at enet-in
```

If the connection is non-directional and the vertical distance is greater than the horizontal, describe displays information such as the following:

```
top end connected to enet--1 at enet-out and bottom end
    connected to enet--2 at enet-in
```

where enet-out and enet-in are optional names.

19

Relations

Describes how to associate items in a non-graphical way.

Introduction

A relation associates items in a KB, without drawing physical connections between the items. A single item can be related to one or more items, and multiple items can be related to one or more items.

You can create and break relations programmatically, which can cause forward chaining in rules. You can also refer to items based on their relationship to other items, and use relations to limit the scope of generic statements.

Using Relation Definitions and Relations

A **relation definition** creates a type of association between items of a first class and items of a second class. A **relation** is an association of a particular name between two particular items. A relation is an item of the relation class.

- The first class of a relation is known as the **relation source**.
- The second class is known as the **relation target**.

Note There is no class within G2 called *relation definition*. However, this chapter uses the term to differentiate between the item you create to specify the name and properties of a relation (the *relation definition*) and the association that can exist between items (the *relation*), based on that definition.

To use relations in your KB, first you create a relation definition, then you use the conclude action to create, remove, or replace relations based on their definition.

For example, you can define the relation the-holding-tank-for to represent an association between two items of the class tank. Based on this relation definition, you can conclude that one tank is the-holding-tank-for another tank.

By default, relations are transient: they disappear when you reset G2, and are not included in a saved KB. You can create permanent relations in your KB as described in Using Permanent Relations.

You can work with relations programmatically in the following ways:

- Forward chain to rules that detect when relations are established and broken.
- Use an item expression to test for the existence of a relation.
- Use an item expression to refer to items that participate in relations.

Creating a Relation Definition

When you create a relation definition, you are creating an instance of the relation class. For a summary, see The Relation Class.

To create a relation definition:

1. Select KB Workspace > New Definition > relation.
2. Display the relation definition table, and edit the relation-name attribute to specify the relation name.
3. If the relation is not to be symmetric, edit the inverse-of-relation attribute to specify an inverse name.
4. Enter the relation source class in the first-class attribute.
5. Enter the relation target class in the second-class attribute.

When you create a relation based on its definition, you conclude that one or more items of the first-class are related to one or more items of the second-class via the relation-name. For more information, see Creating a Relation.

Choosing a Relation Name

When specifying the relation-name attribute of a relation definition, choose a name that is descriptive and meaningful when read in a conclude action.

Unlike most items, a relation definition does not include a names attribute. The name of a relation is the symbol you provide in the relation-name attribute. For a description of how you use the conclude action with relations, see Creating a Relation.

For example, you might conclude that a particular tank is the-holding-tank-for another tank:

```
conclude that tank-1 is the-holding-tank-for tank-2
```

When concluding that a particular tank is one of many holding tanks for another tank, you might use the prefix *a* in the relation name:

```
conclude that tank-1 is a-holding-tank-for tank-2
```

Sometimes relation names refer to a location, for example, a bottle might be located-at a station:

```
conclude that bottle-1 is located-at washing-station
```

Using Permanent Relations

You can create permanent relations by completing a relation definition and changing the value of its relation-is-permanent attribute to yes. The relation then persists when you reset G2, and is included when you save the KB. provided that the items that participate in the relation comply to permanency, as described under Complying to Permanency.

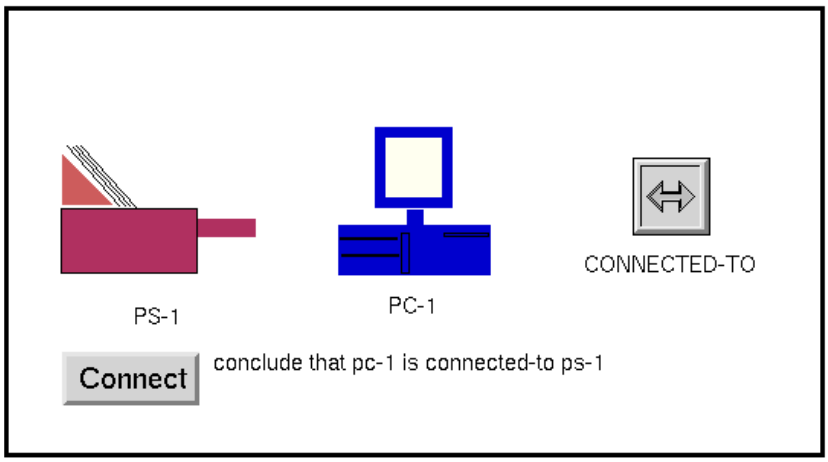
Leaving the relation-is-permanent attribute with its default no value causes relations to be transient: instances of the relation are removed at KB Restart and Reset operations, and are not saved with the KB.

Understanding How G2 Saves Relations

Each time you create or conclude a relation, three items participate in that relation:

- The definition of the relation you are creating.
- The relation source.
- The relation target.

G2 creates source and item relations for every relation you create between two or more items. Consider the next diagram:



Given the connected-to relation definition, if you conclude that pc-1, a PC computer, is connected-to ps-1, a printer, G2 maintains three pieces of knowledge to sustain that relation:

- The connected-to relation definition.
- A relation for pc-1 indicating it is connected-to ps-1.
- A relation for ps-1 indicating that pc-1 is connected to it.

Complying to Permanency

For a relation to persist through a reset of G2, the relation definition, and the items that participate in that relation, must be permanent. G2 establishes a relation even when one or both of the items are transient, but it deletes such relation on reset without posting a message.

For G2 to save relations as permanent knowledge, each item must be permanent and uniquely identifiable during a KB save and subsequent load operation. G2 assures that every item in a G2 process is uniquely identifiable by its uuid attribute which is saved with the KB. However, if you change the value of a related item's uuid you can jeopardize successful reloading of a relation. See Changing a UUID at Load Time.

Restoring Permanent Relations

After successfully saving permanent relations, G2 restores each relation when you load the KB. The inability to restore a previously saved relation is called a **rendezvous failure**. Such failures can occur if G2 is unable to locate one or more of the participating relation items at KB load time.

This is how G2 handles rendezvous failures:

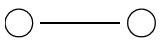
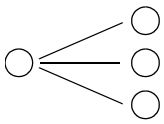
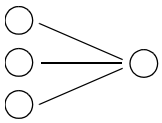
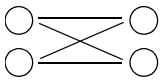
If...	Then G2...
G2 cannot locate the relation definition	Deletes all relations with that name.
The UUID reference of a relation in one item does not correspond to another qualifiable item.	Deletes the missing relation.
One item relates to another, but only one item has a relation, possibly because only selected modules were saved in a KB save operation	Replaces the missing relation automatically.

Note Inter-module item references in permanent relations do not compromise single-module KB saving.

Specifying the Cardinality of Relations

An important property of a relation is its **cardinality**, which specifies how many instances of the relation's first class can be related to how many instances of the relation's second class.

You specify the cardinality of a relation in the type-of-relation attribute, which can be one of the values listed in the following table:

Type-of-Relation Attribute Value	Description
one-to-one 	An instance of the relation's first class can be related to at most one instance of the relation's second class, and an instance of the second class can be related to at most one instance of the first class. For example, each server on a network can have only one backup server, and each backup server can support only one server.
one-to-many 	An instance of the relation's first class can be related to one or more instances of the relation's second class, and an instance of the second class can be related to at most one instance of the first class. For example, one local-area network supports connections to one or more computer nodes, and each computer node can be connected to only one local-area network.
many-to-one 	One or more instances of the relation's first class can be related to at most one instance of the relation's second class, and an instance of the second class can be related to one or more instance of the second class. For example, each of many computer nodes can be connected to one local-area network, and the local-area network supports connections to more than one computer node.
many-to-many 	An instance of either of the relation's classes can be related to one or more instances of the other class. This is the default cardinality. For example, each printer in a local-area network can be the print server for more than one computer node, and each computer node on the network can be served by one or more printers.

As soon as you enter the relation name, the first class, and the second class in the relation's table, G2 displays a description of the possible relations based on the type of relation. If you change the type of relation, G2 updates this description.

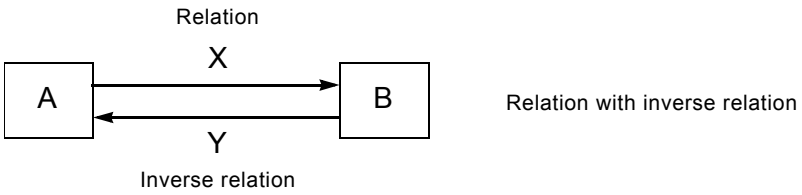
For example, the following description shows how two computers are related based on a many-to-many relation named in-communication-with:

IN-COMMUNICATION-WITH, a relation	
Notes	OK
Authors	ghw (23 May 2000 2:14 p.m.)
Change log	0 entries
Item configuration	none
First class	computer
Second class	computer
Relation name	in-communication-with
Inverse of relation	none
Type of relation	many-to-many
Relation is symmetric	yes
Relation is permanent	no
a computer may be in-communication-with more than one computer; more than one computer may be in-communication-with a computer	

Defining an Inverse Relation

When defining a relation, you can also define an **inverse relation**, which is a relation between a relation target (the second class) and a relation source (the first class). You specify the name of the inverse in the inverse-of-relation attribute.

The following figure shows a relation and an inverse relation between two items, A and B. The relation X is the relation name, and the relation Y is the inverse relation. The arrow between the two items represents the relation.



If you define a relation with an inverse, concluding an instance of that relation also concludes an instance of the inverse relation.

Concluding an inverse relation automatically concludes the relation. For example, if A is the first-class and B is the second-class, you can conclude that A is related to B, or that B is related to A.

For example, the relation previous-linestation-for could specify an inverse relation named next-linestation-for, as this table shows:

For an inverse relation, when you conclude this relation...	G2 also concludes this inverse relation...
grinding-station-1 is the-previous-linestation-for rinsing-station-2	rinsing-station-2 is the-next-linestation-for grinding-station-1

Note If you have specified an inverse relation, you cannot also specify that the relation is symmetrical. The relation-is-symmetric attribute must be no. A symmetrical relation creates its own inverse relation, which has the same name as the relation, as described in Defining a Symmetric Relation.

The inverse of a relation has the inverse cardinality of that relation, as this table shows:

If the relation's cardinality is...	Then its inverse relation is...
one-to-one	one-to-one
one-to-many	many-to-one
many-to-one	one-to-many
many-to-many	many-to-many

To create an inverse relation:

1. Enter any allowable value in the type-of-relation attribute.
2. Use the default value of no for the relation-is-symmetric attribute.
3. Enter a name for the inverse relation in the inverse-of-relation attribute.

Note G2 prevents you from specifying a value for the inverse-of-relation attribute when the relation-is-symmetric attribute is yes.

Defining a Symmetric Relation

A relation can be **symmetric** or not. You specify whether a relation is symmetric in the relation-is-symmetric attribute.

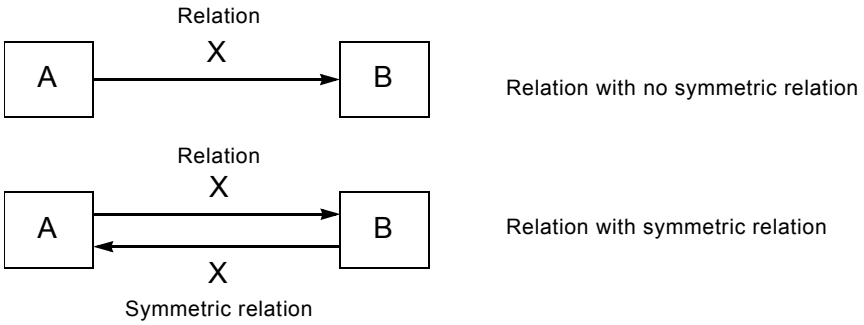
By default, relations are not symmetric; the default value of relation-is-symmetric is no. This means that when you create a relation between two items, by default, G2 creates a single relation between the relation source and the relation target. Further, relation-is-symmetric must be no when creating inverse relations.

If you define a relation to be symmetrical, however, concluding an instance of that relation also concludes an inverse relation of the same name as the relation.

You can also conclude the inverse relation, which automatically concludes the relation. For example, if A is the first-class and B is the second-class, you can conclude that A is related to B, or that B is related to A.

Note For symmetric relations, when concluding the relation or its inverse, the relation source and relation target can be items of the first-class or second-class. For example, A can be an item of the second-class and B can be an item of the first-class, or vice versa.

The following figure shows a relation between two items, A and B, first with no symmetric relation, then with a symmetric relation. The relation X is both the relation name and the inverse relation name of the symmetric relation. The arrow between the two items represents the relation.



For example, a relation between two computers is symmetric when one computer is in-communication-with a second computer, and the second computer is also in-communication-with the first computer:

For a symmetric relation, when you conclude this relation...	G2 also concludes this symmetric inverse relation...
computer-a is in-communication-with computer-b	computer-b is in-communication-with computer-a

Note You can only create symmetric relations between relations whose cardinality is also symmetrical: one-to-one or many-to-many.

To create a symmetric relation:

1. Specify the type-of-relation attribute as either one-to-one or many-to-many.
2. Use the default specification of none in the relation definition's inverse-of-relation attribute.
3. Click on the value of the relation-is-symmetric attribute, and select change to "yes" from the menu.

Note The change to "yes" menu choice is only available when the value of inverse-of-relation is none.

Creating a Relation

You can create a relation between:

- Single items.
- Classes of items.
- A single item and a class of items.
- A class of items and a single item.

Using Conclude to Create Relations

You use a conclude action to create a relation between two relation items or classes.

To conclude a relation value:

➔ conclude that {*item*} is [{not | now}] *relation-name item*

The first *item* serves as the relation source, and the second *item* serves as the relation target. In general, the first *item* returns an instance or class of the first-class, the second *item* returns an instance or class of the second-class.

The exception is a symmetric relation, where the expressions can be items of the first or second class. See the note under Defining a Symmetric Relation.

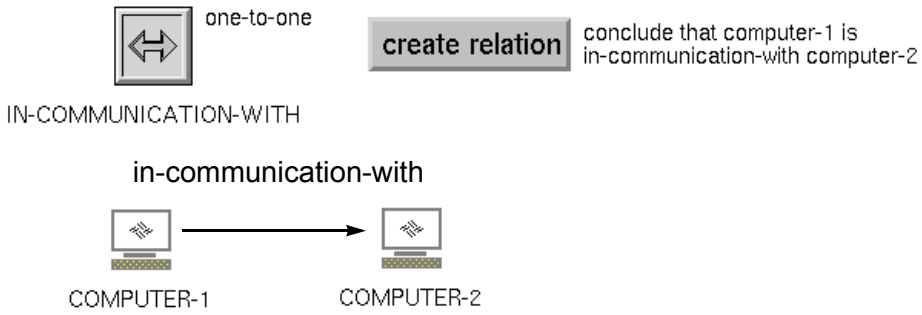
The not statement breaks an existing relation, while the now statement establishes a new relation by breaking an existing one when doing so violates the relation's cardinality. If you do not specify either not or now, you can only conclude a relation between items when it does not violate the relation's cardinality.

You can conclude all types of relations by using this syntax. Here are some examples of concluding different types of relations:

Relation Type	Example of Conclude Statement
one-to-one	conclude that node-1 is not-responding-on network-1
one-to-many	conclude that node-1 is the-server-for every node
many-to-one	conclude that every node is a-part-of network-8
many-to-many	conclude that every node is a-component-of every network

Example of Creating a Relation between Two Items

For example, this action button creates a relation between computer-1 and computer-2, which are both instances of the computer class. In the relation definition, first-class and second-class both specify the computer class.



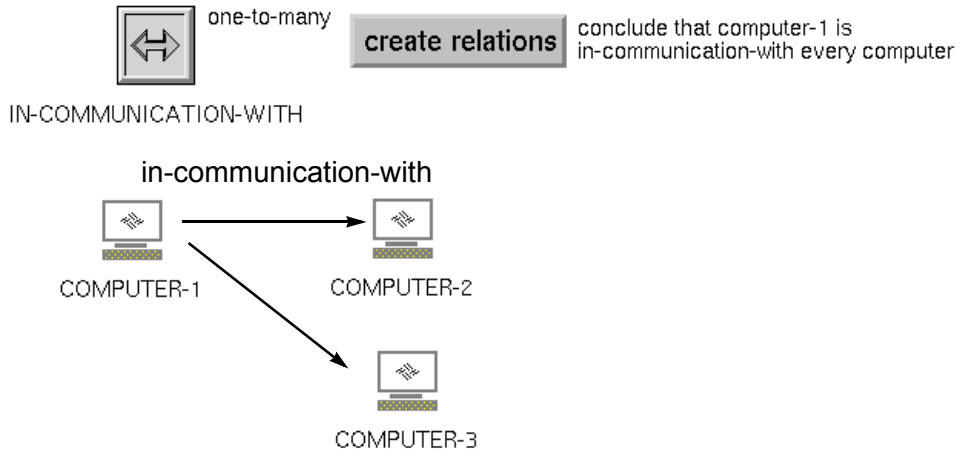
Notice that the relation type is one-to-one. This means that you cannot conclude another relation named in-communication-with between computer-1 and another computer, since doing so would violate the relation's cardinality.

Note If you attempt to conclude a relation between an item that is already participating in a one-to-one relation of the same name, G2 signals an error.

Example of Creating a Relation between an Item and a Class

You can create a relation between more than one instance of a class by using an expression that returns a class instead of an item. (Note that you can also create multiple conclude statements to accomplish the same thing.)

For example, this button creates a relation between computer-1 and every instance of the computer class:



Note In this example, G2 also concludes a relation between computer-1 and itself, because the relation source is also an instance of the second class.

You can only conclude a relation when doing so does not violate the relation definition's cardinality.

Note If you attempt to conclude a relation that violates the relation's cardinality, G2 signals an error.

Tip Use an alternative form of the conclude action to break existing relations before concluding a new relation, when concluding the relation violates the relation's cardinality.

G2 creates, at most, one instance of a particular relation between the same two items. Thus, if you conclude a relation between two items when a relation of that name already exists between the items, G2 does not create another relation.

You *can* conclude multiple instances of different relations between the same two items.

Using a Sequence to Conclude a Relation

You can also conclude relations between items by specifying a sequence value for the relationships virtual attribute of an item.

To use a sequence for creating a relation:

➔ conclude that the relationships of *item-of-interest* = sequence (*relationships*)

where the sequence of *relationships* consists of one or more structures of relation names and the items to which the relation applies. Each structure in *relationships* contains these subattributes:

Subattribute	Type	Description
relation-name-reference	symbol	The name of the relation, which can be either the relation-name or the inverse-of-relation of the relation.
relation-is-inverted	truth-value	An attribute that G2 sets to determine whether the relation is symmetric (true) or not symmetric (false) when concluding relationships.
related-items	sequence	A sequence of items with the relation-name-reference relationship to the *item-of-interest*.

G2 uses the value of the relation-is-inverted subattribute of the relationships virtual attribute to determine the directionality of the relation being set under these conditions:

- The relation is not symmetric.
- The relation does not specify an inverse.
- Both the relation target and the relation source are instances of the first-class of the relation.

Example of Creating a Relation with a Sequence

As an example, a KB includes two relation definitions, married-to and a-daughter-of. Three items exist, george, bill, and edna. Using the conclude action *without* a sequence, you could create two relations for edna:

```
conclude that bill is married-to edna and conclude that
    edna is a-daughter-of george
```

To conclude the same relationships by using a sequence, use an action such as this:

```
conclude that the relationships of edna =
    sequence(
        structure(relation-name-reference: the symbol married-to,
                related-items: sequence(bill)),
        structure(relation-name-reference: the symbol a-daughter-of,
                related-items: sequence (george))))
```

Caution Concluding relations using the relationships virtual attribute *replaces* the item's current relations. Concluding relations using the without-a-sequence grammar adds relations to an item while maintaining the item's current relations.

Removing a Relation

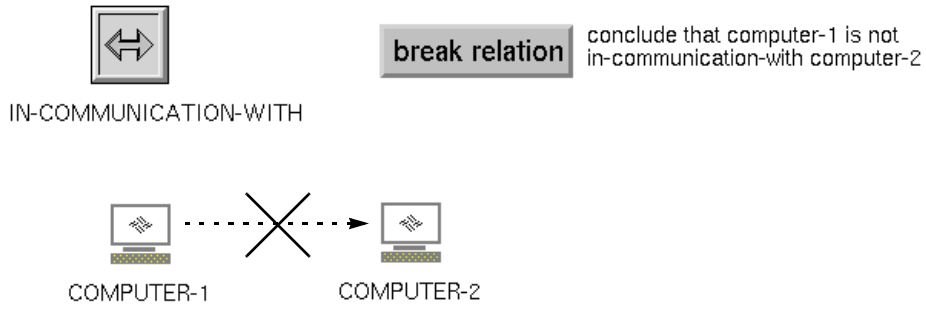
You use a conclude action with the word not to remove a relation instance between two relation items or classes.

To remove a relation:

➔ conclude that *item* is not *relation-name item*

For a description the arguments, see Creating a Relation.

For example, this button breaks a relation between computer-1 and computer-2, which are instances of the computer class. In the relation definition, the first-class and second-class attributes both specify computer class.



Similarly, you can break relations between more than one instance of a class by using a class expression. For example, this statement breaks all relations between every instance of the computer class:

```
conclude that every computer
    is not in-communication-with
    every computer
```

Removing Relations by Deleting Items

If you delete an item that participates in a relation, G2 removes the relation.

Deleting a relation in this manner does *not* cause forward chaining in rules. For more information, see Invoking Rules When a Relation is Deleted.

Replacing a Relation

In general, when concluding a relation, G2 signals an error if you attempt to conclude a relation that violates the relations's cardinality. You can use an alternative form of the conclude action to break existing relations before concluding a new relation.

To replace a relation between items or classes:

➔ conclude that *item* is now *relation-name item*

Using the Now Syntax

When you use the now syntax, G2 determines whether adding the instance of the relation violates the cardinality of a relation. If it does, G2 deletes the conflicting relation when it establishes the new one.

For example, suppose not-responding-on is a one-to-one relation between a node and a network and you conclude the following relation:

```
conclude that node-1 is not-responding-on network-1
```

Then, suppose you conclude the following:

```
conclude that node-1 is now not-responding-on network-2
```

This causes G2 first to delete the not-responding-on relation instance between node-1 and network-1 and then to establish a new not-responding-on relation between node-1 and network-2.

When replacing a one-to-one relation, G2 deletes existing relations in which the relation source or the relation target participate before creating a new relation. When replacing a one-to-many relation, however, G2 only breaks the existing relation between the relation target and another item; it leaves intact any existing

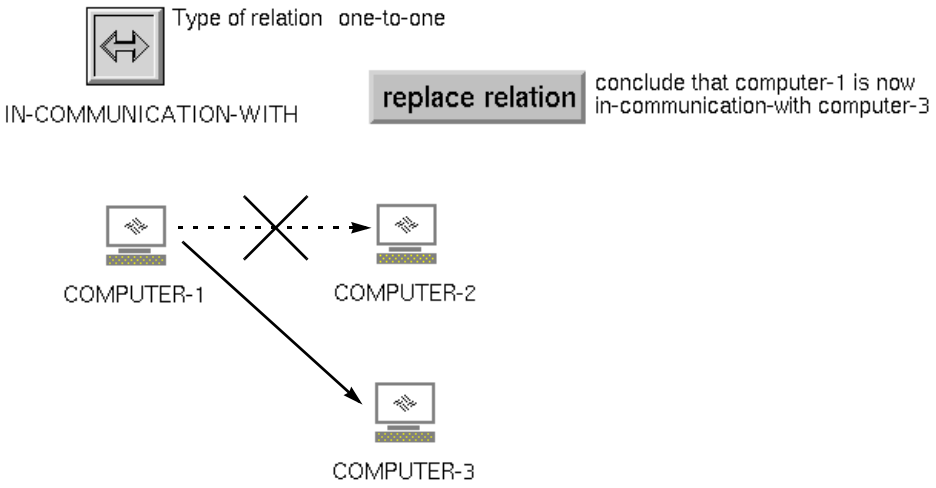
relations between the relation source and another item. G2 does not break any existing relations when replacing a many-to-many relation.

The following table shows how relations are deleted, if necessary, according to the type of relation:

For this type of relation...	G2 deletes the existing relation instance with items of...
one-to-many	The second class
many-to-one	The first class
one-to-one	The first and/or the second class

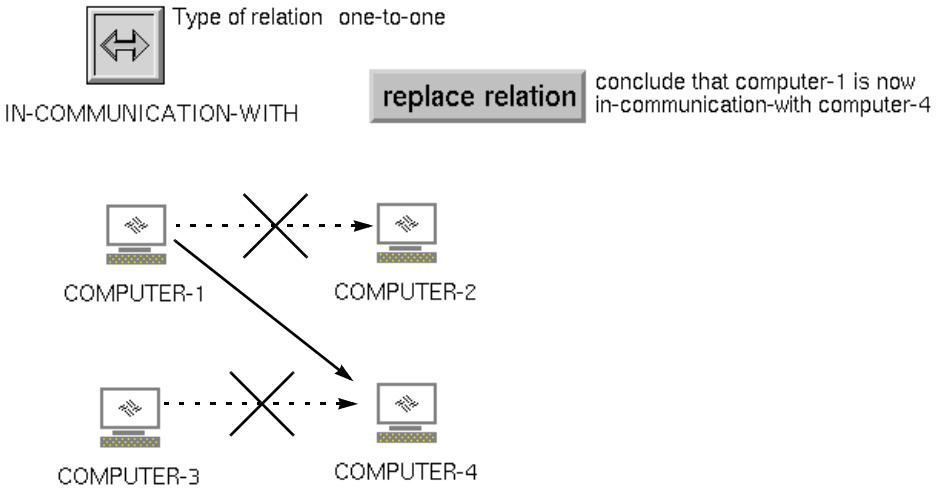
Example of Replacing a One-to-One Relation

For example, assume computer-1 is related to computer-2 by an in-communication-with relation. The following action button breaks the relation between computer-1 and computer-2, and creates a new in-communication-with relation between computer-1 and computer-3. The cardinality of the relation is one-to-one.



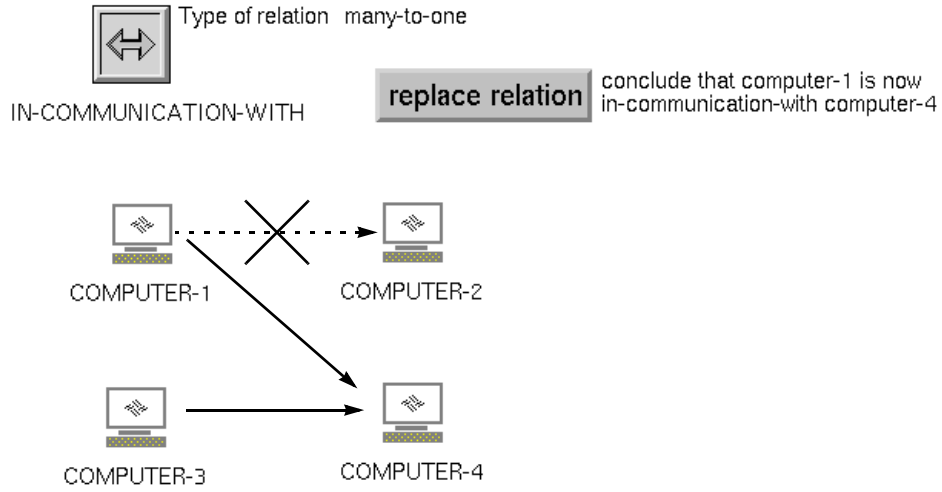
Example of Replacing Multiple One-to-One Relations

Suppose computer-1 is related to computer-2, and computer-3 is related to computer-4, both via an in-communication-with relation. The action button in this figure breaks both existing relations, and creates a new in-communication-with relation between computer-1 and computer-4.



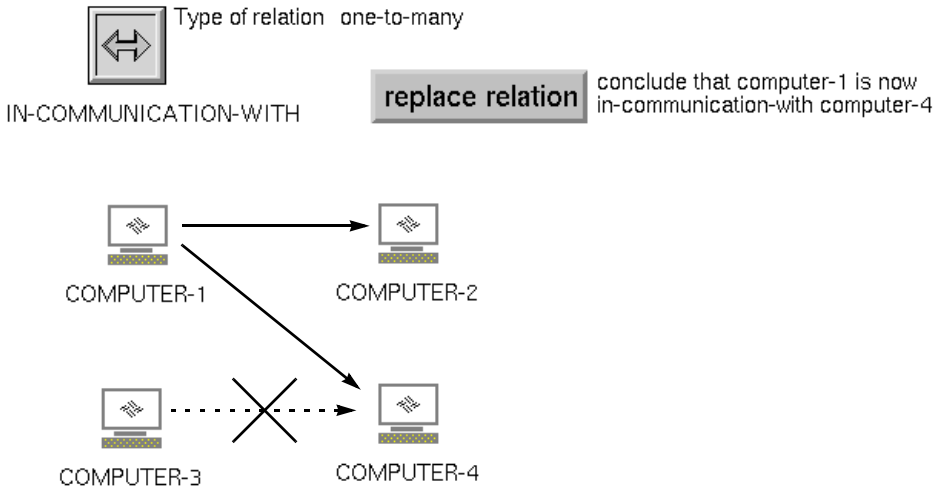
Example of Replacing a Many-to-One Relation

Suppose computer-1 is related to computer-2, and computer-3 is related to computer-4 via a many-to-one relation. The action button in this figure breaks the existing relation between computer-1 and computer-2, and creates a new in-communication-with relation between computer-1 and computer-4. Notice that computer-3 is still related to computer-4 because the relation allows more than one relation source to be related to the same relation target.



Example of Replacing a One-to-Many Relation

Suppose computer-1 is related to computer-2, and computer-3 is related to computer-4 via a one-to-many relation. The action button in this figure breaks the existing relation between computer-3 and computer-4, and creates a new in-communication-with relation between computer-1 and computer-4. That computer-1 is still related to computer-2 because the relation allows for one relation source to be related to more than one relation target.



Note If the computers were related via a many-to-many relation, G2 would not break any existing relations.

Invoking Rules Using Relations

Relations can cause forward chaining to rules under the following circumstances:

- When a rule tests whether a relation is created or deleted.
- When the antecedent of a rule refers to an item that participates in a relation.
- When a generic rule refers to an item that participates in a relation.
- When a generic rule refers to a variable that participates in a relation, and the variable receives a new value or expiration time.

Note When G2 evaluates whether a relation exists between two items, G2 does *not* backward-chain to rules that can establish a relation.

Using Whenever Rules to Detect Relatedness

You can create a whenever rule to detect that two items have become related irrespective of the relation involved.

To detect relatedness:

➔ whenever *item* becomes related to *item*

For example:

```
whenever any plane P becomes related to any runway R
    then start check-takeoff-or-landing (P, R)
```

A whenever rule that detects a relation event fires every time any relation is made. Thus if two items become related by one relation, and then by another, a becomes related to rule fires twice, once for each relation established.

Using Whenever Rules to Detect Cessation of Relations

You can create a whenever rule to detect that two items have ceased to be related irrespective of the relation involved.

To detect cessation of relations:

➔ whenever *item* ceases to be related to *item*

For example:

```
whenever any plane P ceases to be related to departure-schedule
    then conclude that the departure of P is complete
```

Deleting an item that participates in a relation does not invoke any whenever rules checking for a ceases to be related to event, because the item no longer exists.

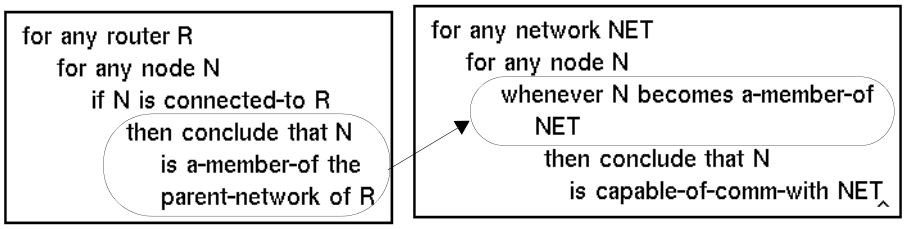
Invoking Rules When a Relation is Created

When you create a relation by using a conclude action, G2 detects this event as part of processing, which causes forward chaining to rules. You test when one or more relations is created in the antecedent of a rule.

To invoke rules when a relation is created:

➔ *item* becomes *relation-name item*

In this example, the generic if rule on the left concludes a new a-member-of relation, if N is connected to R. When G2 detects the creation of this relation, G2 invokes the generic whenever rule on the right.



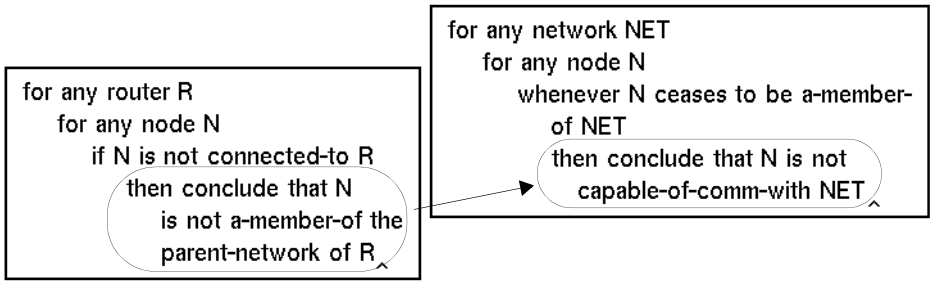
Invoking Rules When a Relation is Deleted

When you delete a relation by using a conclude action with the word not, G2 also detects this event during processing, which also causes forward chaining to rules. You test when one or more relations is deleted in the antecedent of a rule.

To invoke rules when a relation is deleted:

➔ *item* ceases to be *relation-name item*

For example, the generic if rule on the left deletes the relation named a-member-of, which invokes the generic whenever rule on the right:



Note Deleting a relation by deleting an item that is participating in the relation does *not* cause forward chaining to rules.

For a description of whenever rules and the events they can detect, see Whenever Rules.

Invoking Rules That Test Whether a Relation Exists

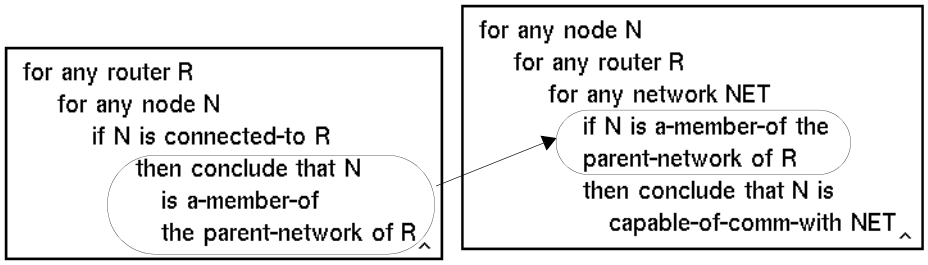
You can test whether a relation exists, in the antecedent of a rule. When G2 detects the creation or deletion of the relation, G2 invokes this rule via forward chaining.

You test for the existence of one or more relations in the antecedent of a rule.

To invoke rules that test whether a relation exists:

➔ if *item* is [not] *relation-name item*
-> *truth-value*

For example, the generic if rule on the left concludes a new a-member-of relation, if N is connected to R. The rule on the right tests for the existence of this relation. G2 invokes this rule when the relation is created.



To test whether N is not related to the parent-network of R, you would use:

```
if N is not a-member-of the parent-network of R
```

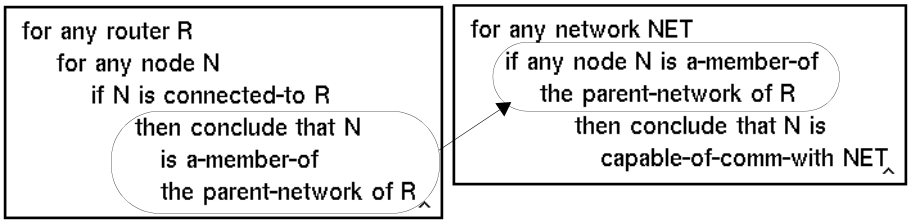
Invoking Rules That Refer to Items with Relations

You can use an expression in the antecedent of a rule to refer generically to a class of items that participate in a relation. When G2 detects the creation or deletion of the relation, G2 invokes this rule via forward chaining. You refer generically to a class of items that are related to one or more items in the antecedent of a rule.

To invoke rules that refer to items with relations:

➔ for any *class-name* [*local-name*] that is *relation-name item*

In this example, the rule on the left concludes the a-member-of relation, which triggers the rule on the right that refers generically to any node that is a-member-of the parent network of the router.

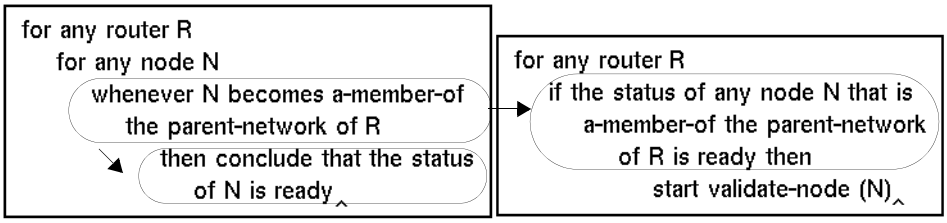


Invoking Rules That Refer to Variables with Relations

You can use an expression in the antecedent of a rule to refer generically to a variable that participates in a relation. When the variable receives a value or an expiration time, G2 invokes this rule via forward chaining.

You refer to variables that are related to items by using the same syntax as outlined in Invoking Rules That Refer to Items with Relations.

For example, the rule on the left concludes a value for the variable N. router via the relation named a-member-of, where node is a subclass of the system-defined symbolic variable class. When the relation is created, the rule concludes a value for the variable, which invokes the rule on the right that refers generically to the variable that is participating in the relation.



Working with Transient Items

Some relations are transient knowledge in your KB; thus, resetting the current KB deletes all relations, and saving the current KB does not save these relations. However, saving the current KB into a KB snapshot file does save all the KB's relations. See Saving Permanent and Transient Data in Snapshot KBs for more information about KB snapshot files.

You can establish, break, and replace relations based on:

- A relation definition that is either permanent or transient.
- A relation source or relation target that is either permanent or transient.

For information on how to use an action to change the permanent/transient status of a relation definition, relation source, or relation target, see make.

Working with Deactivated and Disabled Items

You can establish, break, and replace relations that are based on:

- A relation definition that is deactivated or disabled.
- A relation source or relation target that is activated or enabled; you *cannot* establish, break, or replace a relation when a relation source or item is deactivated or enabled.

If a relation definition, relation source, or relation target becomes deactivated or disabled while existing relations exist, G2 does not break the relation. However, you cannot use an expression to refer to deactivated or disabled items that participate in relations.

Updating Relations While a KB is Running

You can change a relation definition while the KB is running, paused, or reset. In general, when G2 is not reset, changing a relation definition changes any instances of that relation definition accordingly. For example, if you change the name of a relation definition while relations of that name exist, G2 renames the relations.

Note We do not recommend making significant changes to relation definitions while your KB is running and while relations exist, because G2's short-term performance can be significantly degraded.

The following sections describe special precautions that you should take when making changes to relation definitions while your KB is not reset and while existing relations exist.

Updating the First Class and Second Class

If you update the first-class or second-class attributes of a relation definition, G2 preserves existing relations based on the new class, as long as the new class is a superior class of the existing class; otherwise, it may delete existing relations if an item conflicts with the new class.

Updating the Type of Relation

If you change the cardinality of a relation to be more restrictive, for example, changing a many-to-one relation to a one-to-one relation, G2 removes all relation instances that conflict with the new relation type, leaving only one. G2 chooses the remaining relation arbitrarily from the original set.

Updating Symmetric Relations

For relations whose relation-is-symmetric attribute is yes, the effect on existing relations of changing the first-class, second-class, type-of-relation, and relation-is-symmetric attributes can be more complex, depending on the classes of items that you specify.

If the items participating in the relation are subclasses of both the first class and the second class, and you change the definition such that G2 must delete one or more relations, G2 arbitrarily determines which item is the relation source and which item is the relation target.

Updating Relations While Executing Procedures

If a statement or action in a procedure refers to a relation, and you make a change to the relation definition while the procedure is executing that statement, the procedure continues to execute using the existing relation. Any changes to the relation are not visible to the procedure's execution.

For example, given a for statement in a procedure that refers to a set of items based on their relation to another item, if you change the definition of that relation while the loop is processing, G2 neither adds to nor removes from the definition of the set of items.

Similarly, if a local name in a procedure refers to an item that participates in an instance of a relation, and you change the definition of the relation while the procedure is executing, G2 uses the existing value of the local name.

Updating a Relation While a Rule is Executing

If a rule refers to a relation, and you change its relation definition while the rule is executing, the rule continues executing using the existing relation.

Note Changing the definition of a relation can cause G2 to remove relations. For each relation that is removed for this reason, G2 generates a ceases to be related event. In turn, generating these events causes G2 to invoke whenever rules that refer to these events. For more information, see Invoking Rules Using Relations.

Updating a Relation When Saving a KB Snapshot File

After you begin saving the current KB to a KB snapshot file (by invoking the g2-snapshot system procedure), if you change a relation definition, G2 first writes any relations affected by the change to the file before changing relations in the current KB.

Expressions Involving Relations

G2 provides the following expressions involving relations.

Event Expressions

To detects the creation or deletion of a relation:

➔ *item* {becomes | ceases to be} *relation-name item*

For information on using this expression to forward chaining to rules, see Invoking Rules When a Relation is Created and Invoking Rules When a Relation is Deleted.

Logical Expressions

You can test for the existence of a relation to produce a truth-value that indicates whether one item participates in a relation of the specified kind with another item.

To test for the existence of a relation with a logical expression:

➔ *item* is [not] *relation-name item*
-> *truth-value*

For example:

```
if my-item is not aligned-with my-message then
    conclude that the status of my-item is unaligned
```

For information on using this expression to forward chain to rules, see Invoking Rules That Test Whether a Relation Exists.

Relation Participation Expressions

Two expressions return information about relation participation as:

- The relationships in which an item participates.
- The items participating in a relation.

Obtaining the Relationships of an Item

Items include a hidden attribute, relationships. When you refer to the this attribute, G2 returns a sequence of structures describing the relations in which the given item is participating.

To refer to the relationships of an item:

➔ the relationships of *item-of-interest*
-> *relationships*

Argument	Description
item-of-interest	The name of the item whose relations you are testing.

Return Value	Description
relationships	A sequence of structures, each consisting of these subattributes: • relation-name-reference • relation-is-inverted • related-items

If the *item-of-interest* is not participating in any relationships, the expression returns the empty sequence, sequence(). For a complete description of this expression, see Referring to the Relationships of an Item.

Obtaining Items Participating in a Relation

You can refer to a relation's items-in-this-relation hidden attribute. G2 returns a sequence of the items participating in the relation.

To refer to the items participating in a relation:

➔ the items-in-this-relation of *relation-of-interest*
-> *participating-items*

Argument	Description
relation-of-interest	The name of the relation in which items are participating.

Return Value	Description
participating-items	A sequence of symbols, each naming an item participating in the *relation-of-interest*.

If no items are participating in the given relation, G2 returns the symbol none.

As an example of using this expression, in a KB with a married-to relation definition, you could create relations between four items with this conclude action:

```
conclude that bill is married-to edna and conclude that george is married-to janet
```

Such an action results in four items participating in the married-to relation:

- bill
- edna
- george
- janet

A reference to:

```
the items-in-this-relation of married-to
```

returns this sequence of symbols:

```
sequence(JANET, EDNA, GEORGE, BILL)
```

Generic Item References

You can reference the class of items that is related to any item or class.

To reference generic items by their relations:

➔ the *class-name* [*local-name*] that is *relation-name item*
-> *item*

With the the quantifier, this generic reference expression produces the one and only item of the specified class (or any of its subclasses) that participates in a relation of the specified kind with the specified item. With the any quantifier, this expression produces the set of items of the specified class (or any of its subclasses)

that participate in relations of the specified kind with the specified item. For example:

```
if the status of any mixing-vat that is controlled-by vat-controller-1 is hot
    then ...
```

For information on using this generic reference to forward chain to rules, see Invoking Rules That Refer to Items with Relations and Invoking Rules That Refer to Variables with Relations.

The Relation Class

These are the class-specific attributes of the relation class.

Attribute	Description
first-class	Name of the class of items that participate as the relation source. For example, if a tank is part-of a subsystem, then tank is the first class.
Allowable values:	Any class name
Default value:	item
second-class	Name of the class of items that participate as the relation target. For example, if a tank is part-of a subsystem, then subsystem is the second class.
Allowable values:	Any class name
Default value:	item
relation-name	Name of the relation.
Allowable values:	Any valid symbol that is unique within the KB
Default value:	
inverse-of-relation	Name of an automatically defined relation that is the inverse of relation-name. When a relation defines an inverse, concluding the relation also concludes the inverse relation.
Allowable values:	Any valid symbol that is unique within the KB

Attribute	Description
Default value:	none
Notes:	You can enter an inverse relation only when the relation is not symmetric.
type-of-relation	Cardinality of the relation.
Allowable values:	one-to-one one-to-many many-to-one many-to-many
Default value:	many-to-many
relation-is-symmetric	Whether G2 creates an inverse relation of the same name as the relation between items of the second class and items of the first class.
Allowable values:	yes no
Default value:	no
Notes:	If a relation is symmetric, G2 requires that the inverse-of-relation attribute contains the value none (that is, the relation name and inverse relation name are the same).
relation-is-permanent	Determines whether G2 will save the relation through a KB restart and reset operation. For more information, see Using Permanent Relations.
Allowable values:	yes no
Default value:	no

Describing the Items That Participate in a Relation

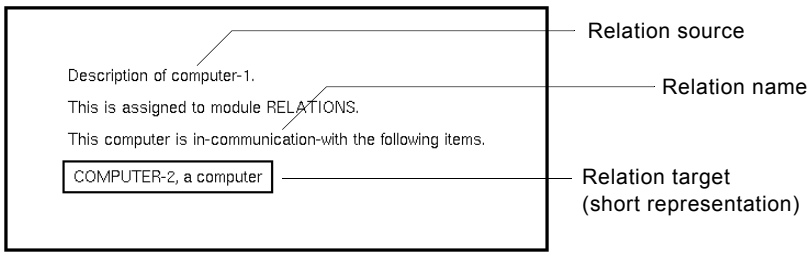
You use the Describe facility to display the relations in which an item participates. You can describe both the relation source and the relation target.

To describe a relation source:

➔ Display the relation source's menu and select describe.

For example, suppose computer-1 is related to computer-2 via a non-symmetric in-communication-with relation.

Describing computer-1 displays the following temporary workspace. G2 displays the relation target using a short description provided by the Describe facility:



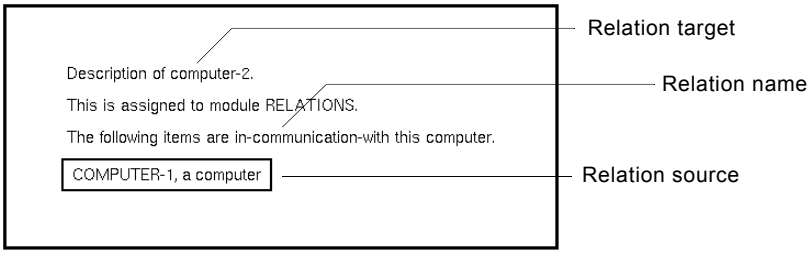
To describe the relation target:

➔ Click on the short description of the relation target in the Describe workspace, and select describe.

or

➔ Click on the relation target on the workspace and select describe.

Describing computer-2 displays the following temporary workspace. Notice that the description does not show an inverse relation for the relation target. The description simply shows the relation in which computer-2 participates.



Part IV

Computational Capabilities

Chapter 20: Actions

Describes each G2 action and shows you how to use it.

Chapter 21: Expressions

Describes the purpose and syntax of each G2 expression.

Chapter 22: Procedures

Shows how to define, customize, and use G2 procedures.

Chapter 23: Methods

Shows how to define and use G2 methods.

Chapter 24: Rules, Inferencing, and Chaining

Describes how G2 invokes rules to perform actions.

Chapter 25: Formulas

Describes generic and specific formulas and their use.

Chapter 26: Text Parsing and Manipulation

Describes capabilities for manipulating text and substrings, parsing and tokenizing text using regular expressions, and interconverting text between the Gensym and Unicode character sets.

Chapter 27: XML Parsing

Describes how to parse XML code and make callbacks to user-defined procedures.

Chapter 28: Functions

Lists system-defined functions and describes how to create new functions.

Chapter 29: Publish/Subscribe Facility

Describes how to use the publish/subscribe facility for event subcription.

Chapter 30: G2 Graphical Language (G2GL)

Describes G2GL, a graphical language for describing processes.

20

Actions

Describes each G2 action and shows you how to use it.

Introduction

An **action** is a task that G2 executes. Actions often provide programmatic equivalents to interactive tasks, such as printing a workspace, pausing the knowledge base (KB), or changing the value of an item's attribute.

Some actions require items to be transient. The effects of most actions performed on permanent KB items revert when you reset the KB, unless you save the change with a make permanent action. For example, if you rotate an icon, resetting the KB causes the icon to revert to its previous position.

The effects of the following three actions persist when you reset the KB:

- conclude, to provide a new attribute value.
- show, to show an item or workspace.
- hide, to hide an item or workspace.

You can use actions in rules, procedures, user menu choices, and action buttons. This example shows a series of actions: create, conclude, transfer, and make:

```
create a class-definition CD;
conclude that the class-name of CD = the symbol musical-selection;
conclude that the direct-superior-classes of CD =
     sequence(the symbol object);
transfer CD to this workspace;
make CD permanent
```

Executing Actions

An action executes when control reaches it during execution of a procedure, rule, action button, or user menu choice. When more than one action appears in one of these contexts, the actions can execute sequentially or in parallel.

For **sequential execution** of a series of actions, G2 performs each statement in the order in which it appears. Thus the following three statements, executed sequentially, swap the values of var1 and var2:

```
conclude that var0 = var1;
conclude that var1 = var2;
conclude that var2 = var0
```

For **parallel execution** of a series of actions, G2 first evaluates the arguments of every action, and then executes each action. Thus, the result of executing these actions in parallel:

```
conclude that var1 = var2;
conclude that var2 = var1
```

is to swap the values, rather than to assign the value of var2 to both var1 and var2, which would happen if G2 executed the same statements sequentially.

The order in which G2 executes actions in parallel after evaluating their arguments is not predictable: G2 may order their execution as needed to optimize scheduling.

Executing Actions in Procedures

In a procedure, sequential execution is the default. Actions in a series therefore execute sequentially unless they appear in a do in parallel statement, as described under do in parallel.

Executing Actions in Other Contexts

In rules, action buttons, and user menu choices, parallel execution is the default. Actions in a series therefore execute in parallel unless they are qualified by an in order clause, as described under Specifying Sequential Execution.

Executing Iterative Actions

Sequential or parallel execution can also occur in the execution of a single action. If an action iterates over a class of items, such as:

```
conclude that the pressure-status of every tire T is full
```

G2 finds each item to which the action can be applied (in this case, every tire) and executes the action for each item it finds. If the action exists in a context where sequential execution applies, G2 applies the actions to the items sequentially. If the context specifies parallel execution, G2 applies the actions to all of the items in parallel.

For more information about parallel and sequential execution within rules, see Understanding Rule Invocation and Execution.

Further Information

The principles that govern the use of actions in rules apply to their use generally, as described in Rules, Inferencing, and Chaining. For further information about specifying actions, see in that chapter:

- Coding the Consequent.
- Executing Actions in the Consequent in Parallel.
- Executing Actions in the Consequent Sequentially.

Dictionary of Actions

The rest of this chapter lists all actions in alphabetical order, and provides complete information about each one.

abort

This action aborts a procedure or a procedure invocation. A procedure is an item. A procedure invocation is a transient item that exists when G2 executes any procedure whose class-of-procedure-invocation attribute has the value procedure-invocation, or any subclass of that class.

To abort a procedure or procedure invocation:

➔ abort {*procedure* | *method* | *procedure-invocation* | *method-invocation*}

This statement illustrates how to abort a specific procedure invocation. The symbol filling is the name of a user-defined relation.

```
abort the procedure-invocation that is filling gas-tank-1
```

The next statement illustrates how to abort a procedure using its name:

```
abort fill-gas-tank
```

For a description of procedures and procedure invocations, see Procedures.

Aborting a Procedure or Method

To abort a procedure or method:

➔ abort *procedure*

where *procedure* is one of:

- The name of a procedure.
- A generic reference to a procedure, expressed with the or every.
- A qualified method name.

You can use the statement abort this procedure only within a procedure or method. When you abort a procedure, G2 aborts all invocations of that procedure.

Aborting a Procedure or Method Invocation

To abort a procedure or method invocation:

➔ abort *procedure-invocation*

where *procedure-invocation* is a statement referring to a procedure-invocation in one of several ways, including every. When you abort a procedure-invocation, G2 aborts only the particular invocation. For example, you can refer to a procedure-invocation from within a procedure or method as this procedure-invocation, or you can specifically identify the procedure-invocation.

If you reference the method by using the *class::method* syntax, invoking the abort action that way aborts all invocations of the method.

To refer to a procedure-invocation, the class-of-procedure-invocation attribute of the procedure or method must have the value procedure-invocation.

activate

This action activates an activatable subworkspace. For the subworkspace of an item to be activatable, the superior item must include this configuration statement in its item-configuration attribute:

```
declare properties as follows : activatable-subworkspace
```

When you start or restart a KB, G2 does not activate activatable subworkspaces: they remain inactive until they are activated with the activate action. For more information about the effects of activating a subworkspace, see How Activating and Deactivating Affects Items.

To activate a subworkspace:

➔ activate the subworkspace of *item*

where *item* refers to:

- Any item whose class definition includes the configuration statement in its instance-configuration attribute.
- Any item, capable of having a subworkspace, whose item-configuration attribute includes the configuration statement.

An example is:

```
activate the subworkspace of every volvo
```

Note Disabling an activated subworkspace also deactivates it. Subsequently enabling the subworkspace leaves it deactivated.

When you activate a subworkspace, G2 invokes any initially rules residing upon the subworkspace and then starts scanning rules on the subworkspace after all initially rules are complete. You cannot activate a subworkspace if its superior item is:

- Disabled, by selecting Disable from its menu.
- On a workspace that has been disabled, by selecting Disable from the KB Workspace menu.
- On an inactive subworkspace.

When you try to do so, G2 signals an error and does not activate the subworkspace.

change

This action changes these properties of items:

- An array or list value element.
- The length of an array.
- The color attribute of an item.
- The name of an item.
- The size of an item.
- The text of an attribute.
- The text of a procedure, statement, free-text, or message.

The change action does not make permanent changes; the effects of the actions are undone when you reset G2. In some cases, you can make these changes permanent by using the make permanent action, as described under make.

You can also make many permanent changes by using the conclude action, as described under conclude, in conjunction with attribute access, as described in Attribute Access Facility.

Changing List and Array Elements

Using the change action with lists and arrays is described in Using Other List and Array Expressions.

Changing the Color Attribute of an Item

To change the color of an item:

➔ change the *color-attribute-name* of *item* to {*color-name* | *symbolic-expression*}
change the *color-pattern* of *item* so that
{*color-attribute-name* is *color-name*} [, ...]

Element	Description
color-name	Any color or metacolor.
symbolic-expression	Any expression evaluating to a color name.
color-attribute-name	background-color border-color foreground-color icon-color stripe-color text-color

The color attribute you specify varies with the type of item. For example, only connections have a stripe-color color attribute. Workspaces and messages have a background-color, while messages and textual items, such as rules, have a text-color.

Two versions of this action exist. The first lets you change a single color-attribute, the second, change the color-pattern of, lets you change more than one color-attribute of an item in one statement.

These examples show you how to change a single color-attribute, and then two attributes, using the color-pattern of statement:

```
change the border-color of warning-message to red

change the color-pattern of subws-of-volvo so that background-color is wheat,
    foreground-color is brown
```

If you try to change multiple icon regions by using the change the color-pattern action, and a specified icon region does not exist, G2 signals an error.

You can provide a symbol of the form RGB*rrggbb* as a valid color name, where *rr*, *gg*, *bb*, are the 8-bit hex values for red, green, and blue. For details, see Other Literal Terms.

You can obtain the various color attributes of any item with color operations systems procedures. For a description of these, see the *G2 System Procedures Reference Manual*.

Changing the Icon Color Region of Instances

The change the icon-color of action changes any region of a system- or user-defined region called icon-color.

All system-defined icons have multiple named regions, including icon-color. You can use the color operations system procedures to get a list of all region names and their corresponding colors, and then change them. The color operations system procedures are described in the *G2 System Procedures Reference Manual*.

To change an icon's icon-color region:

➔ change the icon-color of *item* to *color*

Using this action has different results depending on how the icon is defined:

If the icon-description of the class includes...	Then the action has this effect...
A region called icon-color	Changes only the color of the icon-color region.
At least one color set to the foreground metacolor	Changes the binding of the foreground metacolor to the new color. Subsequently adding an icon-color region causes any foreground color to revert to its previous binding.
No foreground color and no icon-color region	None.

Changing the Color of Any Named Icon Region

You can change the color of any named icon region. Icon regions are described in Defining Regions.

To change the color of an icon region:

➔ change the *region-name* icon-color of *item* to *color-name*

Use this action to change any icon region *not* called icon-color. This example changes the region called inner-circle of the item instance circle1 to green:

```
change the inner-circle icon-color of cirle1 to green
```

You can provide a symbol of the form RGB*rrggbb* as a valid color name, where *rr*, *gg*, *bb*, are the 8-bit hex values for red, green, and blue. For details, see Other Literal Terms.

Use the color operations system procedures, described in the *G2 System Procedures Reference Manual* to change multiple icon regions programmatically.

Changing the Name of an Item

Use this action to give an item a new names attribute value and to replace any existing values.

You can use the change the name of action to generate names for transient items.

To change the name of an item:

➔ change the name of *item* to *symbolic-expression*

An example is:

```
change the name of volvo to the symbol volvo-model-x
```

Note The names attribute requires a symbolic value, so you must precede the item name using the symbol statement.

You can also use attribute access to reference and change most attribute values, as described in Attribute Access Facility.

Minimizing the Size of a Workspace

Use this action to shrink wrap a workspace, which you refer to with any *kb-workspace* expression.

To change the size of a workspace to its minimum size:

➔ change the size of *kb-workspace* to minimum

An example is:

```
change the size of subws-of-volvo to minimum
```

Changing the Text of an Attribute

Note In most cases, the attribute access facility supersedes the use of the change the text of action. However, all such actions continue to be supported. Attribute access is described in Attribute Access Facility.

Use this action to change the value of any user-defined attribute and most system-defined attributes that you would otherwise change interactively with the Text Editor. Unlike text-editor and conclude-action changes, the change the text of action is not permanent unless it is followed by a make permanent action. Without the make permanent action, the attribute will revert to its previous permanent value when G2 is reset, *even when the* change the text of *action is followed by a text-editor or conclude-action change*.

Note Changing the text of a rule deactivates the rule and then reactivates it. The deactivation of the rule causes it to be removed from item arrays and lists.

To change the text of an item's attribute:

➔ change the text of the *attribute* of *item* to *quoted-message*

Enter the *quoted-message* with surrounding quotation marks (""). If G2 detects an invalid value for the changed attribute, G2 signals an error.

This example changes the class name of an class-definition, and makes the name change permanent:

```
change the text of the class-name of auto-1 to "automobile";
make automobile permanent
```

This example transiently changes the scan-interval attribute of a rule:

```
change the text of the scan-interval of check-loss-rule to "10 seconds"
```

Note You cannot use this action to change the attribute values of tabular functions.

To make permanent changes to user-defined value attributes, use the conclude action, described on Concluding Attribute Values. For information about changing class definitions while G2 is running, see Changing Definitions.

Changing the Text of Textual Items

Use this change the text of action any time you wish to change any textual item programmatically.

To change the text of a procedure, statement, free-text, or message:

➔ change the text of {*procedure* | *statement* | *free-text* | *message*}
to *text-expression*

This action changes the text of a procedure, free-text, message, or statement (including generic formulas, rules, function definitions, units of measure, and remote procedure and foreign function declarations).

An example of changing the text of a message is:

```
change the text of overfull-warning to "The gas tank is full"
```

To include quotes within the new text, precede each quote with the @ character. In general, the @ sign is used to escape a special character.

conclude

Use the conclude action to change the value of:

- Arrays and lists.
- Attributes.
- Icon variables.
- Variables and parameters.
- Relations.

Unlike the change action, using conclude makes a permanent change to an attribute value.

Concluding Array and List Elements

Using the conclude action for list and array elements is described in Lists and Arrays.

Concluding Attribute Values

You can change the values of user-defined and most system-defined attributes that contain values, variables, or parameters. Such changes are permanent and are not undone when you reset G2.

Concluding Values for Attributes

To conclude a new value for an attribute:

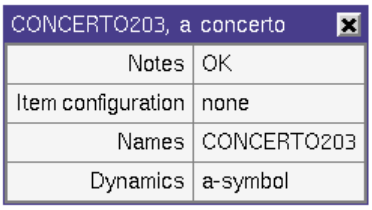
➔ conclude that the *attribute* of *item* =
{ = *value-expression* | is *symbolic-expression* }

This action changes the value of the *attribute-name* of *item*. For attributes with a structure or sequence type, you can use the corresponding function with the conclude action to produce a new attribute value.

You can conclude a symbolic value using either is or =, the difference being that an equals (=) expression evaluates the symbol, and an is expression is equivalent to the statement: = the symbol *symbol-name*. The difference between these two constructs can produce very different results.

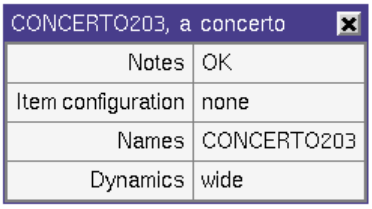
Consider this procedure, which accepts a symbol as its argument. Using the is a-symbol statement changes the value of the dynamics attribute of concerto203 item to the symbol a-symbol, not to the value of a-symbol.

```
TestSymbol1(a-symbol: symbol)
    begin
        conclude that the dynamics of
        concerto203 is a-symbol
    end
```



Changing the procedure to use the equals (=) operand changes the value of the dynamics attribute to the value passed to the procedure, in this example, the symbol wide:

```
TestSymbol1(a-symbol: symbol)
    begin
        conclude that the dynamics of
        concerto203 = a-symbol
    end
```



This example illustrates concluding values for attributes with float, symbolic, and truth values:

```
conclude that the temperature of values-test = 100.2;
conclude that the heat-range of values-test is hot;
conclude that the is-it-hot of values-test is true
```

This change is permanent. Use change the text of attribute actions for transient changes.

Using an Indirect Reference to Conclude an Attribute Value

To conclude a new value for a user-defined attribute by using an indirect attribute reference:

➔ conclude that the {*class-name* | *type* that is an attribute of *item* named by *symbolic-expression* { = *value-expression* | is *symbolic-expression* }

This action changes the value of a user-defined attribute, which is specified by the attribute-name returned by the *symbolic-expression*, as shown in the next example:

```
conclude that the quantity that is an attribute of values-test
    named by gas-tank-level = 250
```

Concluding Icon Variables

To change the icon-variables of an item:

➔ conclude that the icon-variables of *item* ...

For example:

```
conclude that the icon-variables of person =
    structure (width: 30, heigth: 120);
```

For a detailed description of using variables in icons, see The Icon Editor and Icon Management.

Concluding Variable and Parameter Values

Using the conclude action to change the values of variables and parameters is described in Variables and Parameters.

Concluding Relations

Using the conclude action to change relations is described fully in Relations.

create

This action creates a transient connection or item. You can create an item by:

- Naming a particular class.
- Cloning an existing item.
- Specifying a class named by a symbolic expression.

Whenever you create a transient item, it exists within the KB, but does not appear on a workspace until you use the transfer action to place it on a workspace. The item remains transient until or unless you make it permanent with the make permanent action. Transient items are deleted by G2 when the KB is reset. For a description of transient and permanent items, see Understanding the Knowledge Contained in Items.

Creating Transient Connections

Creating transient connections is described in Creating Transient Connections.

Creating an Item of a Particular Class

To create an item of a particular class:

➔ create {a | an} *class-name* [*local-name*]

This action creates an instance of the class-name you specify. You can optionally follow the class name with a local name in the statement. An example is:

```
create an automobile AM and transfer AM to this workspace at (100, 100);
change the name of AM to the symbol transient-auto
```

Creating an Item by Cloning Another

To create an item by cloning an existing item:

➔ create {a | an} *class-name* {*local-name*} by cloning *item*

This create action clones an existing item to create a new one. This action is similar to the clone menu choice, described in Cloning an Item, except that using the menu choice creates a permanent item, while using the create action creates a transient one. Cloning an item also clones its subworkspace and all items and their attributes on the subworkspace, but does *not* clone any relations. An example of creating by cloning is:

```
create a kb-workspace KS by cloning change-action-ws;
show KS at half scale;
change the name of KS to the symbol cloned-action-ws
```

Creating an Instance of a Class by Using an Indirect Reference

To create an item whose class is determined from an expression:

➔ create an instance of the class named by *symbolic-expression*

In this example, class-name is a the symbolic-expression that names a class:

```
create-transient-item(class-name: symbol)
I: class item;
begin
    create an instance I of the class named by class-name;
    transfer I to new-item-ws at (100, 100)
end
```

deactivate

This action deactivates the subworkspaces of items that are activatable.

To deactivate a subworkspace:

➔ deactivate the subworkspace of *item*

For a description of activatable subworkspaces, see activate.

When you deactivate the subworkspace of an item, G2 behaves as though any items upon the subworkspace do not exist. All items upon the subworkspace are no longer active. The subworkspace itself, however, can still be referenced and tested with the expression:

kb-workspace has [not] been activated

For more information about the effects of activating a subworkspace, see How Activating and Deactivating Affects Items. An example of deactivating a subworkspace is:

```
deactivate the subworkspace of every automobile
```

delete

You can delete transient or permanent items.

To delete a transient or a permanent item:

➔ delete *item* {without permanence checks} {removing connection stubs}

You cannot delete a permanent item without first making it transient, unless you use the without permanence checks clause.

Deleting Items without First Making Them Transient

To delete permanent items without permanent checks, optionally removing connection stubs:

➔ delete *item* without permanence checks {removing connection stubs}

Using the delete action in this way lets you delete permanent items programmatically, completely ignoring all transient restrictions. Deleting items by using the without permanence checks is the programmatic equivalent of the interactive delete menu choice.

Note Using the delete action omitting the without permanence checks statement requires that items be made transient prior to deletion. For a discussion about making items transient, see Making Permanent Items Transient.

When the delete action executes, any rule that is waiting for a value and that refers to a deleted item (or to one of its attributes, its subworkspace, or to any item on its subworkspace) is cancelled. If a delete action is part of a rule, the rule is not cancelled, but any parallel or subsequent sequential action in the rule that refers to the deleted item causes G2 to signal an error.

A procedure that referenced the item before it was deleted continues to execute, but if it references the deleted item, G2 signals an error.

Removing Stubs While Deleting an Item

You can remove connection stubs from a remaining item when deleting an item to which another is connected.

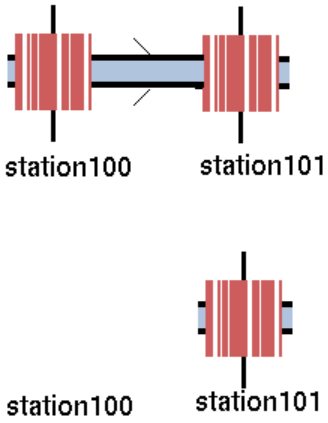
To remove stubs when deleting an item connected to another:

➔ delete *item* removing connection stubs {without permanence checks}

For example:

```
delete filling-auto-1 removing connection stubs without permanence checks
```

The top grouping in this example shows two connected objects station100 and station 101, and the bottom grouping shows the result of deleting station100 with the delete action using the removing connection stubs expression:



Deleting Connections

To delete a connection:

➔ delete {*connection* [without permanence checks]
[removing connection stubs] }

You can delete a transient connection or a permanent one by using the without permanence checks grammar. Deleting a connection does not delete the stubs, unless you use the removing connection stubs statement with the delete action, as in the next example:

```
delete the connection at the output end of k11 without permanence checks
    removing connection stubs
```

focus

This action invokes all rules that have a specified object in their focal-objects attribute, or it invokes all rules that have an object's class or superior class specified in their focal-classes attribute. The focal-objects and the focal-classes attributes of all rules let you specify the object or classes, respectively, to which a focus action applies.

To focus on an object:

➔ focus on {*object* | *object-class*}, awaiting completion

When a focus action is invoked, such as the action in this example, G2 invokes all the rules that have capacity specified in their focal-objects attribute or all rules that have the class or a superior class of capacity specified as focal-classes.

```
whenever maximum-volume receives a value and
    when maximum-volume > 200 then focus on capacity
```

You cannot focus directly on a class to focus on all instances at one time. If you need to do that, write a generic rule with this structure:

for any *class* such that *qualification* if *antecedent* then focus on the *class*

For a description of the , awaiting completion option, see Waiting for Rules to Complete When Invoked from a Procedure.

halt

This action stops G2 from running the KB. Use this action for debugging.

To enter a halt action:

➔ halt [with *text-expression*] if breakpoints are enabled

The halt action lets you halt at a certain location in a procedure, rule, or action button. Using the halt action with a *text-expression* displays the message when the halt action is executed.

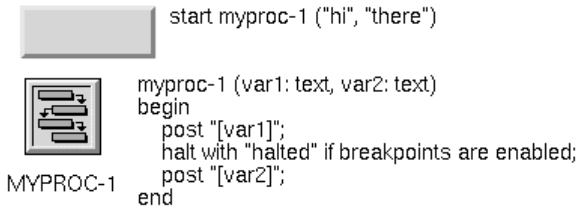
The halt action is a debugging tool that lets you pause at a certain point of execution. As such, you may not want to use the halt action when G2 is controlling a real-time application.

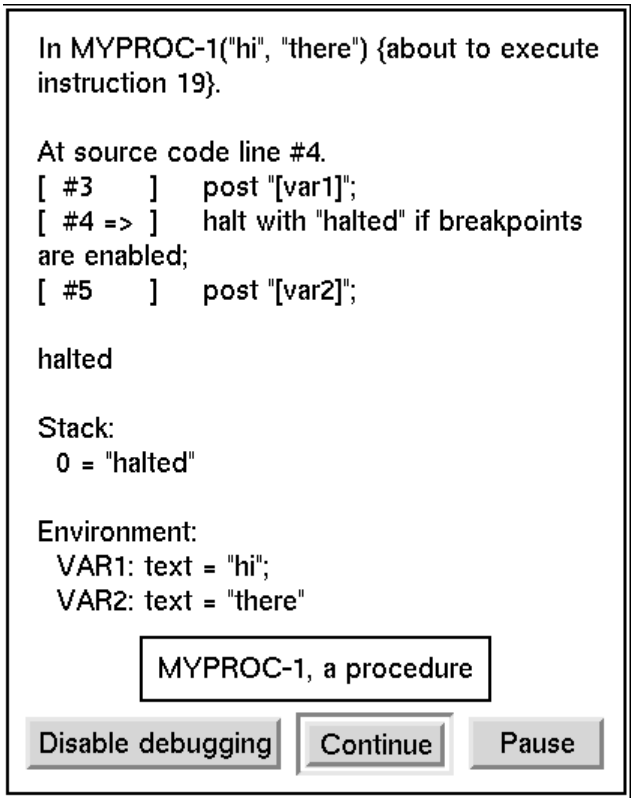
Note The uninterrupted-procedure-execution-limit of a procedure applies even when using the halt action. Therefore, using the halt action may cause the procedure to time out.

You can use the halt action only when the tracing-and-breakpoints-enabled? attribute in the Debugging Parameters system table is set to yes. The attribute default is no. The breakpoint-level of the Debugging Parameters need not be set to a level greater than zero (0).

You can use the halt action to display the procedure invocation hierarchy by setting the show-procedure-invocation-hierarchy-at-pause-from-breakpoint of the Debugging Parameters system table to true.

A dialog such as the following appears in the server when G2 executes the halt action. The dialog shows the line number of the source code that G2 is about to execute and the surrounding source code with line numbers. G2 cannot continue processing until you select one of the buttons.





hide

The hide action lets you hide one of these items:

- The workspace upon which a given item resides.
- The workspace of the superior item of a given subworkspace.
- The subworkspace of a given item.
- The workspace upon which an action button or a user menu choice resides.
- A workspace in a given G2 window.

To use the hide action:

➔ hide {*item* |
{ the workspace of | the item superior to | the subworkspace of *item*} |
this workspace} [on g2-window]

Hiding an Item

To hide any item by hiding its workspace:

➔ hide *item*

If the specified *item* does not exist, G2 signals an error.

Hiding the Workspace Containing an Item

To hide the workspace that contains the specified item:

➔ hide the workspace of *item*

If the specified *item* does not exist, G2 signals an error.

Hiding the Workspace of a Superior Item of a Subworkspace

To hide the item that is the superior of the specified subworkspace:

➔ hide the item superior to *kb-workspace*

This action hides the workspace upon which the superior item to the given workspace item resides. If the subworkspace of that item is visible, it remains so. If the specified *kb-workspace* does not have a superior item, G2 signals an error.

Hiding the Subworkspace of an Item

To hide the subworkspace of a specified item:

➔ hide the subworkspace of *item*

Hiding the Workspace of an Action Button or User Menu Choice

To hide the workspace upon which the action button or user menu choice that invokes the action resides:

➔ hide this workspace

Hiding Workspaces on any G2 Window

You can use each of the hide actions to operate on any G2 Window. Following are examples of each version of the hide action, where kmm-window is a named g2-window of a Telewindows client, and each of the action buttons would be invoked from the G2 server.

hide the workspace of cp1 on kmm-window

hide the item superior to the-subws-of-cp1 on kmm-window

hide the subworkspace of cp1 on kmm-window

hide this workspace on kmm-window

inform

This action sends a text message to:

- The operator, which, by default, refers to the message board.
- An item that has an external message facility.
- A workspace.
- Any other specified destination.

The inform action lets you send a text message to a single destination. If the log-inform-messages? attribute of the Logbook Parameters system table is yes, all inform messages are posted to the Operator Logbook as well as to their inform destination.

Using Inform to Post to the Message Board

To use inform to send a message to the message board:

➔ inform the *destination* [for the next time-interval] that *text-expression*

Using this syntax, *destination* is typically the operator. However, specifying the *destination* as an item is acceptable and will also post a message to the message board as in this example:

```
inform the connection-post upon this workspace
  that "This message will reach the message board!"
```

Referring to an item as the *destination* with the inform action permits the use of the upon this workspace statement. This contrasts with the use of the on this workspace statement to refer to a specific destination item, as described in Informing a Destination Item.

This action displays the specified *text-expression* on the message board. You can optionally specify a time interval to display the message. Doing so deletes the message after the specified time interval, but leaves the message board displayed. The next example shows an inform message that is displayed for 30 seconds:

```
inform the operator for the next 30 seconds that
    "[the name of tanker] is filling gas-pump [name-of-pump] now"
```

To use post to send a message to the message board:

➔ post [for the next *time-interval*] *text-expression*

The shorter post action syntax is functionally equivalent to:

inform the operator [for the next *time-interval*] that *text-expression*.

See post for information on the post action.

Informing a Destination Item

To use inform to send a message to a destination item:

➔ inform item [on *kb-workspace* [below | above] *item*]
[for the next *time-interval*] that *text-expression*

This action lets you specify the *item* as an inform message destination. The destination can be any item, an object with an external message capability, such as gsi-message-service, or a workspace. For information on how to use the inform action for G2-to-G2 data exchange, see Examples of Remote Data Service.

You can display a message next to an item on a particular workspace, using an expression such as:

```
inform connection-post-9 on CP-WS that "This is a warning."
```

To display a message next to an item on a particular workspace, you can specify the workspace itself as the destination and use the on *kb-workspace* statement, which specifies the name of a workspace in the KB. G2 displays the message on that workspace, optionally positioned below or above an item you specify. The next example shows how to specify a workspace as the destination for the message:

```
inform start-action-ws on start-action-ws below check-if-showing
    that "here's a message for an item"
```

You can optionally indicate a length of time for G2 to display the message. If you do not specify a time interval, the length of time that the message appears is determined either by the validity-interval attribute of the antecedent if this action is in a rule, or the minimum-display-interval attribute of the Message Board Parameters system table.

If you specify an object with an external message capability, G2 ignores the on *kb-workspace* and any above or below item statements.

The above *item* or below *item* statement specifies that G2 displays the message either below or above the specified *item*. You can only use this statement in conjunction with the on *kb-workspace* statement. If the specified item is not on the workspace, G2 signals an error and does not display the message.

G2 uses the following attributes from the Message Board Parameters system table when displaying messages:

- spacing-between-entries
- maximum-number-or-entries
- highlight-new-messages?
- minimum-display-interval

For a description of these attributes, see Message Board Parameters.

The next example displays a message for 10 seconds below the msg-obj, upon the inform-ws workspace. Messages displayed for a time-interval are transient messages that G2 deletes from the KB after the elapsed time period.

```
inform inform-ws on inform-ws below message-object for the next 10 seconds
    that "The filling task is complete."
```

insert

This action inserts an element into a list:

- At an element location.
- At the beginning or end of a list.
- Before or after an existing list element.

To insert an element into a list:

➔ insert *item-or-value* {at the {beginning | end} of} |
{ {before | after} *item-or-value* in | element *integer-expression* of} *g2-list*

For a complete description and examples of using the insert action, see Lists and Arrays.

invoke

This action invokes a category of rules for all objects, or for an object specified in the categories attribute of the rule.

To enter an invoke action:

➔ invoke *rule-category-name* [{, | or} *rule-category-name*]... rules
for {*object* | *object-class*}, awaiting completion

This action invokes all rules in the specified *rule-category-name*. You specify the category in the categories attribute of a rule. When G2 invokes all rules from a certain category, it uses this attribute to determine what rules are in the category.

Using the optional for *object* statement narrows the search for applicable rules. You can specify a class of appropriate objects for a rule in its focal-objects attribute. When an invoke action includes the for *object* statement, G2 invokes only those rules that include the corresponding rules-category and focal-objects values. Following is an example of two invoke actions:

```
invoke safety rules

invoke safety rules for filling-tank-4
```

The first example invokes all rules that specify safety in their category attribute.

The second invokes all rules that specify safety in their category attribute *and* have filling-tank-4 as a focal-object, or its class or one of its superior classes as the value of focal-classes.

For a description of the , awaiting completion option, see Waiting for Rules to Complete When Invoked from a Procedure.

make

Use the make action for three different purposes:

- Making transient items permanent.
- Making permanent items transient.
- Making a transient workspace the subworkspace of an item.

Making Transient Items Permanent

To enter the make permanent action:

➔ make *item* permanent

This action makes a transient item, its attributes, its connections, and its subworkspace (and all their items, attributes and connections) permanent. Making a workspace permanent makes all items upon it permanent.

Once an item is permanent, G2 does not delete it when you reset your knowledge base. G2 retains the current state of the item including rotation, color, and location. G2 does not save the values of variables and parameters, or the elements of arrays and lists, unless their elements are permanent. For more information about permanent arrays and lists, see Lists and Arrays.

There are two restrictions to making items exist permanently:

- If a transient item does not reside on a workspace, or resides on one that is transient, you cannot make such an item permanent. An exception to this is any object that is an attribute of another object. An attribute object does not reside on a workspace, but making its holding object permanent also makes permanent the attribute object.
- You cannot make a connection permanent if it connects to something that is transient.

You can use the make permanent action on a permanent item to make a transient change permanent. For example, changing the array-length of a permanent array and subsequently using a make permanent action on that array retains the array length.

Here are examples of the make permanent action, one making a transient item permanent, and the other making transient changes to a permanent item permanent.

```
create a kb-workspace WS and show WS at (50, 50) in the screen;
change the name of WS to the symbol new-ws;
make WS permanent
```

```
length = the number of elements in name-list;
change the array-length of name-array to length;
make name-array permanent
```

Making Permanent Items Transient

You can make a permanent item transient so that you can transfer it programmatically.

Tip You do not have to make items transient to delete them. Permanent items can be deleted programmatically as described in Deleting Items without First Making Them Transient.

To enter the make transient action:

➔ make *item* transient

This action makes a permanent item, its attributes, its connections, and its subworkspace (and all items, their attributes and connections on its subworkspace), or a workspace itself transient.

When you make an item transient, all of its connections become transient. When you make a workspace transient, all items upon it become transient.

The next example makes a parameter transient, creates a new workspace, and transfers the parameter to the new workspace:

```
gds-transfer-parameter(WS: class kb-workspace)
NewWS: class kb-workspace;
    begin
        if there exists a parameter P upon WS then
            begin
                make P transient;
                create a kb-workspace NewWS;
                show NewWS at the left center of the screen;
                change the name of NewWS to the symbol Parameter-WS;
                make newWS permanent;
                transfer P to NewWS;
                make P permanent;
                post "[the name of P] has been transferred to Parameter-WS";
            end
        else post "No parameter exists upon [the name of WS]."
    end
```

Limitations to Transiency

An item cannot be made transient when any one of these conditions is true:

1. The item is a class definition for an instantiated or subclassed class.
2. The workspace hierarchy of the item contains a class definition for an instantiated or subclassed class. (When an item is made transient, all of the items within its workspace hierarchy are transient, too.)
3. The item is an attribute value.

Making a Workspace the Subworkspace of an Item

To make a workspace the subworkspace of an item:

➔ make *kb-workspace* the subworkspace of *item*

This action makes the specified *kb-workspace* a subworkspace of *item*. A workspace must be transient to become the subworkspace of an item. If the workspace is already an item's subworkspace, this action automatically changes its association with the original item and makes it the subworkspace of the target item. The transient action cannot be used to accomplish these tasks.

If the target item already has a subworkspace, you must remove that subworkspace before executing this action by either transferring the subworkspace to another item, or deleting it.

The next example makes a subworkspace transient, makes it the subworkspace of an item, and makes it permanent:

```
make subws-of-volvo transient;
make subws-of-volvo the subworkspace of name-list;
make subws-of-volvo permanent
```

Creating a Subworkspace Programmatically

Making a transient workspace the subworkspace of an item is part of creating a subworkspace programmatically. First, use the create action to create a workspace item, and then make it the subworkspace of the desired item.

Activation Status of Subworkspaces

You can enable or disable all workspaces and subworkspaces. Additionally, some subworkspaces are activatable, as described under activate. By default, workspaces and subworkspaces are enabled when G2 starts, while activatable subworkspaces are deactivated, and must be activated programmatically.

While the enabled and activated statuses of a subworkspace are separate properties, they can have similar results. For example, when G2 starts, it invokes every initially rule upon enabled workspaces and subworkspaces. Activating an activatable subworkspace at any time invokes any initially rules that reside upon it.

Making a workspace the subworkspace of another item usually has no effect on its enabled or activation status. An exception to this is making an activatable subworkspace, which is currently deactivated, the subworkspace of an item that is not configured to support activatable subworkspaces. Since the subworkspaces of items are enabled by default, executing such an action enables the previously deactivated subworkspace, which is akin to activating it.

move

This action changes the position of an icon on a workspace.

To move an icon:

➔ move *item* {to | by} (*x*, *y*)

Using the move action with a by statement moves the icon by the amount specified by (*x, y*) from its current position.

Using move with a to statement moves the icon to a new position specified by (*x-workspace-units-right, y-workspace-units-up*) relative to the center of the workspace.

A workspace unit is equivalent to one pixel when a workspace is full size. As you reduce the size of a workspace, the size of each workspace unit also reduces. Specifying a negative number for x or y moves the icon to the left or down, respectively.

The next examples illustrate the use of the item-x-position and item-y-position of other icons as a starting place for the icon being moved:

```
move cart by
    (the item-x-position of installation + 20,
    the item-y-position of installation + 20)
```

You cannot use the move action on workspaces or on items that you are moving manually, nor can you move an icon beyond the edge of a workspace; the object stops at the border. Using the move action can cause event updating, invoking whenever rules.

pause

This action pauses the knowledge base.

To enter the pause action:

➔ pause knowledge-base

This action is the same as the Pause menu choice. You can use the pause action anywhere in a procedure or as the last action of a rule. If you use the pause action in a procedure, when you resume running, execution continues at the next procedure statement. An example is:

```
when number-of-parts >= the array-length of array-of-parts
    then pause knowledge-base
```

This example shows how you can use the pause action to debug your knowledge base during development.

post

The post action sends a message to the message board. It supplies shorter syntax for message-board posting than the inform action does; but, unlike the inform action, it does not send messages to non-message-board destinations. See inform for more information on inform syntax and functionality.

To use post to send a message to the message board:

➔ post [for the next *time-interval*] *text-expression*

The shorter post action syntax is functionally equivalent to:

inform the operator [for the next *time-interval*] that *text-expression*

An example is:

```
post "task 450 has completed"
```

print

This action prints a workspace.

To enter the print action:

➔ print *kb-workspace*

The *kb-workspace* specifies the name of the workspace to print. You can use the statement this workspace if you are issuing the action from an action button or a user menu choice.

A workspace does not have to be visible for you to print it. The print action uses the formats specified in the Printer Setup system table. An example is:

```
print this workspace
```

remove

This action removes an element from a list by specifying an element, and *item-or-value* or a position, and list-type.

To remove an element from a list:

➔ remove {element *integer-expression* | *item-or-value* | {the { first | last } *type*} from *g2-list*

For a complete description and examples of using the remove action, see Lists and Arrays.

reset

This action resets a knowledge base.

To enter the reset action:

➔ reset knowledge-base

This action is the same as the Reset menu choice. When G2 resets a knowledge base, it:

- Stops the knowledge base.
- Reinitializes all variables and parameters.
- Returns all icons to their initial positions.
- Restores the default colors to all items.
- Deletes any transient items.
- Removes list elements, unless the elements are permanent.
- Reinitializes array elements, unless the elements are permanent.
- Removes any relations between items, unless the relations are permanent.
- Reverts any modifications made using the change action that were not subsequently made permanent by the make permanent action.

Use this action any time you want to reset the KB programmatically. For example, in a factory environment, you may want each operator to reset the knowledge base when each shift ends. To do so, you can place the following action button on each shift operator's top-level workspace:

```
in order
    post for the next 30 seconds that
        "[shift] has ended. Resetting knowledge base."
    and reset knowledge-base
```

rotate

This action rotates an icon in 90 degree increments.

To use the rotate action:

➔ rotate *item* {by | to the heading} *quantity-expression* degrees

Rotating an Icon

To rotate an icon:

➔ rotate *item* by {0 | 90 | 180 | 270} degrees

This action rotates the icon of *item* the specified degrees clockwise from its current degree of rotation. Selecting this action repeatedly causes the icon to continue rotating clockwise by the specified degree of rotation. An example is:

```
rotate every g2-window upon this workspace by 90 degrees
```

Entering a value other than those specified causes G2 to round the number to the closest allowable value. For example, if you enter 40 degrees, G2 rounds that value down to 0 degrees; entering 50 degrees causes G2 to round the value up to 90 degrees.

Some items, such as charts and dials, do not permit rotation. G2 signals an error if you try to rotate an item that cannot be rotated.

Rotating an Icon from its Vertical Position

This action rotates an icon clockwise a specified number of degrees from its original upright position, which G2 refers to as its heading. The number of degrees you specify can be 0, 90, 180, or 270. Use the to the heading clause to rotate an icon regardless of its current rotation.

To rotate an icon from its upright vertical position:

➔ rotate *item* to the heading {0 | 90 | 180 | 270} degrees

An example is:

```
rotate every g2-window upon this workspace to the heading 180 degrees
```

Note Using the rotate action can cause event updating, such as invoking whenever rules that test for icon rotation.

---

set

This action assigns a new value to a G2 Gateway variable or a simulation variable.

To assign a new value to a gsi-variable:

➔ set {*gsi-variable* | *simulation-variable*} to *value-expression*

The G2 Simulator is a superseded capability, so using set for a *simulation-variable* is not documented here. For more information, see Appendix F, Superseded Practices.

This action also assigns a new value to a variable in a KB running remotely from a KB running locally. See Examples of Remote Data Service for information.

Setting the Value of a Gsi-Variable

To assign a value to a gsi-variable:

➔ set *gsi-variable* to *value-expression*

This action provides the specified *gsi-variable* with a value. A gsi-variable is subclass of a logical-, quantitative-, float-, integer-, symbolic- or text-variable that includes gsi-data-service as one of its direct superior classes. The G2 Gateway data server must be active to set the *gsi-variable* to a *value-expression*.

The next example initially sets the value of the set point attribute of pi-controller-1, a symbolic gsi-variable in the external application, to on:

```
initially set the set-point of pi-controller-1 to the symbol on
```

Comparing Set with Conclude

Using the set action for a gsi-variable differs from using the conclude action. Concluding a value to a gsi-variable:

- Updates the last-recorded-value attribute of the variable in G2.
- Does not transmit the value to G2 Gateway.

Using the set action to change a gsi-variable value:

- Transmits the value to G2 Gateway.
- Does not change the last-recorded-value attribute of the gsi-variable unless the G2 Gateway that is data serving the variable returns the value to G2.

show

This action changes the way a workspace is displayed. You can use this action to change these characteristics of a workspace:

- The window it is displayed on.
- Its window position.
- Its scale.
- Its layering position in the drawing hierarchy.

To show a workspace:

➔ show {*kb-workspace* | *item*}
[[
{on *window* at *scale* } |
{on *window* at (*x*, *y*) in the screen } |
{at the *window-location* of the screen} |
{scaled by *numeric-expression* } |
{scaled by its current scale times {*numeric-expression* | (*x*, *y*) } } |
{with focal point (*x*, *y*) at (*x*, *y*) in the screen} |
{with its *workspace-location* at the *window-location* of the screen}]
[preserving workspace layering]]

Element	Description
window	An optional window specification. You can show the workspace or item on a particular window.
workspace-location	A location in reference to some part of a workspace, which can be: top left corner | top right corner | top center | left center | center | right center | bottom left corner | bottom-center | bottom-right corner
window-location	The location of the workspace in the G2 window which can be: top center | top left corner | top right corner | left center | center | right center | bottom center | bottom left corner | bottom right corner

Note Some of the show action statements deal with a workspace origin or extent, or its relationship to G2's window. To familiarize yourself with these terms and concepts, see Positioning Items upon a Workspace.

A show action that does not specify preserving workspace layering always displays the specified workspace at the top of all other workspaces currently being displayed.

The preserving workspace layering phrase can be optionally specified only when a scaling or positioning detail is also specified. Specifying preserving workspace layering causes the workspace to be rescaled or repositioned without changing its current layering position.

Note When viewing workspaces in Telewindows, the show action on minimized workspace views automatically restores the workspace view if the action also lifts the workspace to the top. For example, show ws-1 restores ws-1 if the workspace is minimized, but show ws-1. . . preserving workspace layering does not. Similarly, entering go to ws-1 in Inspect for a minimized workspace view restores the view, but Control + - does not.

The show action has many capabilities, some of which can be used together in complex ways. The following sections describe the different ways to reference a workspace, and demonstrate the scaling and positional variations of the show action. In most cases, you can combine statements to include multiple options in one action. For example, this statement combines the basic show action with workspace scaling and positional options:

```
show show-action-ws scaled by its current scale times 1.5 with its top left
    corner 10 units to the right of the center of the screen
```

Showing a Workspace without Changing its Scale or Position

This section describes show actions that do not include scaling or positioning specifications. These actions result in the workspace being drawn on the top layer of the drawing hierarchy at its current scale and window position. If the workspace is already on the top layer, these actions do nothing. The statements demonstrate the various ways of referencing the workspace to be shown.

To show a workspace by referencing the workspace itself or an item that resides on the workspace:

➔ show {*kb-workspace* | *item*}

To show the workspace or subworkspace of an item:

➔ show the {workspace | subworkspace} of *item*

Notice that show the workspace of *item* and show *item* both display the workspace the item resides on.

To show a workspace through indirect item references:

➔ show the item superior to {*subworkspace* | *object*}

The action show the item superior to *subworkspace* will display the workspace of the item that has the subworkspace.

The object in show the item superior to *object* should be the value of a user-defined attribute of an item. The action will display the workspace of the parent item.

Showing a Workspace at a Different Scale or Window Position

When the show action includes scaling or positioning details, you can add the optional preserving workspace layering specification and the workspace will be rescaled or repositioned on its current layering position rather than being drawn on the top layer. Depending on the scale and position of the other workspaces in the window, such an action may obscure the workspace.

You can use the layering syntax to create dynamic workplace displays that avoid the flickering effect of alternately shown and hidden workspaces. If you omit the layering syntax, the workspace will be displayed at the top layer of the drawing hierarchy.

The examples below show the optional layering syntax at the end of each statement.

Showing a Workspace at a Fixed Scale

To display a workspace at a system-defined fixed scale:

➔ show {*kb-workspace* | *item*} {at *scale*} [preserving workspace layering]

where *scale* can be one-quarter scale, half scale, three-quarter scale, or full scale.

Showing a Workspace at an Arbitrary Scale

To display a workspace at an arbitrary scale based on a factor of the workspace's current scale:

➔ show {*kb-workspace* | *item*} scaled by its current scale
times *quantity-expression* [preserving workspace layering]

where *quantity-expression* can be any float value. When a workspace is at full size, you can scale it up to four times its current scale.

Showing a Workspace with Different Horizontal and Vertical Axes Scaling Factors

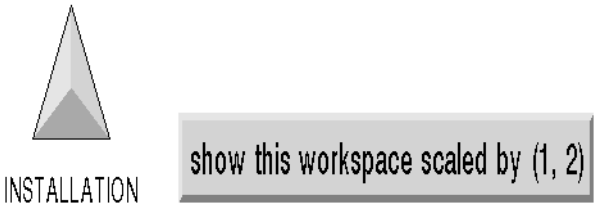
To use different scaling factors for the horizontal and vertical axes of the workspace:

➔ show {*kb-workspace* | *item*} scaled by its current scale
times (x, y) [preserving workspace layering]

This action scales the workspace by the scaling factors of x and y.

Whenever a workspace is scaled, G2 scales everything upon the workspace proportionately. Showing a workspace with different horizontal and vertical scaling factors can produce disproportional effects, which are especially noticeable with text elements such as item names, labels, and messages.

The next diagram shows the label of an action button upon a workspace scaled by its current width and two times its height:



Note Changing the scale of a workspace can change the location of its origin within the G2 window.

There are three available scaling options for the show command.

This example scales a workspace a fixed amount:

```
show inform-ws at three-quarter scale
```

This example scales a workspace an arbitrary amount:

```
show inform-ws scaled by its current scale times 2.5
```

This example scales a workspace by specifying x-y scales:

```
show hide-ws scaled by (1.5, 2.0)
```

Showing a Workspace at a Specific Screen Location

To show the workspace at a certain location:

➔ show {*kb-workspace* | *item*} at (*x*, *y*) in the screen
[preserving workspace layering]

➔ show {*kb-workspace* | *item*} at the *window-location* of the screen
[preserving workspace layering]

The first action shows the workspace at the specified *x* and *y* integer values, which determine the horizontal and vertical positions in workspace units within the G2 window.

Alternatively, you can specify a *window-location*. The next examples illustrate both variations of the show action using a specific screen location.

```
show hide-ws at (100, 100) in the screen

show hide-ws at the left center of the screen
```

Showing a Workspace by Positioning its Edges

To show a workspace using a part of the workspace as a reference point in the G2 window:

➔ show {*kb-workspace* | *item*} with its *workspace-location* at the *window-location*
of the screen [preserving workspace layering]

This action positions the workspace at the *window-location* of the G2 window, which can be center, left, right, top, or bottom.

The next examples show the workspace in the center or right center of the G2 window, using the bottom left corner, or bottom center of the workspace, as a reference point.

```
show hide-ws with its bottom left corner at the center of the screen

show hide-ws with its bottom center at the right center of the screen
```

Showing a Workspace Using a Focal Point

To use a focal point upon the extent of the workspace to position the workspace at a specific location within the G2 window:

➔ show {*kb-workspace* | *item*} with focal point (*x*, *y*) at (*x*, *y*) in the screen
[preserving workspace layering]

The *x* and *y* are the horizontal and vertical locations, respectively, provided in workspace units. The first pair of numbers provide a reference point in the extent of the workspace, and the second pair provide a location within the G2 window.

For example, you may wish to use a particular item upon a workspace as the focal point to position that workspace within the G2 window. The next example uses an item's item-x-position and item-y-position values as the focal point upon the

workspace, and then positions that workspace focal point at a specific G2 window location.

```
show hide-ws with focal point
    (the item-x-position of cart1, the item-y-position of cart1)
        at (-200, -300) in the screen)
```

Ensuring that the Workspace is Always Visible

To ensure that a portion of the workspace is always visible, use the g2-ui-show-workspace system procedure. For details, see User Interface Operations.

shut down g2

This action automatically shuts down G2 and returns control to the operating system.

To enter the shut down action:

➔ shut down g2

When this action executes, G2 aborts all executing tasks and procedures, closes all open files and any Telewindows connections, and shuts down.

This action is similar to the Shut Down G2 menu choice, but the action does not display a button that asks you to confirm the shutdown. Here is an example of a shut down command:

```
post for the next 20 seconds that "G2 is shutting down now"
wait for 20 seconds;
shut down g2
```

start

This action invokes a procedure asynchronously:

- With arguments or without.
- Locally or on a remote G2 or G2 Gateway process.
- Optionally with a priority or after a time interval.

Use the start action to start a procedure for asynchronous processing. The start action schedules the specified procedure for execution; processing then continues through any remaining statements within the calling rule or procedure. At some later time, the scheduler will actually invoke the started procedure.

When starting a remote procedure, because of this scheduling order, it is possible for the remote procedure to complete before the remainder of the calling rule or procedure. To invoke a procedure synchronously from a procedure, use the call statement, as described under call.

You cannot use the start action to obtain a return value from a procedure: any value returned by a started procedure is discarded. Use start with caution for operations such as writing to a file: by the time the started procedure executes, the data it was to write may no longer exist, or the stream it was to write to may no longer be open.

A started procedure cannot assume that the context that existed when it was scheduled still exists. Any amount of processing may have occurred between its scheduling and its invocation by the scheduler. Once the procedure starts, it can assume that nothing else in the KB will execute unless the procedure enters a wait state, as described under Allowing Other Processing.

Starting a Procedure

To start a procedure:

➔ start *procedure* ([*argument* [, ...])

This action starts *procedure*, which is a procedure that requires arguments or does not. Separate all arguments with commas (,).

This example shows an action that starts a procedure, using the this window statement to pass the current g2-window as the single argument of the procedure.

```
start check-if-showing(this window)
```

Starting a Procedure on a Remote G2 Process

To start a procedure on a remote G2 or G2 Gateway process:

➔ start *procedure* ([*argument* [, ...]) across {*g2-to-g2-interface* | *gsi-interface*}

This example starts *procedure* on another G2 process, using a user-defined *g2-to-g2-interface* object named node-chad-connection:

```
start random-color(new-object) across node-chad-connection
```

Hint To start a procedure on a remote G2, you must declare the procedure as remote by using a remote procedure declaration. Remote procedure declarations are described in the *G2 Developer's Guide*.

Starting a Procedure with a Priority

To start any procedure with a non-default priority:

➔ start *procedure* ([*argument* [, ...]) at priority *integer-expression*

This action schedules the specified procedure at *integer-expression*, which you provide as a value from 1 to 10, 1 being the highest priority and 10 the lowest.

Specifying a priority overrides the default priority of the invoked procedure. Here is an example of the optional priority statement:

```
start provide-a-color(this workspace) at priority 4
```

Starting a Procedure after a Time Interval

To start any procedure after a time interval:

➔ start *procedure* ([*argument* [, ...]) after *time-interval*

This action schedules the specified procedure to execute at the current time, plus the given interval.

```
start get-new-color(this workspace) after 10 seconds
```

transfer

This action transfers items to the mouse, or on and off of workspaces, and to and from items and workspaces. Use the transfer action to:

- Transfer one or more items to the mouse.
- Transfer an item that is not on a workspace to a workspace.
- Transfer an item that is on a workspace to:
  - Another workspace.
  - Any attribute whose value can be an object.
- Transfer an object that is the value of an attribute to:
  - A workspace.
  - Any attribute whose value can be an object.
- Remove an item from a workspace.

The transfer action works only on transient items. You cannot transfer permanent items using this action.

Note You cannot transfer a workspace or a connection.

Transferring Object Attributes

To transfer objects to and from objects, and from workspaces to objects:

➔ transfer {*item* | the *attribute* of *item*} to {*item* | the *attribute* of *item*}

While you can transfer virtually any attribute object to a workspace, some restrictions apply to the target attribute when transferring one attribute object to another object, or an object upon a workspace to the attribute of an object.

- The target attribute must already exist. For example, to use this expression:

  ```
  transfer my-variable to the volume of gas-tank
  ```

  the object gas-tank must have an existing volume attribute.
- A target attribute cannot already have an object value.
- The class of the object being transferred must conform to the type-specification (if any) in the attribute description of the target attribute.

For example, to transfer an object attribute to a workspace:

```
transfer the temp of tank1 to new-workspace at (50, 50)
```

To transfer an object attribute to another object:

```
transfer the temp of tank1 to the new-temp of specialty-pump
```

To transfer an object upon a workspace to the attribute of an object:

```
transfer tank1 to the equipment of tank-holder
```

Referencing Transferred Objects

When transferring a named object to the attribute of an object, the transferred object retains its name. You can then reference the object either as the attribute of its object, or by its name. For example, if temperature-var is a float-variable, which you transfer to be the temp attribute of new-object, you can change the value of temperature-var with either of these statements:

```
conclude that temperature-var = 100.6
```

```
conclude that the temp of new-object = 100.6
```

Transferring an Item to the Mouse

To transfer a transient item to the mouse:

➔ transfer *item* to the mouse of *g2-window*

This action lets you attach the specified *item* to the mouse pointer so that a user can interactively place the item upon a workspace in the specified *g2-window*.

When entering a transfer item to the mouse action, the Text Editor includes a prompt for a positioning statement, such as at (x, y). Adding such a statement has no effect on the transfer action. The item is simply transferred to the mouse at its current location.

In the next example from the text of an action button, G2 creates a new connection post in the current window and transfers it to the mouse. In this case, the connection post will appear attached to the mouse pointer at its current position. Clicking the mouse places the item on the workspace. Action buttons can use the statement this window to refer to the current G2 window.

```
in order
    create a connection-post C
    and transfer C to the mouse of this window
```

Note You cannot transfer a connected item to the mouse.

If the user has already pressed down a mouse button when the transfer to the mouse action begins, the user must release that button and press a mouse button again for G2 to transfer the item to a workspace.

Transferring More Than One Item to the Mouse

You can use the transfer *item* to the mouse action to attach more than one item to the mouse. If you do so, G2 queues the items in a last-in, first-out basis.

Because G2 displays only one item at a time under the mouse pointer, as the user places each item upon a workspace, the next transferred item appears under the mouse pointer.

Transferring an Item to a Workspace

To transfer an item from one workspace to another, or to transfer an item that is not on a workspace to a workspace:

➔ transfer *item* to {*kb-workspace* [at (*x*, *y*)]

The at (*x*, *y*) statement represents the workspace unit integer values that specify the horizontal and vertical coordinates of the item upon the target workspace. An example is:

```
in order
    create an auto-1 A1;
    and transfer A1 to this workspace at (50, 50)
```

Removing an Item from a Workspace

To remove an item from a workspace:

➔ transfer *item* off

G2 removes the item you specify from its current workspace, however, the item exists in the KB as a transient item that you can find through Inspect, but which is not associated with any workspace. An item must be transient before you can transfer it off of its workspace. An example is:

```
make auto1 transient;
transfer auto1 off
```

update

This action updates display items and variables.

Updating a Display Item

To update an item:

➔ update *item*

A display item can be a chart, a dial, a freeform table, a graph, a meter, or a readout table.

Graphs are a superseded capability. For more information see Appendix F, Superseded Practices.

When G2 executes the update action, the item is updated programmatically. The update action is the only means of updating charts. An example is:

```
conclude that the status of every gas-tank T = new-level;
update every readout-table upon gas-tanks-filling
```

This example changes the status of every gas-tank to new-level, and directly updates every readout table on the gas-tanks-filling workspace.

Updating a Variable

To update a variable:

➔ update *variable*

This action updates the variable you specify. The update action causes G2 to data seek for a new variable value, even if it already has a current value.

Using the update action for a variable with an unsolvable formula causes G2 to retry continuously to obtain a variable value after failure once the action is invoked. Such behavior can adversely affect KB performance. To avoid constant retries from occurring, make appropriate adjustments to the timeout-for-variables and the retry-interval-after-timeout attributes of the Inference Engine Parameters system table.

21

Expressions

Describes the purpose and syntax of each G2 expression.

Introduction

This chapter describes how G2 obtains items and values by evaluating expressions. This chapter also shows how to form each expression that G2 supports.

An **expression** indicates how G2 should obtain a value when performing a computation. G2 obtains one value from each expression that it evaluates. To *evaluate* an expression means to obtain one value after performing the specified operations on other referenced values.

The same expression might not produce a value each time that G2 evaluates it. This can happen, for instance, when an item referenced in the expression no longer exists, when a conclude action changes a value, or when the expression refers to a value given by a variable whose value has expired. G2 takes special actions when it cannot obtain a value from an expression. See Evaluating Expressions.

You enter expressions as you code actions and statements in your KB's rules, procedures, and methods. You can also use the same kinds of expressions in the Inspect facility.

The form of G2 expressions is consistent with G2's English-like language. Use G2's syntax-directed Text Editor to enter expressions, as described in The Text Editor.

Forming an Expression

An expression consists of one or more **terms**. In some expressions, the terms combine with an appropriate **operator**. Each term in an expression is either a literal or another expression, also called a **subexpression**.

The next example illustrates how an expression's operator combines with its terms. Here, the arithmetic addition operator combines a literal term with an attribute expression:

```
conclude that flow-variable = 10 + the rate-of-change of outflow-pipe
```

Evaluating Expressions

G2 evaluates each expression to a single item or to a single value of a specific type. G2 evaluates an expression either successfully or unsuccessfully. When G2 evaluates an expression successfully, the expression produces an item or a value, as specified for that expression.

When G2 cannot evaluate an expression successfully, G2 produces a ***no value*** **condition**. Producing a *no value* condition also causes G2 to signal an error. G2 displays a *no value* condition as the reserved symbol none.

Tip To avoid generating error conditions due to non-existing items or values, you can use G2's existence expressions to test for a *no value* condition, as described in Existence of an Item or Value, Using the Has a Value Expression for variables, and Referring to Items or Values.

G2 produces a *no value* condition in these circumstances:

- When there is no possibility of obtaining a value.
- When attempting to obtain a current value that has expired.
- After unsuccessfully attempting to perform an operation that specifies a type mismatch.

Never Obtaining a Value

If no possibility exists of obtaining a value for an expression, evaluating that expression produces a *no value* condition. This occurs when:

- The expression refers to an item that does not exist.
- The expression refers to an attribute that contains no value.
- The expression refers to a variable that has never received a value.

Not Obtaining a Value at this Time

If an expression refers to a variable whose current value has expired, evaluating that expression produces a *no value* condition.

G2 can also produce a *no value* condition after evaluating a conditional expression whose if-clause expression produces the truth-value false, but has no else-clause. Conditional expressions are described in Conditional Evaluation.

Finding a Type Mismatch

When an expression specifies an operator, a mismatch can exist between the operator and the type of any expression term, or among the expression terms. This occurs, for instance, when an expression specifies the + (addition) operator for a numeric value and a symbol value.

When a type mismatch prevents G2 from evaluating an expression, G2 signals an error.

Determining When Expressions Expire

Each value that G2 uses has an **expiration time**, which is the point in time when the value's expiration time interval runs out. Most G2 values have an expiration time of never, indicating they are valid indefinitely. Variables are the exception, because they are the only values in G2 that can expire. G2 determines a value's expiration time based on when the value was obtained at its source and on the explicit or implicit expiration time intervals of the sources for that value.

To illustrate, if your KB concludes a value into a variable at 1000 seconds of G2 clock time, and the variable's validity-interval attribute contains the value 10 seconds, then the expiration time of this value is at 1010 seconds of G2 clock time. The validity interval of a variable is an example of an explicit expiration time interval.

If a value's expiration time interval has passed, that value has *expired*. A variable whose value has expired also has *no current value*.

The value that G2 obtains from evaluating an expression also has an expiration time. G2 determines this point in time as the current KB runs, based on the expiration times for the values that are the expression's terms. In some contexts, such as within an executing procedure, an expression's expiration time must be indefinite, or else G2 signals an error. In these cases, if evaluating an expression produces a value at all (that is, if G2 does not produce an error condition), that value never expires.

In other contexts, such as in rules and displays, G2 might evaluate an expression that refers to the expired value of a variable. In such cases, the expression's expiration time depends upon the expiration times of the values that are its terms.

Note When G2 evaluates an expression that produces a *no value* condition, the expression's expiration time is also a *no value* condition.

For the majority of expressions, the expiration time is the minimum expiration time of its terms. A notable exception to this rule is an expression that includes the and operator. For more information, see Affecting the Expiration Time.

Understanding Transactions and Transaction Scopes

The expiration time of the values that G2 obtains by evaluating expressions is significant because of G2's practice of performing your KB's activities within transactions. A **transaction** is a sequence of your KB's processing in which the set of values in use must remain valid, unchanged, and consistent with respect to each other.

G2 automatically identifies when a sequence of processing must take place as a transaction; this is called a **transaction scope**. G2 also automatically ensures that the values referenced within a transaction's scope remain valid and consistent until G2 finishes performing the transaction.

For instance, by default, the statements and actions that G2 performs within a called procedure form one transaction. G2 ensures that the values referenced in the procedure's statements and actions do not change while the procedure is executing, even if the source of a particular value (such as an attribute or a parameter) is updated by another KB activity while this procedure is executing. Thus, references to a particular attribute in different statements within the procedure will produce the same value.

G2 must perform the current KB's activities within transactions for these reasons:

- The scope of the knowledge contained in each item is global. Any executable item can refer at any point in its execution to the knowledge in any other activated item.
- As the current KB runs, G2 can simultaneously perform more than one thread of execution. G2 can simultaneously execute procedures, invoke rules, perform data seeking and respond to other external connections and interfaces.

Because G2 performs the KB's activities within transactions, your KB's executable items can produce reliable results.

Using Generic Reference Expressions

Certain expressions allow you to use generic references to a set of items or values, depending on the expression's context in an action or procedure statement. These are called generic reference expressions.

Generic reference expressions can refer to one or more items, attributes, variable or parameter values, or list or array elements in certain contexts. For instance, the generic reference expression any custom-object connected to my-valve in the antecedent of the following rule refers to a set of items:

```
whenever the authors of any custom-object connected to my-valve
    receives a value then
    post "The object [the public-name of the custom-object] has been edited."
```

The generic reference expression any symbol in any symbol-list in the antecedent of the following rule refers to a set of values:

```
if any symbol in any symbol-list is not ok then
    post "Validation of [the superior item of the symbol-list]
            found discrepancies."
```

To create a generic reference expression:

➔ *generic-reference-expression* := {*class-name* | *type*} [*local-name*] [*generic-reference-qualifier*]

Including a Generic Reference Qualifier Expression

In a generic reference expression that identifies a set of items, use a **generic reference qualifier expression** to refer to those items with respect to their system-defined relationships with other items.

To create a generic reference qualifier expression:

➔ *generic-reference-qualifier* :=
{upon *kb-workspace*} | {connected *connected-expression*} |
{at *at-expression*} | {nearest to *item*} |
{superior to *kb-workspace* | *object-attribute*} |
{that is *relation-name item*} | {named by *symbolic-expression*} |
{in {*g2-list* | *g2-array*} } | {name of *item*}

For example, the antecedent of this rule specifies a generic reference expression that includes a generic reference qualifier expression connected to my-valve:

whenever the authors of **any custom-object connected to my-valve**
receives a value then
post "The object [the public-name of the custom-object] has been edited."

Using Quantifiers

You must prefix each generic reference expression with a G2 reserved symbol, called a quantifier. Each **quantifier** indicates whether the expression produces *one*, *one and only one*, *at least one*, or *any number* of items or values.

You use particular quantifiers in certain contexts, as summarized in this table:

Quantifier	Meaning	Context and Example
a an	At least one	Use only in there exists expressions: there exists a tank T in tank-list-1 such that (T is full)
any	Zero or more	Use only in generic rules: if **any** tank T in tank-list-1 is full then remove T from tank-list-1

Quantifier	Meaning	Context and Example

each	Zero or more	In expressions over a set of values, items, or attributes, use to compute a value based on the set:
		`the count of **each** tank T in tank-list-1 such that ( T is full )`
		In expressions in procedures, use to iterate over a set of items:
		`for T = **each** tank in tank-list-1 do ...`

every	Zero or more	In a for every expression, use to iterate over a set of items or attributes:
		`for **every** tank T in tank-list-1 ( T is full )`
		In the consequent of a rule, causes an action to execute on every item that matches the reference:
		`if the time-of-day of floor-clock > 600 then show the subworkspace of **every** die-cutting-machine`
		In this example, G2 iterates over all tanks in tank-list-1, and produces the truth-value true if all tanks are full.

the	One and only one	In any expression: **the** tank in tank-list-1 exists This expression produces the truth-value true only when there is one tank in tank-list-1.

Note When the the quantifier is used together with named by, it has the meaning "At least one". This is an exception. See also Referring through a Symbolic Expression.

It is possible to use the every quantifier in outer and inner statements:

```
for every tank T in tank-list1 conclude that
    the background-color of the workspace of every T is the symbol blue
```

When doing so, however, the second every statement cannot refer to a local variable declared in the first, and a statement such as the example here would not compile correctly.

Embedded Generic Reference Expressions

In some G2 expressions you must specify an embedded generic reference expression. In these cases, you must use a particular quantifier, not any quantifier.

For example, the there exists expression requires only the a or an quantifier for its embedded generic reference expression. This rule's antecedent tests whether any items exist that have a value other than OK in the notes attribute:

```
initially
    if there exists an action-button such that
        (the text of the notes of the action-button /= "ok") then
        post "At least one action-button in this KB has errors."
```

Note In this chapter the syntax descriptions for each expression show the required quantifier for the embedded generic reference expression, if any.

Using Class-Qualified Names

G2 allows duplicate class or attribute names in two contexts:

- Attributes of different classes can have the same names.
- Methods defined on different classes can have the same names.

To avoid ambiguity in such cases, G2 uses **class-qualified names** to refer to the duplicate attributes and/or methods. A class-qualified name uses the class-qualifier operator, two colon characters (::), and has the syntax:

class::*name*

where *class* is the class that defines the attribute or method, and *name* is the attribute or method's name.

G2 supports class-qualified names in all grammar categories and value expressions that support symbols:

- User-defined attribute and method names.
- Symbolic values for user-defined attributes.
- The initial and current values of symbolic variables and parameters.
- Symbolic array and list elements.
- All item named by expressions.

Thus you can use expressions such as:

- the symbol *class*::*name*
- symbol "*class*::*name*"

- conclude that *reference* is *class*::*name*
- *attribute* initially is c*lass*::*name*

When G2 provides an attribute or method name, and that name would be ambiguous if given without qualification, G2 provides a class-qualified name. Thus an expression that iterates over all user-defined attributes can distinguish among duplicate attributes.

Using Local Names in Expressions

You can use a **local name** to simplify some expressions, including generic reference expressions. For example, using two local names, V and T, you can recode this expression:

```
the volume of the tank nearest to connection-post-A +
    the maximum-volume-at-shutdown of the inflow-pipe connected to
        the tank nearest to connection-post-A -
        ( 0.10 * the volume of the tank nearest to connection-post-A )
```

as this expression:

```
the volume V of the tank T nearest to connection-post-A +
    the maximum-volume-at-shutdown of the inflow-pipe connected to
    T - ( 0.10 * V )
```

You can use local names implicitly and explicitly. You can also use existing classes or attribute names as local names.

Implicit Use

You can implicitly use local names in expressions to simplify the expression. A local name provides a quick method of referring to the value produced by an expression, such as:

```
if the temperature T1 of tank-1 > 200 and T1 < 300
    then ...
```

In this example, you use the local name T1 to refer to the temperature of tank-1.

An implicit local name must be unique to the expression in which it appears. The scope of an implicit local name is also limited to the expression in which it appears. Thus, you can use the same implicit local name in different expressions in the same rule, action, or procedure statement.

Explicit Use

In procedures and methods you can also use declared local names. In the local name declaration section of the procedure, you must declare each local name that receives an assignment (in a = statement) within that procedure.

This example demonstrates how a procedure can use both explicit and implicit local names:

```
local-names-sample ( ) = ( )
{ C is an explicit local name. }
C : class conveyor ;
begin
    for C = each conveyor upon this workspace do
        { S is an implicit local name. }
        case ( the status S of C ) of
            paused, stopped :
                post "Status of conveyor [the public-name of C] is [S]." ;
        end ; { case }
    end { do }
end
```

Tip Declaring a local name in a procedure allows G2 to compile each statement that uses the name more efficiently.

Class or Attribute Name Use

You can use a class name or attribute name as a local name in a rule, in which case you must use the the quantifier. When you first refer to an attribute in a rule, you must identify the item of the attribute. When you make additional references to that same attribute, however, you need not identify the item, because the item is clear from the first reference.

For example, this rule uses the name of the temperature attribute of the tank class as a local name:

```
if the temperature of tank-1 > 200 and the temperature < 300 then ...
```

The second reference to the temperature attribute omits the reference to tank-1. G2 assumes that the reference the temperature still refers to the temperature of tank-1.

Using Literals

A **literal** directly represents a value of a specific G2 type. The specific value types are integer, float, truth-value, text, and symbol. G2 evaluates a literal to the value that it signifies:

Type of Value	Example	Meaning
integer	54	Integral number in decimal notation.
long	536870912L -100L	Signed 64-bit integral number in decimal notation.
float	165.70 1.0e204	Rational number in decimal notation; rational number in exponential notation.
truth-value	true false 1.0 true 0.33 true -1.0 true -0.79 true	A fuzzy truth-value.
text	"your computer"	Case-sensitive text.
symbol	AB-#304-CABLE X-29-RADAR	Unique identifier.

Note The case of alphabetic characters in a literal symbol is not significant.

For more information about G2 types, see Distinguishing Value Types.

It is valid to specify a literal value as a term in an expression, if the type of the literal's value is valid for that term.

Using Operators in Expressions

An **operator** specifies a type-specific operation. Each operator combines with one or two expression terms, also called **operands**. For most operators, each operand can be a distinct expression.

Each operator in an expression must be appropriate for the types of its operands. For instance, it is invalid for G2 to combine an arithmetic operator with an operand that does not produce a value of type integer or float.

G2 offers these operators:

- **Arithmetic operators**: Unary negation, addition, subtraction, multiplication, division, and exponentiation for operands that produce values of type quantity, integer, and float.
- **Logical operators**: Boolean operations and, or, and not for operands that produce values of type truth-value.
- **Relational operators**: As follows:
  - For operands that produce values of type quantity, integer, float, and text: equal to, not equal to, greater than, greater than or equal to, less than, and less than or equal to.
  - For operands that produce values of type symbol: equal to and not equal to.
  - For operands that produce values of types truth-value: equal to, not equal to, is less true than, is more true than, is not less true than, and is not more true than.
- **Concatenation operator**: For operands that produce values of any type.
- **Class-qualifier operator**: For operands that name a class and a class-specific attribute of that class.

The following sections describe in detail how these operators and their operands participate in an expression.

Using Arithmetic Operators

G2 uses the following reserved characters to signify arithmetic operators:

Reserved Character	Arithmetic Operation
- (hyphen)	Negation
+ (plus)	Addition
- (hyphen)	Subtraction
* (asterisk)	Multiplication
/ (forward slash)	Division
^ (caret)	Exponentiation

Note For G2 to interpret the - (hyphen) character as an operator, rather than as a character in a symbol, include at least one space before and after it.

Note See the arithmetic function remainder in Arithmetic Functions, which is similar to the division (/) operator.

Identifying the Default Order of Evaluation

By default, G2 evaluates an expression with an arithmetic operator from left to right. For example, G2 evaluates this expression:

```
W * X - Y + Z
```

as follows:

1. Obtain the value of the term W.
2. Obtain the value of the term X and multiply it by the previous value.
3. Obtain the value of the term Y and subtract it from the previous product.
4. Obtain the value of the term Z and add it to the previous difference.

Using Parentheses to Affect the Order of Evaluation

You can use parentheses to override the default order in which G2 evaluates an expression that includes an arithmetic operator. In the following expression, enclosing the middle two terms and their operator in parentheses changes how G2 evaluates the expression:

```
W * ( X - Y ) + Z
```

In this case, G2 evaluates this expression as follows:

1. Obtain the value of the term W and remember it.
2. Obtain the value of the term X.
3. Obtain the value of the term Y and subtract it from the previous value.
4. Multiply the memorized value of W by the previous difference.
5. Obtain the value of the term Z and add it to the previous product.

Precedence of Arithmetic Operators

When evaluating expressions that include arithmetic operators, G2 observes the following precedence among operators.

Precedence	Reserved Character	Arithmetic Operation(s)
1	-	Negation
2	^	Exponentiation
3	* and /	Multiplication and division
4	+ and -	Addition and subtraction

For example, in this expression:

```
W + X * Y + Z
```

because the * (multiplication) operator has higher precedence than the + (addition) operator, G2 evaluates the entire expression as if its terms are combined within parentheses as follows:

```
W + (X * Y) + Z
```

Coercion of Values Returned from Arithmetic Operators

When evaluating an expression that includes an arithmetic operator, the type of value obtained depends upon a combination of the operator and the types of the values obtained from the operand terms. The following table summarizes this behavior:

Reserved Character (Operation)	Types of Operand Values	Type of Resulting Value
- (negation)	quantity	quantity
	integer	integer
	long	long
	float	float
+ (addition) - (subtraction) * (multiplication)	quantity, quantity	quantity
	quantity, integer	quantity
	quantity, long	float
	quantity, float	float
	integer, integer	integer
	integer, long	long
	integer, float	float
	long, long	long
	long, float	float
	float, float	float
/ (division)	quantity, quantity	float
	quantity, integer	float
	quantity, long	float
	quantity, float	float
	integer, integer	float
	integer, long	float
	integer, float	float
	long, long	long
	long, float	float
	float, float	float
^ (exponentiation)	quantity, quantity	float
	quantity, integer	float
	quantity, long	float
	quantity, float	float
	integer, integer	float
	integer, long	float
	integer, float	float
	long, long	float
	long, float	float
	float, float	float

Constraints on Exponentiation Operations

G2 disallows certain values to participate in an exponentiation operation, as follows:

- G2 disallows an exponentiation operation that produces a complex number.
- G2 disallows an exponentiation operation where the base value is zero (that is, 0 or 0.0) and the exponent value is a negative number.
- G2 disallows an exponentiation operation where the base is negative and the exponent is positive.

Using Logical Operators

Logical operators specify boolean operations on either one or two operand terms for which G2 obtains values of type truth-value, as summarized in this table:

Operator	Resulting Value	Example
and (two operands)	true, only if the values obtained for both operand terms are true false, otherwise	X and Y
or (two operands)	true, if the value obtained for either operand term is true false, otherwise	X or Y
not (one operand)	true, if the value obtained for the operand term is false false, if the value obtained for the operand term is true	not X

An expression that includes a logical operator produces a value of type truth-value.

These expressions include logical operators:

```
the tank is empty or the input-valve connected to the tank is broken

not valve-is-broken
```

Short-Circuited (Lazy) Evaluation of Logical Operators

G2's and and or operators are **short-circuited**:

- X and Y

  If X evaluates to false, G2 returns the value of X without evaluating Y; otherwise G2 returns the smallest of the values.

- X or Y

  If X evaluates to true, G2 returns the value of X without evaluating Y; otherwise G2 returns the largest of the values.

Short-circuited (also called lazy) evaluation avoids wasting processor time evaluating terms in a logical expression whose value is already known.

Affecting the Expiration Time

For an expression that includes a logical operator and that refers to one or more logical variables, the expiration time of the entire expression depends on which logical operator is used, as follows:

- For the and operator, G2 uses the expiration time of the first logical variable whose current value is false.
  - If each term in the expression produces the truth-value true, G2 evaluates the entire expression as the truth-value true, and the expiration time for entire expression is the expiration time among those of the operand terms that is nearest into the future from the current time.
  - If some terms produce a *no value* condition but none of the known terms is false, G2 produces the *no value* condition for the entire expression, and the expiration time for the entire expression also produces the *no value* condition.
  - If any term produces the truth-value false, G2 evaluates the entire expression as the truth-value false, and the expiration time of the expression is that of the term that produced false.

- For the or operator, G2 uses the expiration time of the first logical variable whose current value is true.
  - If all terms referenced in the expression produce the truth-value false, then G2 evaluates the entire expression as the truth-value false, and the expiration time for the entire expression is the expiration time among those of the operand terms that is nearest into the future from the current time.
  - If some terms produce the *no value* condition but none of the known terms produce the truth-value true, then G2 produces the *no value* condition for the entire expression, and the expiration time also produces the *no value* condition.
  - If any term produces the truth-value true, G2 evaluates the entire expression as the truth-value true, and the expiration time of the entire expression is the same as the expiration time of the term that produced true.

Tip Logical operators can also combine with terms that produce fuzzy truth values. For more information, see Producing Fuzzy Truth Values from Relational Operations.

Precedence and Order of Evaluation

The precedence for logical operators is as follows:

1 not

2 and

3 or

An example is:

```
X and not Y or Z and W
```

Because the not operator has higher precedence than and, and because the and operator has higher precedence than the or operator, G2 associates the operators and operands as follows:

```
( X and ( not Y ) ) or ( Z and W )
```

By default, logical operators have lower precedence than arithmetic and relational operators. So, G2 evaluates these two expressions in the same manner:

```
2 + X > Y and Z > W
```

```
( 2 + X > Y ) and ( Z > W )
```

Using Relational Operators

A relational operator causes G2 to compare the values obtained for its two operands and to return a value of type truth-value.

Use these relational operators to compare two terms for which G2 obtains a value of type quantity, integer, or float:

Reserved Character	Relational Operation	Usage
=	Is equal to	A = B
/=	Is not equal to	A /= B
>	Is greater than	A > B
<	Is less than	A < B
>=	Is greater than or equal to	A >= B
<=	Is less than or equal to	A <= B

Use these relational operators to compare two terms for which G2 obtains a value of type text:

Reserved Character	Relational Operation	Usage
=	Is equal to	A = B
/=	Is not equal to	A /= B
>	Is greater than	A > B
<	Is less than	A < B
>=	Is greater than or equal to	A >= B
<=	Is less than or equal to	A <= B

G2 ignores the alphabetic case when comparing two text values. For example:

- The text strings "Text" and "text" are equal, that is, "Text" = "text" is true.
- The text string "Text" is a member of the text-list that contains "text".
- The text string "Text" is a member of the sequence that contain "text".

G2 also ignores the type when comparing two quantity values. For example:

- The integer 2 and the float 2.0 are equal, that is, 2 = 2.0 is true.
- The float 2.0 is a member of the quantity-list that contains the integer 2.
- The float 2.0 is a member of the sequence that contains the integer 2.

Use these relational operators to compare two terms for which G2 obtains a value of type symbol, sequence and structure:

Reserved Symbols	Relational Operation	Usage
=	Is equal to	A = B
/=	Is not equal to	A /= B

Use these relational operators to compare a term for which G2 obtains a value of type symbol with a literal symbol:

Reserved Symbols	Relational Operation	Usage
is	Is the same symbol as	A is red
is not	Is not the same symbol as	A is not red

Tip There are corresponding relational operators for fuzzy truth expressions. See Fuzzy Truth Operators for more information.

Producing Fuzzy Truth Values from Relational Operations

A truth-value expression that includes a relational operator produces a value of type truth-value. As introduced in Using the Truth-Value Type, a value of type truth-value can range from -1.0 true to +1.0 true, where -1.0 true signifies complete certainty that a comparison is false and +1.0 true signifies complete certainty that a comparison is true. Thus, such an expression can also produce a fuzzy truth value, which signifies a partial certainty in the truth of a comparison.

You produce a fuzzy truth value from a truth-value expression with a relational operator when you also specify a **fuzzy truth band subexpression**. In the following expressions that contain a relational operator, the fuzzy truth band subexpressions appear in boldface:

X < Z **(+ - 4)**

the level of tank-1 < 110 **(+ - 5)**

Specifying a Fuzzy Truth Band Subexpression

To produce a fuzzy truth value from an expression that includes a relational operator, specify a fuzzy truth band subexpression. Its syntax is:

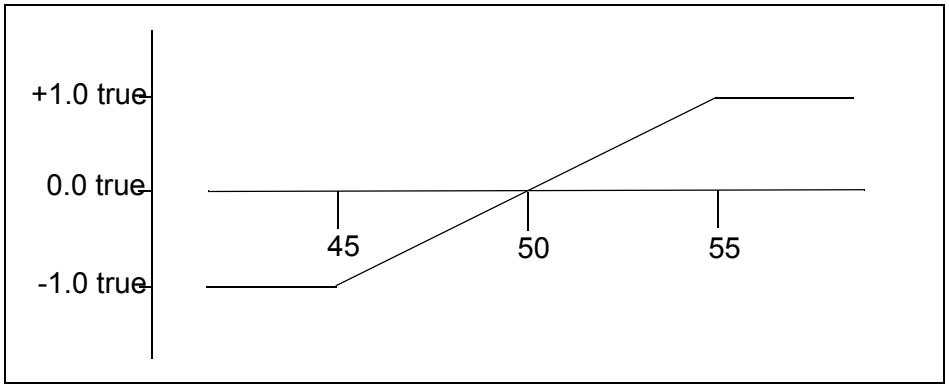
(+- *quantity-expression*)

For example, this truth-value expression compares an attribute's value with a literal integer, according to a fuzzy truth band subexpression:

```
the flow of pump-1 > 50 ( +- 5 )
```

The fuzzy truth band subexpression (+- 5) specifies that the fuzzy truth value produced corresponds to a range of values, from 45 (50 minus 5) to 55 (50 plus 5). That is, as the value of the flow attribute of pump-1 ranges from 45 to 55, the corresponding fuzzy truth values range from -1.0 true to +1.0 true.

Thus, the expression the flow of pump-1 > 50 (+- 5) has a +1.0 true truth-value when the the flow of pump-1 is 55 or greater, and a -1.0 true truth value when the flow of pump-1 is 45 or less, as shown in the next diagram:



As the diagram indicates, the truth-values in the fuzzy-truth band fall along a line between the specified end-points of the band.

You can specify a fuzzy truth band subexpression in the antecedent of a rule. G2 executes such a rule after consulting the value of the truth-threshold attribute in the Inference Engine Parameters system table. Its value can range from 0.0 to 1. 0, and its default value is .800 (or +0.8 true). When G2 invokes a rule, for G2 to execute the actions in the rule's consequent, the truth-value that results from evaluating the rule's antecedent must be greater than or equal to the truth-threshold.

For example, the truth-value expression in this rule's antecedent specifies a fuzzy truth value subexpression:

```
if the temperature of tank-1 >100 ( +- 2 ) then
    conclude that the tank-is-boiling of tank-1 is true
```

If the temperature attribute of tank-1 has a value of 101.8, the subexpression causes the entire truth-value expression to produce the fuzzy truth value of

+0.9 true. Because the value +0.9 true is greater than the value of truth-threshold (that is, +0.8 true), G2 executes the action to conclude the truth-value (+0.9 true) into the tank-is-boiling attribute of tank-1.

Using Logical Operators with Terms That Produce Fuzzy Truth Values

You can use logical operators to combine terms in a truth-value expression that produce fuzzy truth values. For example, you can specify a truth-value expression like this:

```
X > Y ( +- 5 ) and Z > W ( +- 3 )
```

G2 evaluates these expressions as follows:

- If the and operator is used, G2 produces the lesser fuzzy truth value as the value of the entire truth-value expression.
- If the or operator is used, G2 produces the greater fuzzy truth value as the value of the entire truth-value expression.
- If the not operator is used, G2 produces the fuzzy truth value with the opposite sign.

Given this truth-value expression that includes the and logical operator:

```
X > Y ( +- 5 ) and Z > W ( +- 3 )
```

G2 evaluates this expression as follows:

1. G2 finds values for X and for Y, then uses the fuzzy truth band (+- 5) to calculate a truth-value for the subexpression X > Y.
2. G2 finds values for Z and for W, then uses the fuzzy truth band (+- 3) to calculate a truth-value for the subexpression Z > W.
3. G2 produces the lesser of the two fuzzy truth values as the value of the entire expression. For example, if the value of the left side is +0.6 true and the value of the right side is +0.7 true, G2 produces the value +0.6 true for the entire expression.

Given this truth-value expression that includes the or logical operator:

```
X or Y >= W ( +- .3 )
```

G2 evaluates this expression as follows:

1. G2 finds the value for X.
2. G2 finds values for Y and for W, then uses the fuzzy truth band (+- .3) to calculate a truth-value for the subexpression Y >= W.
3. G2 produces the greater of the two fuzzy truth values as the value of the entire expression. For example, if the value of the left side is +0.4 true and the value of the right side is +0.9 true, G2 produces the value +0.9 true for the entire expression.

Given this truth-value expression that includes the not logical operator:

```
not ( X > Y ( +- .2 ) )
```

G2 evaluates this expression as follows: If the subexpression X > Y (+- .2) evaluates to a fuzzy truth value of +0.3 true, G2 produces the value -0.3 true for the entire truth-value expression.

Fuzzy Truth Operators

You can specify a truth-value expression that compares one fuzzy truth value to another. This kind of expression uses one of these fuzzy truth operators:

- = (is equal to)
- /= (is not equal to)
- is more true than
- is less true than
- is not more true than
- is not less true than

A truth-value expression that specifies a fuzzy truth operator produces a truth-value of either true or false. It does *not* produce a fuzzy truth value.

The following truth-value expressions specify both fuzzy truth band subexpressions and fuzzy truth operators. The fuzzy truth operators appear in boldface:

```
Y < 6 ( +- 3 ) ) /= ( Z > 4 ( +- 2 )
```

```
( X < Z ( +- 4 ) ) is less true than ( X > 7 ( +- 2 ) )
```

```
the level of tank-1 < 110 ( +- 5 ) ) is not more true than
  ( the level of tank-2 < 120 ( +- 10 )
```

In an expression that specifies a fuzzy truth operator, if either truth-value expression term being compared is more complex than a reference to a logical variable or logical parameter, enclose that term in parentheses, such as:

```
( X > Y ( +- 2 ) ) is more true than ( Z > X ( +- 3 ) )
```

Using the Concatenation Operator

G2 provides a concatenation operator that you specify by using the square bracket characters: [ ]. You use this operator to insert, prepend, or append a text version of any value to a literal text value.

For information on using square brackets to get the Unicode character code of a single character in a text, see Getting Unicode Character Codes.

To concatenate by inserting, specify a text expression such as:

```
"The direction of flow for the water-pipe [the public-name of the water-pipe
  connected to tank-1] is [the direction-of-flow of the water-pipe
  connected to tank-1]."
```

In this case, G2 converts the value produced by the expression the public-name of the water-pipe connected to tank-1 to a text value and inserts it into the literal text enclosed in double quotes. Likewise, G2 converts the value produced by the expression the direction-of-flow of the water-pipe connected to tank-1.

To concatenate by prepending, specify an expression such as:

```
"[the public-name of the first alarm in the
  master-alarm-list] is ready for your response."
```

To concatenate by appending, specify an expression such as:

```
"The sum of the volumes is [the volume of tank-1 +
  the spillage-volume of drainage-pipe]."
```

You can also use the concatenation operator to produce an entire text value, as in this change the text of action:

```
change the text of the item-configuration of my-item to
  "[the text of the item-configuration of workspace-label-free-text]"
```

G2 can convert to a text value any expression specified within the concatenation operator characters.

Formatting Using the Newline Character

You can include a newline character in a literal text value, by pressing Control + j within the two double quotes (") characters that enclose the value. However, including a newline character within the [and] concatenation operators only formats that line of code; G2 does not include the newline character in the literal text value.

For example, this literal text value includes a newline character after the word "ready":

```
"[the public-name of the first alarm in the master-alarm-list] is ready
     for your response."
```

For example, this literal text value does not include a newline character, because its author specified the newline character between the [and] operators:

```
"[the public-name of the first alarm in the
  master-alarm-list] is ready for your response."
```

Formatting Numeric Values

Within the concatenation operator, you can optionally specify how G2 formats a numeric value: its number of decimal digits to the left and/or right of a decimal point, as a time-stamp, or as a time interval.

For example, this post action embeds a numeric (the pressure of the tank) in a literal text value, with the numeric value formatted with three digits left and two digits right of the decimal point:

```
if the pressure of any tank has a value
    then post "Pressure measurement of [the public-name of T] is
        [the pressure of the tank as ddd.dd]."
```

For both a float pressure value of 45678.8 and an integer pressure value of 45678, G2 posts: 4.57e4.

Here is the syntax for a formatting expression:

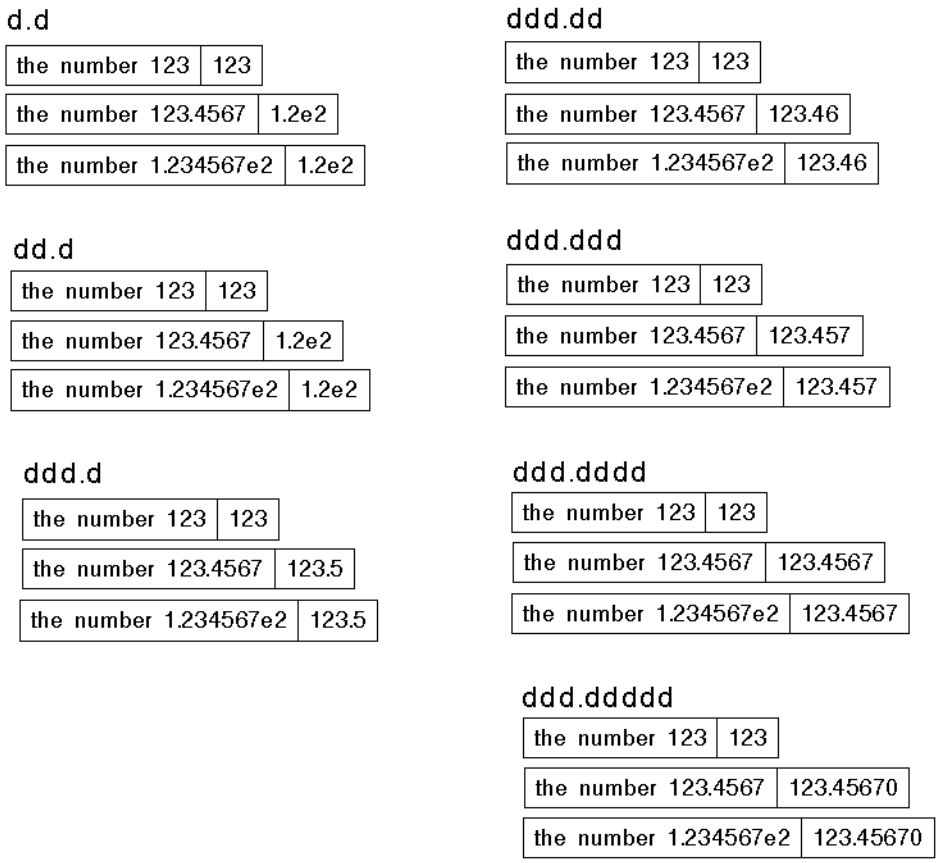
as {*ddd.dddd-format* | a time stamp | an interval}

The formatting alternatives in this expression are also those for formatting the display of a float value in a readout table. The display-format attribute of a readout table has no effect on integer values. See Readout Tables.

For a float value, a *ddd.dddd-format* expression specifies the number of digits in the value to display left and right of the decimal point. For instance, the expression dd.ddd specifies to format a number with two digits left of the decimal point and three digits right of the decimal point.

For a float value expressed in exponential notation, a *ddd.dddd-format* expression adheres to the specified format and adds the exponential notation to the right. For example, an exponential value that adheres to the expression dd.dddddd is 2.323372e4.

The next figure shows a workspace whose readout tables demonstrate how the same *ddd.dddd-format* expression affects differently the display of an integer value, a nonexponential float value, and an exponential float value.



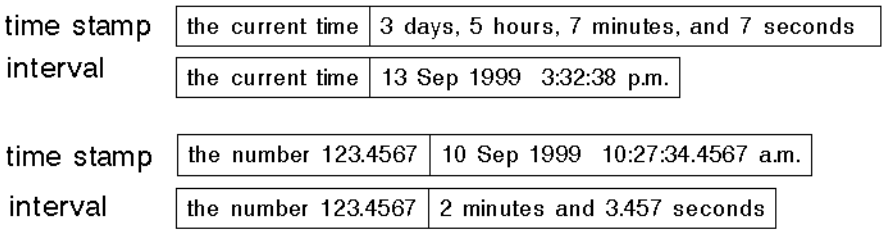
You can also format time values using a formatting expression. For example, this post action embeds a time value formatted as a time-stamp in a literal text value:

```
post "The last update occurred with a time-stamp of
    [the update-time of my-object as a time stamp]."
```

A time value is either an integer or float value. You can capture a time value by using one of the G2 time functions or one of these expressions:

- the current time
- the current real time
- the current subsecond time
- the current subsecond real time

The next figure shows a workspace whose readout tables demonstrate how the as time stamp and as interval formatting expressions affect differently the display of the current time and a literal float value.



Notice that when you format a number as a time-stamp, as in the third readout table in the figure above, the displayed value represents a time-stamp as of that number of seconds after the current KB was started, *not* after the current time.

Producing a Symbol Value

To procedure a symbol value:

➔ the symbol *symbol*
-> *symbol*

This expression produces a value of type symbol. For example, the expression the symbol red produces a symbol value red that G2 does not interpret as the name of an item, attribute, class, and so on.

Referring to a Superior or Inferior Class

To refer to the superior or inferior class of an item:

➔ the symbol [*local-name*] that is
{a superior-class | an inferior-class} of *symbolic-expression*
-> *symbol*

This generic reference expression produces one or more symbols that name the superior or inferior classes of the class named by the symbolic expression. This expression references a class *indirectly*.

Referring to Items or Values

The following expressions refer to items or values.

Existence of an Item or Value

To determine whether an item or value exists:

➔ *item-or-value* {exists | does not exist}
-> *truth-value*

This expression produces a truth-value that indicates whether the specified item exists in the current KB, or whether the specified value-expression has a value, that is, does not produce a *no value* condition. Use this expression to avoid references to nonexistent items and attribute values; such references cause G2 to signal an error.

For an item, use this expression to determine whether the item exists and is active. For example:

```
wait until proof-checker-1 exists checking every 5 seconds
```

For a value-expression, use this expression to determine whether evaluating the expression produces a value. For example:

```
if the public-name of the temperature T of my-tank exists
    then ...
```

Tip To test whether a variable or parameter has a value, or whether a variable has a current value, use the has a value and has a current value expressions. See Variable and Parameter Expressions.

There Exists

To determine whether an item or value exists:

➔ there exists {a | an} *generic-reference-expression*
such that (*truth-value-expression*)
-> *truth-value*

This expression produces a truth-value that indicates whether an item or value exists in the current KB that meets the criterion specified in a truth-value expression. Use this expression to determine whether an item or value *referenced generically* exists and is active.

If the specified generic reference expression refers to items, this expression produces either truth-value true or false, or a *no value* condition, as follows:

- Produces the truth-value true if the current KB contains at least one item that meets the specified criterion. The expiration time of this expression is the minimum expiration time of either the generic reference expression or the first instance of the embedded truth-value expression that produces the truth-value true.
- Produces the truth-value false if no items in the current KB meet the specified criterion. The expiration time of this expression is the minimum expiration time of either the generic reference expression or the minimum expiration time among those of the instances of the embedded truth-value expression.
- Produces the *no value* condition if no items meet the specified criterion, but some might be *unknown*.

If this expression produces the truth-value true, you can refer to the generic reference expression's local name in other expressions within this expression's transaction scope. In this case, the local name refers to a value when the such that phrase evaluated to the truth-value true. If there is no such that phrase, the local name refers to the first instance of the generic reference expression. For example, this rule's antecedent contains a there exists expression:

```
if there exists a vat V1 such that ( V1 is overflowing ) then ...
```

Here, the expression V1 is overflowing produces the truth-value true if at least one vat is overflowing, but produces the truth-value false if no vat is overflowing.

If the there exists expression produces the truth-value false, the value that the local name refers to is *not predictable*. In the sample rule shown above, if the there exists a vat V1 ... expression produces the truth-value false, then references to the V1 local name in this rule's consequent might produce a different truth-value.

Note You can optimize the execution of a there exists expression if it references indexed attributes. See Defining an Indexed Attribute for more information about indexed attributes.

Class or Type of Item or Value

To determine the class of an item or type of value:

➔ *item-or-value* is {a | an} {*class-name* | *type*}
–> *truth-value*

This expression produces a truth-value that indicates whether an item is an instance of the specified class (or any of its subclasses), or whether a value is an instance of any type. For example:

```
if the item I nearest to help-button is a custom-object then
conclude that the status of I is ok
```

By Generic Reference

To reference items or values generically:

➔ for every *generic-reference-expression* (*truth-value-expression*)
-> *truth-value*

This expression produces the truth-values true or false, or produces a *no value* condition, as follows:

- Produces the truth-value true if the specified truth-value expression produces the truth-value true for *every* item or value in the specified generic reference expression. If so, the expiration time of the expression is the *minimum* of the expiration times of the items or values produced by the generic reference expression and of all instances of the embedded truth-value expression.
- Produces the truth-value false if the specified truth-value expression produces the truth-value false for *any* item or value produced by the specified generic reference expression. The expiration time of this expression is the expiration time of the first instance of the specified truth-value expression that produces the truth-value false.
- Produces a *no value* condition if the specified truth-value expression does *not* produce the truth-value false for every item or value produced by the specified generic reference expression. In this case, the expiration time of the expression produces the *no value* condition.

This expression allows you to iterate through the items or values referenced in the specified generic reference expression and to detect whether a condition specified in the truth-value expression is true. This is similar to a universal quantifier in traditional logic.

For example:

```
for every valve V connected to tank-1 ( V is broken )
```

This expression produces the truth-value true if every valve connected to tank-1 is broken. Otherwise, it produces the truth-value false. Note that if no items or values match the generic reference expression, the expression produces the truth-value true.

Conditional Evaluation

To perform conditional evaluation:

➔ (if *truth-value-expression* then *item-or-value* [else *item-or-value*])
-> {*item* | *integer* | *float* | *symbol* | *text* | *truth-value*}

This expression produces a value, based on the result of evaluating a truth-value expression. Parentheses are required around this expression.

If there is no else phrase, and the truth-value expression produces the truth-value false, then the entire expression produces a *no value* condition.

The expiration time of this expression is the minimum of the expiration times of the specified truth-value expression and of G2's evaluation of either the then phrase or the else phrase.

You can nest this expression within other expressions, even with other conditional expressions. Some examples are:

```
the area of tank-1 *
  (if the current time < 2 then 10 else the level of tank-1)

the status-light of tank-1 =
  (if the level of tank-1 > 100 then the symbol red else the symbol green)

if (if valve-1 is open then 10 else 0) > the outflow of tank-1 then
     conclude that tank-1 is filling

(if tank-1 is empty then
  (if valve-1 is closed then the symbol green else the symbol red)
  else the symbol green)
```

Value Expressions

These expressions deal with the values of items or values.

Value of an Item or Value

To determine the value or an item or value:

➔ the value of *item-or-value*
-> {*item* | *integer* | *float* | *symbol* | *text* | *truth-value*}

This expression produces the value of the specified expression, which can be either an item or a value. If any variable referenced in the specified expression has never had a value, or if its current value has expired, then G2 performs data seeking for each such variable. If an attempt at data seeking for any variable referenced in the specified expression fails for any reason, this entire expression produces a *no value* condition.

The expiration time of the entire expression is the expiration time of the value produced by the specified expression. For example:

```
the value of
    (if the custom-variable CV that is synchronized-with the
        temperature-variable giving the temperature of tank-1 is not broken
    then CV else the temperature of tank-1)
```

This expression refers to the value of the custom-variable that is related to the temperature-value variable that gives the temperature attribute of tank-1. If either that custom-variable or the temperature-value variable that gives the temperature attribute of tank-1 does not have a current value, then G2 performs data seeking to obtain a value.

Has a Value

To determine whether an item or value has a value:

➔ *item-or-value* has {a | no} value
    -> *truth-value*

This expression produces a truth-value that indicates whether the specified value-expression has a valid value. For example:

```
the temperature of the most-reliable-measures of tank-1 has a value
```

If the expression the temperature of the most-reliable-measures of tank-1 refers to any variables or parameters, and if those variables or parameters each have a valid current value, then this expression produces the truth-value true. If those referenced variables or parameters have never received a value, or if any of their respective current values have expired, then this expression causes G2 to perform data seeking to obtain new current values. If G2 obtains a value from that data seeking, then the entire expression produces the truth-value true; otherwise, the entire expression produces the truth-value false.

First of the Following Expressions That Has a Value

To determine the first item or value with a value:

➔ the first of the following expressions that has a value
    (*item-or-value* [*item-or-value*...])
    -> {*item* | *integer* | *float* | *symbol* | *text* | *truth-value*}

This expression produces the item or value that is produced by the first expression in the specified list of *item-or-value* expressions that has a valid value.

G2 attempts to evaluate each *item-or-value* expression in the specified list, going from left to right. If the expression being evaluated refers to a variable or parameter that has never received a value or that has an expired current value, then G2 performs data seeking for those variables or parameters. If G2 cannot

obtain a value for any of the specified *item-or-value* expressions, the entire expression produces the truth-value false. For example:

```
the first of the following expressions that has a value
    (the temperature of tank-1,
    average-system-temperature,
    (if the current time >= 10 then 50 else 55))
```

G2 evaluates this expression as follows:

1. If the temperature of tank-1 has a value, G2 produces that value as the value of the entire expression. If not, G2 performs data seeking, as needed, to obtain a new current value for any variables referenced in the the temperature of tank-1 expression.
2. If G2 cannot obtain a new value from evaluating the the temperature of tank-1 expression, G2 next checks average-system-temperature for a valid value. If it has a valid value, G2 produces that value as the value of the entire expression. If average-system-temperature does not have a valid value, and if it is a variable, then G2 performs data seeking to obtain a new current value.
3. If G2 cannot obtain a value for average-system-temperature, the value of the entire expression is either 50 or 55, depending on the G2 clock's current time, as specified in the expression (if the current time >= 10 then 50 else 55).

Current Value of an Expression

Whether or not an expression has a current value depends on whether any item referenced in that expression is a variable and, if so, whether the value of each referenced variable has expired.

For the majority of expressions, the expiration time is the minimum of the expiration time of the values of the expression's terms. As an example, if the expiration time of X is 50, and the expiration of Y is 10, the expiration time of the expression X + Y is 10.

Current Value Of

To determine the current value of an item or value:

➔ the current value of *item-or-value*
-> {*item* | *integer* | *float* | *symbol* | *text* | *truth-value*}

For the specified *item-or-value* expression, a current value exists if G2 can evaluate the expression without finding a new value for the variables, if any, referenced in that expression. If the specified expression has a current value, this entire expression produces it. Otherwise, this expression produces a *no value* condition. For example:

```
if the current value of the temperature of tank-1 exists then ...
```

The expression in this rule's antecedent produces an item or a value if the temperature attribute of tank-1 has a current value. If the temperature of tank-1 does not have a value, is no longer current, or refers to an item that does not exist, then this expression produces a *no value* condition.

Has a Current Value

To determine whether an item or value has a current value:

➔ *item-or-value* has {a | no} current value
-> *truth-value*

This expression produces a truth-value that indicates whether the specified expression has a current value. For the specified *item-or-value* expression, a current value exists if G2 can evaluate that expression without finding a new value for the variables, if any, referenced in that expression.

If this expression produces the truth-value true, its expiration time is the expiration time of the value that the specified *item-or-value* expression produces. If this expression produces the truth-value false, the expiration time is indefinite.

If a has a value expression is nested within a has a current value expression, G2 evaluates the nested expression no differently than a has a value expression. That is, G2 performs data seeking, if necessary, to obtain a new current value for any variables referenced in the nested expression. For example:

```
if the temperature of the most-reliable-measures of tank-1 has a current value
    then ...
```

If G2 can evaluate the expression the temperature of the most-reliable-measures of tank-1 without the need to obtain a new current value for any variables referenced in that expression, then this expression produces the truth-value true. Otherwise, this expression produces the truth-value false.

First of the Following Expressions That Has a Current Value

To determine the first item or value with a current value:

➔ the first of the following expressions that has a current value
(*item-or-value* [i*tem-or-value* ...])
-> {*item* | *integer* | *float* | *symbol* | *text* | *truth-value*}

This expression produces the value of the first *item-or-value* expression in the specified list of expressions that has a current value. For each specified expression, a current value exists if G2 can evaluate the expression without finding a new value for any variables referenced in that expression.

G2 attempts to evaluate each expression in the specified list, going from left to right. If none of the specified expressions has a value, or if none have a current

value, then this expression produces a *no value* condition and no error is signalled. For example:

```
the first of the following expressions that has a current value
    (the temperature of tank-1, average-system-temperature,
        (if the current time >= 10 then 55 else 50))
```

G2 evaluates this expression as follows:

1. If the temperature of tank-1 has a current value, G2 produces that value as the value of the entire expression.
2. If the temperature of tank-1 does not have a current value, G2 next checks average-system-temperature for a current value. If it has a current value, G2 produces that value as the value of the entire expression.
3. If average-system-temperature does not have a current value, the value of the entire expression is either 50 or 55, depending on the G2 clock's current time, as specified in the expression (if the current time >= 10 then 55 else 50).

By Iterating Over a Set

These expressions produce a value by directing G2 to iterate over the set of items specified in a generic reference expression.

Note Your KB's processing cannot depend upon the same order of iteration over a set of items from one evaluation to another of the same expression.

The Count Of

To determine the count of:

➔ the count of each *generic-reference-expression*
[such that (*truth-value-expression*)]
-> *integer*

This expression produces the number of items or values in the set specified in the generic reference expression that also meet the criterion in the specified truth-value expression.

For example, the following expression finds the number of objects upon this executable item's workspace whose number-of-edits attribute is greater than four:

```
the count of each object O upon my-object-ws such that
    ( the number-of-edits of O > 4 )
```

Note You can optimize the execution of a the count of expression if it references indexed attributes. See Defining an Indexed Attribute for more information about indexed attributes.

Also notice that this expression doesn't count for those inactive objects. So a expression like the count of each item will match the result when inspecting show on a workspace every item whose status is active.

The Average Over Each

To determine the average over each:

➔ the average over each *generic-reference-expression* of (*quantity-expression*)
-> *float*

This expression produces a calculated value of type float from values in the set of items or values in the specified generic reference expression. For example:

```
the average over each valve V connected to tank-1 of ( the flow of V )
```

Other Operations Over a Set

To perform operations over a set:

➔ the {sum | product | minimum | maximum}
over each *generic-reference-expression* of (*quantity-expression*)
-> {*integer* | *float*}

This expression produces a calculated value of type quantity from values in the set of quantitative variables, quantitative parameters, or numeric values in the specified generic reference expression.

For example, the following expression computes the minimum of the flows of all valves that are connected to tank-1:

```
the minimum over each valve V connected to tank-1 of ( the flow of V )
```

Referring to the Current Time

These expressions produce values pertaining to time. G2 allows you to access two clocks:

- The real-time clock, a hardware facility of your computer that G2 accesses through the computer's operating system.
- The G2 clock, which is the clock maintained by G2's task scheduler. G2 calibrates the G2 clock to the real-time clock when you select Start from the Main Menu.

The G2 clock runs at the same rate as real time only if you have specified real time in the scheduler-mode attribute of the Timing Parameters system table. If you have set simulated time or as fast as possible in that attribute, the G2 clock might be out of synchronization with the real-time clock. It can be very useful to change this synchronization; see Optimizing Task Scheduling for more information.

Tip You can format the display of values produced from these time expressions. See Formatting Numeric Values.

G2 provides system-defined functions that return time values obtained from either the G2 clock, the real-time clock, or from any numeric expression. See Time Functions for more information.

Current Subsecond Time

To refer to the current subsecond time:

➔ the current subsecond [real] time
-> *float*

This expression produces a time value, represented as a floating-point number of seconds:

- The expression the current subsecond time produces the number of seconds since you started the current KB, using the G2 clock.
- The expression the current subsecond real time produces the number of seconds of since you started the current KB, using the real-time clock.

Note The precision of subsecond time expressions is affected by the setting of the minimum-scheduling-interval attribute of the Timing Parameters system table.

Current Time by Time Unit

To refer to the current time by unit:

➔ the current {time | real time | year | month | day of the month | hour | minute |second}
-> *integer*

This expression produces a time value, as follows:

- the current time expression uses the G2 clock and produces the number of seconds since you started the current KB.
- the current real time expression uses the real-time clock and produces the number of seconds since you started the current KB. Note that this produces

the actual number of elapsed seconds, regardless of the setting of the minimum-scheduling-interval attribute in the Timing Parameters system table.

- the current year expression uses the G2 clock and produces the year, such as 2002.
- the current month expression uses the G2 clock and produces the number (1 through 12) of the current month.
- the current day of the month expression uses the G2 clock and produces the number (1 through 31) of the day of the current month.
- the current hour expression uses the G2 clock and produces a number representing the hour of the current day (0 to 23). Note that the number 0 (zero) indicates 12:00 a.m. midnight.
- the current minute expression uses the G2 clock and produces a number representing the minute of the current hour (0 to 59).
- the current second expression uses the G2 clock and produces a number representing the second of the current minute (0 to 59).

These expressions have an equivalent G2 function. For example, the the current day of the month expression is equivalent to the expression day (the current time) that uses the G2 day function.

Current System Time

To refer to the current system time:

➔ the current system [real] time
-> *float*

This expression produces a time value, as follows:

- the current system time — Returns the current UNIX time, as a float, which is the number of seconds since January 1, 1970, using the G2 clock.
- the current system real time — Returns the current UNIX time, as a float, which is the number of seconds since January 1, 1970, using the real-time clock.

Current Day of the Week

To refer to the current day of the week:

➔ the current day of the week
-> *symbol*

This expression uses the G2 clock and produces one of the following symbols that names a day of the week: sunday, monday, tuesday, wednesday, thursday, friday, saturday.

Referring to Specific Items

Numerous expressions exist for referring to specific items and attributes. Expressions are presented on these topics as follows:

For expressions referring to...	See...
Attributes	Expressions That Refer to Attributes.
Connections items and connections	Using Connection Expressions.
Items	Item Expressions.
Lists and arrays	Lists and Arrays.
Procedures	Expressions for Procedures.
Relations	Expressions Involving Relations.
Rules	Expressions That Refer to Rules.
Variables and parameters	Variable and Parameter Expressions.
Workspaces of items	Expressions That Refer to KB Workspaces,

22

Procedures

Shows how to define, customize, and use G2 procedures.

Introduction

A **procedure** is a predefined sequence of operations that execute sequentially and/or in parallel each time the procedure is invoked. Procedures are convenient when you want to perform the same operations repeatedly under different circumstances and/or on different data values. For a comparison of procedures and rules, see Procedures and Rules.

G2 executes a procedure when the procedure's name and arguments (if any) appear in a call statement or a start action. The procedure executes synchronously when called and asynchronously when started. A called procedure can return one or more values, which are obtained by including the invoking call statement in an assignment statement.

A procedure that returns a value is not the same as a G2 function. Functions have a simpler syntax and are less powerful than procedures, but can be invoked by embedding references to them directly into expressions. Complete information on G2 functions appears in Functions.

This chapter shows you how to define, use, and debug user-supplied G2 procedures. Several other documents and chapters in this document supply related information about procedures:

- G2 provides many system-supplied procedures, as described in the *G2 System Procedures Reference Manual*.
- You can call procedures remotely using G2 Gateway, as described in the *G2 Gateway Bridge Developer's Guide*.
- You can call procedures remotely across a G2-to-G2 interface, as described in G2-to-G2 Interface.

Methods have the same syntax as procedures, but are defined and invoked differently. Methods also provide a locking mechanism. Complete information on creating and using methods appears in Methods.

Procedure Syntax

The syntax of a G2 procedure is similar to the syntax of a procedure in any ordinary programming language. You can write G2 procedures without knowing every detail of their syntax, because the text editor warns you of any syntactic error as soon as it occurs.

A G2 procedure consists of four major parts:

- Procedure header
- Optional local declarations
- Procedure body
- Optional error handler

For example:

procedure header	create-item(class-name: symbol, rank: integer) = (class item)
local declarations	new-item: class item; new-name: symbol;

procedure body	begin
	  create an instance new-item of the class named by class;
	  transfer new-item to this workspace;
	  make new-item permanent;
	  new-name = symbol("[class-name]-[rank]")
	  conclude that the new-names of new-item = new-name
	  return new-item
	end

The formal syntax of a G2 procedure is:

procedure-name ([*argument*: *type*] [, ...]) [= (*type* [, ...])]

[*local-name* [, ...]: *type* [= *value-expression*];] ...

begin
  [*statement-label:*] *statement* [; ...]
end

[on error (*local-name*)
  [*label:*] *statement* [; ...]
end]

The rest of this section describes procedure syntax in detail.

Local Names in Procedures

A **local name** in a procedure is a name that represents an item or value while the procedure executes. A local name has no attributes, collection time, or expiration time, and need not be explicitly deleted when a procedure returns: it has no properties except its type and the item or value that it represents.

Procedure Header Syntax

The **procedure header** names the procedure and specifies what arguments it accepts and the types of values (if any) that it returns. The syntax is:

procedure-name ([*argument*: *type*] [, ...]) [= (*type* [, ...])]

Element	Description
procedure-name	Any name that uniquely identifies the procedure.
argument	A local name to be used within the procedure body to represent a supplied argument. The name shadows any same-named item existing outside the procedure.
type	The type of a local name used as an argument, or of a return value. • If the argument or return value is an item, *type* is the keyword class followed by the class of the item. • If the argument or return value is a value, *type* is the type of the value.

For example, the following header begins a procedure that takes two arguments, a mixer and the name of a part used in it, and returns a truth-value that tells whether that part is currently in stock:

```
check-inventory-for-part (mixer1: class mixer, part: text) = (truth-value)
```

Tip Omitting the keyword class when specifying an argument or return value is one of the most common errors in defining procedures.

A procedure can accept a maximum of 255 arguments and can return a maximum of 255 values.

Duplicate Procedure Names

If a procedure name is not unique, G2 posts a warning in the notes attribute of every procedure that shares the name. Invoking the duplicated procedure invokes an arbitrarily selected instance of it. The selection may differ from one invocation to the next.

Procedures used as methods can have duplicate names provided that the number of arguments differs in each method that shares a name, as described under Duplicate Methods.

Local Declarations Syntax

A procedure's argument definitions (if any) define local names to hold the arguments. Any additional local names needed in the procedure are defined in the **local declarations** section. If no additional local names are needed, the local declarations section is omitted.

Local declarations specify local names, their associated types, and their initial values (if any). The syntax is:

[*local-name* [, ...]: *type* [= *expression*];] ...

Element	Description
local-name	Any name that defined as a local name elsewhere in the procedure. The name shadows any same-named item existing outside the procedure.
type	• If *local-name* represents an item, *type* is the keyword class followed by the class of the item. • If *local-name* represents a value, *type* is the type of the value.
expression	Any expression of the specified *type*. G2 evaluates *expression* to obtain the initial value for the *local-name*.

If more than one local name is of the same type, you can combine the declarations. The following declares three local names of type integer:

```
inventory, products-shipped, orders: integer;
```

You cannot combine local names of the same type if you also need to declare an initial value. You must declare initial values for local names separately, for example:

```
inventory: integer = 0;
products-shipped: integer = 0;
orders: integer = 0;
```

Terminology

In other programming languages, declared names that hold values during procedure execution are called variables. In G2, the term "variable" refers exclusively to a g2-variable, as described in Variables and Parameters.

Variables and parameters can have initial values, but these are not the same as the initial values of local names. Be careful not to confuse the two meanings of "initial value." The context always clarifies which is intended.

Procedure Body Syntax

The procedure body contains one or more procedure **statements**, which specify the operations that the procedure performs. All of the statements in a procedure are enclosed in a **begin-end block**. The syntax is:

```
begin
    [statement-label:] statement [; ...]
end
```

Element	Description
statement-label	An integer or symbol.
statement	Any of the actions described in Actions, or any of the statements described in the Dictionary of Procedure Statements. See Statements.

If a *statement-label* appears, the statement can be the target of a go to, as described under go to.

Statements

Every statement in a procedure is either an action statement or a procedure statement. An action statement specifies some action, using the same syntax that the action uses when specified in a rule. For information on the various actions that a procedure can contain, see Actions.

Procedure statements make assignments, control the flow of execution, and do other things typical of statements in any computer language. The following table summarizes all G2 procedure statements. The Dictionary of Procedure Statements provides complete information about all procedure statements.

Statement	Description
allow other processing	Lets G2 interrupt a procedure to perform other tasks with the same priority.
assignment (=)	Associates a value with a given local name without causing data-seeking.
begin ... end	Defines a compound statement, including the body of a procedure.

Statement	Description
case	Specifies a series of alternative statements to be executed based on the value of an expression
call	Invokes a procedure and transfers control to it.
collect data	Associates a value with a given name by causing data-seeking. Procedures must use this statement when referencing variable values.
do in parallel	Executes two or more statements at the same time.
exit if	Transfers control outside a loop, based on the value of a specified logical expression.
for	Specifies a loop, that is, a sequence of one or more statements to be executed repeatedly until a specified condition has been fulfilled.
go to	Transfers control explicitly to a specified label.
if-then	Specifies conditional execution of statements based on the value of an expression.
on error	Accepts control of the procedure when an error occurs.
repeat	Explicitly iterates over a sequence of one or more statements.
return	Returns control to the calling procedure.
signal	Signals errors that you have named.
wait	Suspends procedure execution for a specified time or based on the value of some expression.

Error Handler Syntax

G2 provides **error handlers**. When errors occur, G2 searches for and invokes the error handler. The default error handler prints a message to the logbook describing the error.

You can also define your own error handlers within procedures by using the on error statement. The syntax is:

```
[on error (local-name)
    [label:] statement [; ...]
end]
```

For details, see on error. For information on all aspects of G2 error handling, see Error Handling.

Comments

You can include comments within the text of a procedure body by enclosing comments within braces:

```
{This is a comment.
This is the second line of the comment.}
```

You can also include single-line comments within the text of a procedure body by beginning the comment line with double slashes:

```
//This is a single-line comment
```

You can place a comment anywhere that whitespace is allowed. Comments are saved with the procedure and do not affect its compilation.

Defining a Procedure

The first step in creating a procedure is creating a blank procedure definition.

To create a procedure definition:

➔ Select KB Workspace > New Definition > procedure > procedure.

To enter statements into a procedure:

➔ Click the mouse on the procedure item, and select the edit menu choice.

When you edit a procedure, you are opening its text attribute for editing. The text attributes of items are described under Identifying the Knowledge in Attributes.

In a procedure's attribute table, the procedure's text attribute appears as one entire row of the table. The procedure's statements appear in this row.

A procedure definition does not have an explicit names attribute. The name of a procedure definition is the name of the procedure itself, which appears at the beginning of the procedure text.

To specify what a procedure does when invoked:

➔ Edit the text of the procedure to specify the desired operations.

You can write G2 procedures without knowing every detail of their syntax, because the text editor warns you of any syntactic error as soon as it occurs. The

warning consists of an ellipsis at the site of the error, and a description of the error in a message posted below the edit box.

If your procedure contains syntactic errors, G2 does not permit you to close the Text Editor until you have corrected them. As soon as you close the Text Editor, G2 compiles the procedure. Such compilation aborts any currently executing invocations of the procedure.

If G2 discovers any problems during compilation, it describes them in the notes attribute of the procedure's table, and posts a message on the Operator Logbook that names the procedure and states that a problem exists in it. After you modify the text of a procedure, be sure to check the logbook or the procedure's notes attribute.

Compiling a Procedure with Error-Location Information

When a procedure in your KB generates a stack error, G2 can tell you what statement in your procedure source code is responsible for the error. G2 does this by retrieving the source-code annotation location information it creates when it compiles your procedure code. You can control the generation of this information by editing the generate-source-annotation-info attribute of the Debugging Parameters system table.

For information on this debugging feature, see Obtaining Procedure Source-Code Error Location Information.

Procedure Attributes

The class-specific attributes of a procedure definition are:

Attribute	Description
tracing-and-breakpoints	Allows you to set tracing and breakpoints on the procedure. See Debugging a Procedure.
Allowable values:	default warning message level tracing message level breakpoint level
Default value:	default

Attribute	Description
class-of-procedure-invocation	Specifies whether G2 automatically creates procedure invocations when a procedure is called or started. Creating Procedure Invocations.
Allowable values:	none procedure-invocation
Default value:	none
default-procedure-priority	Controls the default priority at which G2 executes the tasks associated with the procedure. Setting Procedure Priority.
Allowable values:	Any priority (an integer from 1 through 10)
Default value:	6
uninterrupted-procedure-execution-limit	Limits the *cumulative* amount of time this procedure can run without allowing other processing to occur. Limiting Procedure Execution Time.
Allowable values:	none use default Any *time-interval*
Default value:	use default

Sample Procedure

The following sample shows a complete procedure. This procedure, create-and-move-robot, creates a robot object, transfers it to a workspace, and slowly moves it to a particular place on the workspace. This place is represented by the destination argument, which is defined in the procedure header. The newly created robot will move until it reaches whatever is displayed at that point.

```
create-and-move-robot (destination: class item)

myrobot: class robot;
new-x: quantity = 0;
new-y: quantity = 0;

begin
    create a robot myrobot;
```

```
    transfer myrobot to the workspace of destination;
    repeat
        exit if
          (the item-x-position of myrobot = the item-x-position of destination and
           the item-y-position of myrobot = the item-y-position of destination);
        if the item-x-position of destination > the item-x-position of myrobot
            then new-x = the item-x-position of myrobot + 1
        else
            if the item-x-position of destination < the item-x-position of myrobot
                then new-x = the item-x-position of myrobot - 1;
        if the item-y-position of destination > the item-y-position of myrobot
            then new-y = the item-y-position of myrobot + 1
        else
            if the item-y-position of destination < the item-y-position of myrobot
                then new-y = the item-y-position of myrobot - 1;
        move myrobot to (new-x, new-y);
        wait for 1 second;
    end;
end
```

This procedure uses four local names: destination, myrobot, new-x, and new-y. Destination is declared in the procedure header, as described under Procedure Header Syntax. The other three (myrobot, new-x, and new-y) are used only within the procedure body and are declared in the local declarations part of the definition, as described under Local Declarations Syntax.

This procedure first uses a create action to create a new instance of the class robot called myrobot. It places myrobot on whatever workspace the destination object is displayed, using a transfer action.

- A loop is established to repeatedly increment the display coordinates, moving the robot within the workspace. This is embedded in an if-then statement so that the procedure will end when the destination is reached.
- For each repetition of the loop, a move action moves the robot. Note the use of the wait statement. This slows the loop down so that the movement is visible to the human eye.

Note that this procedure includes references to the item-x-position of destination and the workspace of destination, where destination is an item that is passed to the procedure as an argument.

If you want to change the value of a variable from within a procedure, you must use a conclude action, as described under conclude. You can also use a set action to set the value of a variable in the G2 Simulator or an external data server, as described under set.

The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices.

You can use implicitly declared local names in procedure statements just as you can in expressions generally. For details see Using Local Names in Expressions.

Using Procedures

Using G2 procedures is similar to using procedures in any programming environment. G2 also provides some capabilities beyond those ordinarily available. These allow you to optimize and control procedure execution in various ways, as described later in this section.

Invoking a Procedure

You can invoke a procedure from within another procedure by using either a start action or call statement.

The start action invokes a procedure and runs it asynchronously. It tells G2 to schedule the new procedure for execution, then continue processing the current procedure. You can also use the start action to invoke a procedure from within a rule, a button, or any other context where you can execute an action. For more information about the start action, see start.

The call statement invokes a procedure and runs it synchronously. It tells G2 to invoke the new procedure and wait until the procedure returns before continuing to execute the calling procedure. Call is a procedure statement, *not* an action: you can use it only in the body of a procedure. For more information on the call statement, see call.

Passing Arguments to a Procedure

Whenever you invoke a procedure, you must pass it the correct number and type of arguments. When a procedure is defined, the arguments are specified in the procedure header. For example:

```
plant-record (plant-name: text, inventory-total: quantity)
```

The plant-record procedure has two arguments, whose local names are plant-name and inventory-total. The plant-name is of type text, and inventory-total is of type quantity. You could invoke the plant-record procedure like this:

```
call plant-record ("soap-plant-1", 245)
```

Any argument to a procedure can be given literally or as an expression that evaluates to an item or value of the correct type. Thus you could have specified a text parameter as the *plant-name*, in which case, you would not surround it with double quotation marks. G2 would then use the value of the text parameter as the value of the plant-name.

For example, if soap-1 is a text variable or text parameter whose value is soap-plant-1, then:

```
call plant-record (soap-1, 200+45)
```

is equivalent to the call in the previous example.

Using the Procedure Signature Prompts in the Editor

When you enter a procedure-body statement in the editor that is a call to a procedure or a function, G2 prompts you with the signature of that procedure or function. In the case of multiple methods with the same name, G2 prompts you with the signatures of all methods with that name. G2 does this in the text editor for all items that contain executable code such as rules, function-definitions, formulas, and action-button actions.

G2 prompts by putting up a workspace that displays the argument names for a function; and the argument names, argument types, and return values for a procedure. When you type a defined procedure or function name followed by a left parenthesis, the signature workspace appears in the upper right-hand corner of the G2 window. The workspace remains there as long as the cursor is within the opening and closing parentheses. It automatically disappears when you type the closing right parenthesis, and it reappears when you relocate the cursor within the parentheses.

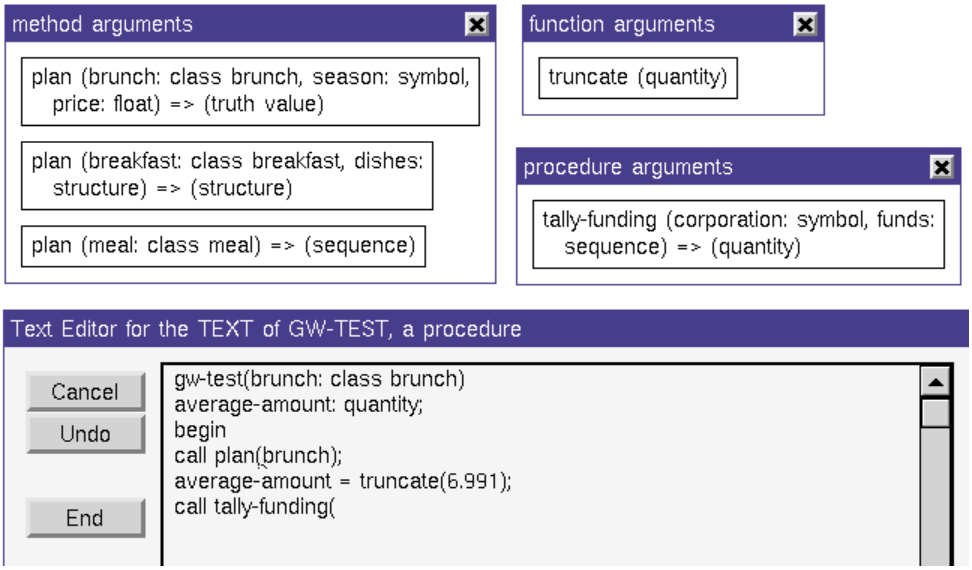
Whether G2 displays procedure and function signatures is controlled by the show-procedure-signatures? attribute on the Editor Parameters system table. The default value of this attribute is true.

To turn off signature prompting in the editor:

➔ Specify no for the show-procedure-signatures? attribute on the Editor Parameters system table.

Here is an example of the text editor open for editing a procedure. The three argument workspaces show sample signatures for a procedure, all the methods named plan, and a system-defined function.

These sample argument workspaces were captured over time. G2 actually displays only one argument workspace at a time and it is for the procedure or function which has the cursor in its argument list.



Accessing Variables in a Procedure

In order to reference the value of a variable, a procedure must use a collect data statement whether or not the variable has a value at the time of the reference. For details, see collect data.

Memory Management in Procedures

When a procedure creates an item, that item is *not* automatically deleted when the procedure exits: the item persists indefinitely until explicitly deleted with a delete action. Therefore, a procedure must delete every item that it creates unless the item is specifically intended to persist and be used elsewhere in the KB.

If this requirement is not met, and undeleted items accumulate without limit, they will eventually consume all memory and abort G2. For additional information, see:

- Failure to Delete Transient Items.
- create.
- delete.

Allowing Other Processing

Concurrent processing during procedure execution can allow other processes to change data used by the procedure, with unpredictable results. For an example, see Allowing Other Processing During List and Array Iteration.

To prevent such problems, by default, G2 executes a procedure without interruption. However, uninterrupted procedure execution prevents G2 from accomplishing any other tasks, such as serving data servers and user interfaces, performing tasks for other procedures and for rules, and so on.

G2 provides two mechanisms for preventing uninterrupted execution from locking out other processing to a harmful extent:

- A procedure can enter a **wait state**. This interrupts procedure execution and allows other processing to occur.
- When a procedure exceeds the time limit for uninterrupted execution, G2 signals an error, as described under Limiting Procedure Execution Time.

Wait States

A procedure can enter a wait state when it executes any of the following statements:

- allow other processing
- call ... across
- collect data
- for each ... do in parallel
- wait
- call when calling a synchronized method

Whether the procedure enters a wait state when one of these statements occurs can depend on the conditions that exist when G2 executes the statement. G2 processes each of the statements as follows:

- **allow other processing**: G2 enters a wait state if the procedure has been running for more than 200 milliseconds; otherwise, it continues executing the procedure. For more information, see allow other processing.
- **call ... across**: G2 cannot predict the response time of the remote system, so it enters a wait state and allows other processing for the duration of the call. The wait state avoids needless suspension of the local G2 or timeout of the calling procedure. For more information, see call.
- **collect data**: If G2 has to wait for one or more values, it allows other processing to occur so that the appropriate variables can receive values. If the variables all have values, G2 continues without allowing other processing. For more information, see collect data.

- **for each ... do in parallel**: G2 enters a wait state between the iteration that launches the parallel iterations and the entrance to each parallel iteration. This enables an arbitrary number of threads to be launched through the iteration without risk of the procedure timing out. For more information, see for and do in parallel.
- **wait**: If the condition for the statement is false or the interval for the statement has not yet passed, G2 allows other processing to occur until the appropriate interval has passed or the conditions of the statement are met. If the condition in the statement is true, G2 continues without entering a wait state. For more information, see wait.
- **call**: G2 allows other processing to occur when using the call procedure statement to call a synchronized method when the first argument to the method is an item that is currently locked by another procedure or method. For more information, see Locking Mechanism for Objects.

Processing During Wait States

The fact that G2 enters a wait state does not guarantee that it will execute all or any scheduled tasks before the wait state ends. If the wait was predicated to end with a particular event, such as a variable obtaining a value or a time interval passing, the procedure can be sure that the specified outcome has occurred, or the wait would not have ended. Beyond that, the procedure should make no assumptions about what did or did not happen while it waited.

Note that certain G2 expressions execute as implicit loops, such as there exists, for every, and conclude that *item* is *relation-name item*. G2 schedules these expressions as a block, rather than scheduling each iteration. As a result, if the implicit loop does not finish executing before the procedure execution limit is reached, an allow other processing statement following such an expression might never be reached. To avoid this problem, break up iteration with long execution times into explicit loops with explicit allow other processing statements.

Using Wait States Cautiously

When a wait state ends and the interrupted procedure resumes execution, the environment in which it executes may have changed in ways that invalidate the procedure's assumptions. Be sure to revalidate the environment as needed before continuing execution of the procedure.

Statements that can cause wait states should not be inserted into procedures without careful consideration of the possible consequences. To prevent concurrency problems, use them only in controlled ways to serve definite purposes.

Limiting Procedure Execution Time

The uninterrupted-procedure-execution-limit attribute sets a limit on the amount of execution time a procedure can use without in some way allowing other processing to occur. The possible values are:

- *An integer and units that represent the time limit*. The default is 30 seconds. The maximum is 24 hours.
- **use default**: The procedure's execution time limit is given by Main Menu > System Tables > Timing Parameters > uninterrupted-procedure-execution-limit.
- **none**: The procedure has no execution time limit.

Caution Specifying none can be dangerous. If the procedure enters an infinite loop, or otherwise fails to return, it will prevent other processing indefinitely, effectively freezing G2. If this happens, you can stop the procedure only by terminating your G2 process from the operating system.

G2 maintains a tally of the cumulative execution time per invocation of each executing procedure. For a given executing procedure, G2 resets this measure only when that procedure, or some procedure called by it, enters a wait state, allowing other processing to occur.

If an executing procedure exceeds its specified execution time limit, G2 signals an error. If no user-defined handler exists for the error, G2 aborts the procedure and posts an error message to the Operator Logbook.

When G2 aborts a procedure that was called by another, it also aborts the caller. Thus procedure timeout in the absence of an error handler aborts the entire procedure stack, not just the procedure that timed out.

For more information, refer to Timing Parameters.

Setting Procedure Priority

The default-procedure-priority attribute controls the default priority at which G2 executes an asynchronously invoked procedure. For information on priorities, see Task Scheduling.

The default priority is 6. You should accept this default unless you have a specific reason to change it. If you need to change it, set the attribute to an integer between 1 and 10 that indicates the priority at which you want the procedure to run.

Three different factors can set the priority at which a procedure executes:

- A called procedure runs at the same priority at which its caller is running, irrespective of the value of its default-procedure-priority attribute.
- A started procedure runs by default at the priority declared in its default-procedure-priority attribute.
- You can override the default priority by including the clause at priority *integer-expression* in the start statement that invokes the procedure. Thus:

  ```
  start plant-record (SOAP-1, 245) at priority 2
  ```

  starts the procedure at priority 2, rather than at the priority given by its default-procedure-priority attribute.

Debugging a Procedure

The tracing-and-breakpoints attribute allows you to set tracing and breakpoints on the procedure. G2 provides three techniques for debugging procedures:

- The notes attribute of a procedure definition provides descriptions of problems involving usage (like an undefined local name) or syntax. You cannot execute the procedure until you correct these errors.
- The tracing-and-breakpoints attribute of a procedure definition allows you to step through the procedure to find problems. This attribute overrides the default settings in the Debugging Parameters system table.
- You can include inform or post actions in your procedure to send messages at important steps in the procedure. For example, you can use inform actions to display the values of local names, or to show what statement G2 is executing.

For information on debugging, see Debugging and Tracing.

Displaying the Invocation Hierarchy of a Procedure

You can use Inspect to display on a workspace:

- All procedures that invoke a given procedure.
- All procedures that are invoked by a given procedure.

Instructions appear under Showing Procedure Caller and Calling Hierarchies.

Inlining a Procedure

A procedure declared as inlineable exists as a separate procedure definition, but its code body is compiled as part of the code of the procedure that calls it. Inlining a procedure improves performance by:

- Avoiding the overhead of a procedure invocation when the inlineable procedure is called. Procedure invocations consume runtime memory.
- Reducing the total number of instructions executed between the calling procedure and the inlined procedure.

However, inlined procedures increase the size of the executable code, because the code for the procedure is copied redundantly to every point where the procedure is invoked. Inlining is best for small procedures that are called frequently, typically from a loop that iterates many times.

Inlining Restrictions

The following situations prevent G2 from compiling the code of an inlineable procedure into a calling procedure:

- The inlineable and calling procedures are not defined in the same module.
- The invocation to the inlineable procedure is asynchronous.
- The invocation to the inlineable procedure is recursive.

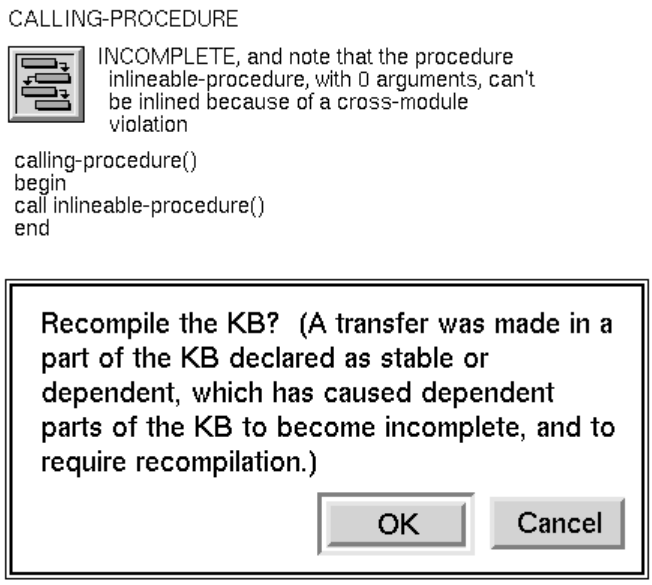
When you compile a procedure that calls an inlineable procedure defined in another module, G2 does not inline the code. Instead, G2 compiles the calling procedure to contain a normal call to the inlineable procedure. This restriction helps protect your KB from unexpected runtime behavior due to intermodual code inconsistencies. Similarly, G2 does not inline asynchronous or recursive invocations to inlineable procedures.

G2 does not give you notification of these failures to inline. See Testing for an Inlined Procedure for information on how to detect whether code has been inlined.

If you have successfully inlined a procedure but then perform a transfer action that places the calling and inlineable procedure definitions in different modules, G2 does the following:

- It changes the status of the calling procedure to incomplete.
- It puts up a Recompile the KB? popup dialog containing these buttons:
  - An OK button. If you click it, G2 compiles the calling procedure *without* inlining.
  - A Cancel button. If you click it, G2 does nothing. The calling procedure is left in an incomplete state.

This example shows the pop-up dialog that appears immediately after a transfer action places the inlineable procedure and the calling procedure in different modules:



To change the incomplete status of a calling procedure:

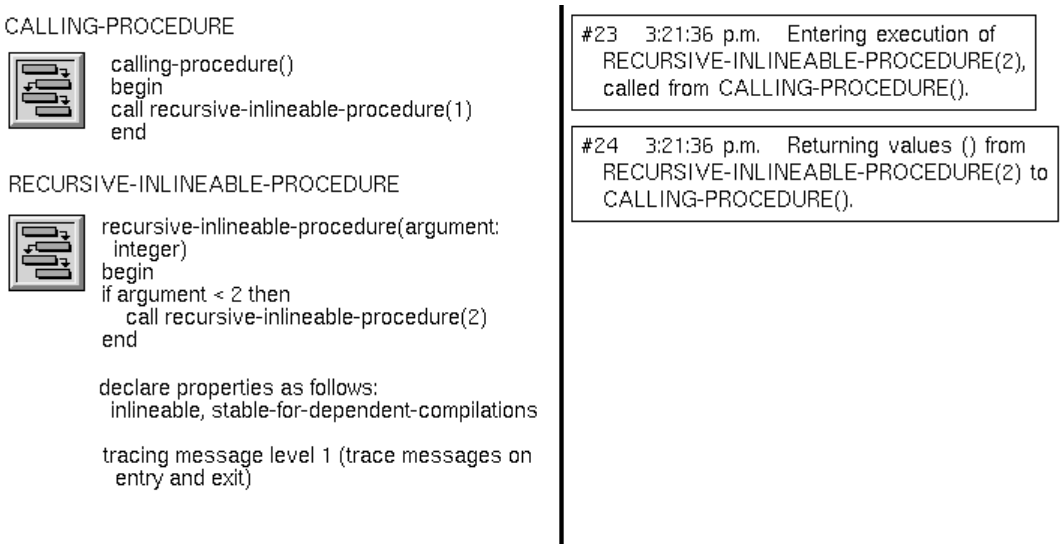
➔ Transfer the calling and inlineable procedures to the same module and recompile *with* inlining.

or

➔ Leave the procedures in different modules and recompile *without* inlining.

The following example illustrates both successful and unsuccessful inlining. The code from recursive-inlineable-procedure has been inlined into the body of calling-procedure because both procedure definitions reside in the same module and the inlineable procedure is invoked through a synchronous call statement rather than an asynchronous start statement.

However, the recursive call statement within recursive-inlineable-procedure does not result in inlining. With tracing on entry and exit specified for the inlineable procedure, the Logbook page shows a single pair of trace messages that designate the invocation as recursive-inlineable-procedure(2) invoked from the inlined code in calling-procedure.



A KB saved in an earlier version of G2 may have procedures that violate the intermodual inlining restriction. When G2 loads a procedure that contains inlined code from an inlineable procedure defined in another module, it first determines whether the inlineable procedure has changed since its caller was compiled; then:

- If the inlineable procedure has not changed, G2 accepts the inlining. The non-conforming inlining is grandfathered in.

  or

- If the inlineable procedure has changed, G2:
  - Changes the status of the calling procedure to incomplete.
  - Adds this note to the calling procedure:

    note that the procedure <*inlineable procedure*>, with <*integer*> arguments, is no longer the same as what was inlined

---

Note You will not be able to remove the incomplete status of a text-stripped procedure because it cannot be recompiled. To restore the procedure to a runnable state, you will need to obtain non-text-stripped versions of both the calling and inlineable procedures.

Declaring a Procedure as Inlineable

When declaring a procedure as inlineable, you must also use the configuration clause: stable-for-dependent-compilations.

To declare that a procedure can be inlined:

➔ Add these item configurations:

```
declare properties as follows : inlineable, stable-for-dependent-
compilations
```

Tip As with other configurations, the stable-for-dependent-compilations configuration statement can be applied to a workspace. Whenever configurations are applied to workspaces, their effects propagate to every applicable item upon the workspace (procedures in this case), and all item subworkspaces.

Recompiling an Inlineable Procedure

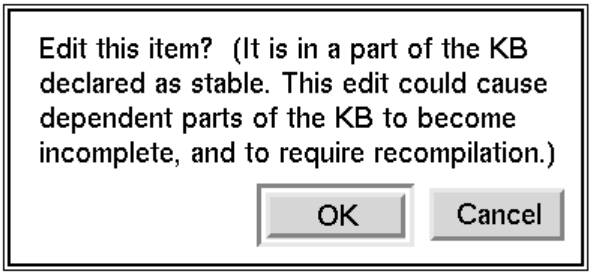
When you change the item-configuration of a procedure to include these properties, the notes attribute of the procedure changes to read:

```
OK, and note that this item needs to be recompiled
to generate data needed for inlining
```

To recompile the inlineable procedure:

1 Choose edit from the item menu of the procedure attribute table.

G2 displays this dialog, because the procedure has been declared as stable:



2 Click OK to continue.

3 Save the edited procedure to recompile it. The procedure notes status will be OK.

Note After making a procedure inlineable and recompiling it, other procedures that call the inlined procedure must be recompiled to incorporate the inlineable code. You can recompile a single procedure by editing it in the Text Editor and saving any changes, or recompile your entire KB using Inspect.

Determining Inlined Procedures

G2 procedures define a new read-only hidden attribute inlined-calls, which is a sequence of symbols that name all procedures that are inlined in this procedure. When the inlined procedures are methods, the symbols are class-qualified. The order of the inlined procedures is undefined. When multiple procedures are inlined, the procedure only appears once in the sequence.

Testing for an Inlined Procedure

You can test whether a procedure has been inlined through the G2 Tracing facility.

To test for an inlined procedure:

1. In the Debugging Parameters system table, change the tracing-and-breakpoints-enabled? attribute to yes.
2. Change the value of the tracing-message-level attribute to:

   1 (trace messages on entry and exit)

3. Run the procedure that calls the inlined procedure. If the inlined code has been incorporated into the calling procedure, the inlined procedure will not appear in the trace messages.

For a further discussion on using these configuration statements, see Using Compilation Configurations.

Creating Procedure Invocations

A **procedure invocation** is an item that represents an instance of an executing procedure. You can put procedure invocations into lists, assign them to participate in relations, and manipulate them as you would any other item.

The class-of-procedure-invocation attribute specifies whether G2 automatically creates a procedure invocation when a procedure is called or started. Possible values are none and procedure-invocation. The default is none.

If the class-of-procedure-invocation is specified as procedure-invocation, G2 automatically creates a procedure invocation each time that procedure is invoked from a start action or call statement. G2 creates a separate procedure invocation each time the procedure is invoked, so a procedure that is recursive or runs concurrently with itself can have more than one procedure invocation at the same

time. G2 automatically deletes a procedure invocation when the procedure returns.

For information on referencing procedure invocations in expressions, see the example The Procedure Invocation Associated with the Procedure Containing the Expression.

Aborting a Runaway Procedure

When a procedure goes into an infinite loop that never enters a wait state, only expiration of the uninterrupted procedure execution time limit can abort it, as described under Limiting Procedure Execution Time. If no such time limit is in effect, the only recourse is to abort G2 itself from the operating system command level.

If the procedure does enter a wait state at some point in the loop, you can use several techniques to abort the procedure:

- Edit the procedure, then close the text editor without making any changes.
- Disable the procedure.
- Deactivate the superior workspace of the procedure.
- Use the abort action on the procedure or procedure invocation.
- Reset the KB.

When you use one of these techniques, G2's response may not be immediate, because G2 cannot process the operation until the runaway procedure enters a wait state.

Caution When you abort a runaway procedure, the KB may be left with undeleted items or incorrect global data that the procedure would have cleaned up if it had exited normally. If any such problem is possible, reset the KB before you rely on it to function correctly.

Expressions for Procedures

These are the expressions to use with procedures.

The Procedure Containing the Expression

this procedure
-> *procedure*

This expression produces the procedure within whose invocation G2 evaluates this expression. For example, the announce-self procedure creates and places a message upon its workspace, then deletes that message 10 seconds later:

```
announce-self ( )
M : class message ;
begin
     create a message M ;
     change the text of M to
       "[the name of this procedure] is now being performed.";
     transfer M to the workspace of this procedure ;
     wait for 10 seconds ;
     delete M ;
end
```

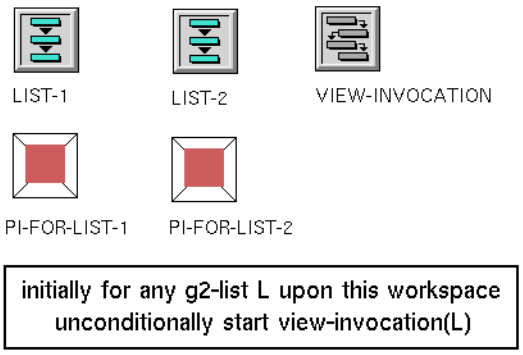
The Procedure Invocation Associated with the Procedure Containing the Expression

this procedure-invocation
-> *procedure-invocation*

This expression produces the executing procedure invocation within which G2 evaluates this expression.

Specify this expression only in the text of a procedure or method whose class-of-procedure-invocation attribute has the value procedure-invocation. Invoking such a procedure causes G2 to create a procedure invocation item. For more information about procedure invocations, see Creating Procedure Invocations.

For example, the next figure shows a workspace that contains two lists, an initially rule, and the view-invocation procedure. The initially rule starts the view-invocation procedure once for each list that exists.




```
view-invocation(list: class g2-list)
PIcount: integer = the count of each procedure-invocation;
counter: integer;

begin
     transfer this procedure-invocation to the workspace of list
```

```
        at (the item-x position of list, the item-y-position of list - 50);
    conclude that the names of this procedure-invocation =
        symbol("PI-for-[the name of list]");
    for counter = 1 to 5 * the count of each g2-list do
        begin
            change the icon-color of this procedure-invocation to yellow;
            wait for (2 * PIcount);
            change the icon-color of this procedure-invocation to blue;
            wait for (2 * PIcount)
        end
    end
end
```

Each invocation of view-invocation causes G2 to create one procedure-invocation. view-invocation transfers this procedure-invocation to the workspace of the list passed as its argument and graphically aligns its icon with that list's icon. Once per second for several seconds, view-invocation changes the icon-color of this procedure-invocation to indicate that it is active.

Procedures and Rules

Procedures and rules have much in common, but their purposes are different, and their syntax differs slightly. This section compares and contrasts procedures with rules:

- Use procedures to perform an explicit series of operations and to control the flow of events. Example: to perform the steps involved in starting up or shutting down a plant.
- Use rules to monitor asynchronous events and to detect or anticipate problems that might occur. Example: to watch for conditions that exceed specified limits.

Rules are better than procedures for monitoring events because they do not consume resources with busywaiting as procedures would.

Within a procedure, you control exactly when G2 waits and allows other processing to occur. This differs from the way rules operate, because you cannot control when a rule waits for a value. As a result, you are guaranteed that G2 executes the tasks for a procedure without interruption, except where you indicate in the procedure that other processing can occur.

Because a procedure only waits at specific points, all of the expressions in it must have current values. Data-seeking implies waiting and allowing other processing to take place, so a procedure does not cause data-seeking unless you instruct it to do so by using a collect data or wait statement. This is another difference between procedures and rules: rules automatically manage data collection and waiting for values, while procedures do not.

The following table summarizes the major differences between procedures and rules:

Capability	Procedures	Rules
May cause event updating (forward chaining)		✓
Can only seek data explicitly	✓	
Always allows other processing to take place while it waits for a value		✓
Explicitly allows other processing to take place while it waits for a value	✓	
Automatically manages data collection and waiting for values		✓
May call functions	✓	
May start procedures (so that they run in parallel)	✓	✓
May call procedures (so that they run in sequence)	✓	
Can contain actions	✓	✓

The following table shows minor differences between procedures and rules that can cause trouble when they are written. Action buttons use the same syntax that rules do, so the table applies to them also.

Rules/Action Buttons	Procedures
Procedures can only be started.	Procedures can be started or called.
No control structures are available.	All standard programmatic control structures are available.
Default: parallel execution. Use in order to specify sequential execution.	Default: sequential execution. Use do in parallel to specify parallel execution.
Local names are defined by context, for example: `for any bottle B ...`	Local names are explicitly declared in the local declarations section, for example: `B: class bottle;`

Rules/Action Buttons	Procedures
Iterate over items using every, for example: `move every bottle ...`	Iterate over items using each, for example: `for B = each bottle do ... end;`
Separate actions with and.	Separate actions and other procedure statements with semicolons.

Dictionary of Procedure Statements

The rest of this chapter lists all procedure statements in alphabetical order, and provides complete information about each one. For information on the various action statements that a procedure can contain, see Actions.

allow other processing

This statement lets G2 interrupt a procedure (enter a wait state) to perform tasks, to service networks, the user interface, the G2 clock, and other current processes, such as rules and procedures, with the same priority. See Allowing Other Processing for more information. The syntax is:

allow other processing

When G2 encounters an allow other processing statement, it first checks to see how long the procedure has been running. If it has been running for more than 200 milliseconds, G2 puts the procedure at the end of the list of tasks with the same priority on the current task queue and proceeds to execute the other tasks on the queue.

Note The 200 millisecond time limit does not include the time that G2 waits for a called procedure to run.

If the procedure has not been running for more than 200 milliseconds, G2 continues processing the procedure. Thus, G2 will not interrupt the procedure, even if there is an allow other processing statement, if the procedure has not used more than 200 milliseconds of processing time.

Caution When a procedure returns from a wait state, the environment in which it executes may have changed in ways that invalidate the procedure's assumptions. Be sure to revalidate the environment as needed before continuing execution of the procedure.

assignment (=)

This statement assigns a value to a local name in a procedure. The syntax is:

local-name = *expression*

When the statement is executed, the *expression* is evaluated and that value is associated with the *local-name* to the left. The association persists until another assignment occurs. For example:

```
weight = 200
.
.
weight = weight+1
```

Here the first assignment sets the value of weight to 200. The value remains 200 until the subsequent assignment increments its value to 201.

The *expression* can be a call statement to a procedure that returns a value. That value is assigned to *local-name*:

```
weight = call get-weight (height)
```

For further information, including the syntax for obtaining multiple return values, see call.

An assignment statement *does not* cause data-seeking so it cannot contain any references to G2 variables. Use the collect data statement to assign the value of a variable to a local name, as described under collect data.

begin-end

You can specify a block of two or more statements wherever a single statement can appear by enclosing the statements in a begin-end statement. This construction is also called a **compound statement** or a **begin-end block**. The syntax is:

```
begin
    [label:] statement [; ...]
end
```

The body of a procedure is enclosed in a begin-end block. Such blocks can be nested to arbitrary depth. Any begin-end blocks can have an associated error handler, as described under on error.

A begin-end statement must end with a semicolon whenever a single statement in that location would do so, unless it is the value of the then clause of an if-then statement. For further information, see if-then.

call

This statement allows you to invoke another procedure synchronously from within the current procedure. The calling procedure suspends execution, and remains suspended until the called procedure returns. The calling procedure then resumes execution. The syntax is:

[*local-name* [, ...] =] call *procedure* ([*argument* [, ...]])
    [across {*g2-to-g2-interface* | *gsi-interface*}]

where:

local-name	A local name in which G2 places a return value of the called procedure. The number of local names must not exceed the number of values returned by the called procedure. See return for details on providing return values.
procedure	A procedure name or a reference to a procedure name, such as the procedure that is the-open-procedure-of tank-1.
argument	An item or value used by the specified procedure. Specify multiple arguments by separating them with commas.
across	Designates the name of a G2-to-G2 interface or a G2 Gateway (GSI) interface that G2 uses to call a procedure that is running on another G2 process.

Note A called procedure runs at the same priority as that of the calling procedure, *not* at the priority declared in its default-procedure-priority attribute.

Here is an example of how to use the call statement to invoke the procedure check-inventory-for-part described in Procedure Header Syntax. The procedure is defined as follows:

```
check-inventory-for-part (mixer1: class mixer, part: text) = (truth-value)
```

The example declares two local variables. In the procedure call, the ingredient-1 of recipe1 is a text attribute of recipe-1.

```
is-in-Inventory: float;
mixer1:class mixer;

is-in-Inventory = call check-inventory-for-part(mixer1, the ingredient-1
    of recipe-1)
```

Calling and Wait States

When one procedure calls another in the local G2, the calling procedure does *not* enter a wait state. The called procedure may or may not enter a wait state, depending on its code. For information about wait states see Allowing Other Processing.

However, in a call ... across, G2 cannot predict the response time of the remote system, so it enters a wait state and allows other processing for the duration of the call. The wait state avoids needless suspension of the local G2 or timeout of the calling procedure.

Caution When a procedure returns from a wait state, the environment in which it executes may have changed in ways that invalidate the procedure's assumptions. Be sure to revalidate the environment as needed before continuing execution of the procedure.

Asynchronous Invocation

To invoke a procedure asynchronously, use the start action, as described under start.

case

This statement branches to a statement based on the value of an expression. After the selected statement executes, control passes to the first statement after the case statement. The syntax is:

```
case (value-expression) of
    {case-tag [; ...]: statement [; ...] } ...
    [otherwise: statement]
end
```

where:

value-expression	Any expression that returns any type. Its value determines how the case statement branches.
case-tag	A quantity, text value, or symbol. The type must match that of *value-expression*.
statement	Any procedure or action statement.
otherwise	Designates a statement to execute if the value of *value-expression* does not match any *case-tag*.

If no match exists, control passes to the otherwise statement if one is provided. If none is provided, G2 signals an error. For example:

```
case (the icon-color of light) of
    red:
        conclude that power is off;
    yellow, orange:
        begin
            conclude that power is low;
            invoke safety-rules
        end;
    green:
        conclude that power is high;
    otherwise:
        begin
            conclude that power is off;
            inform the operator that
              "The light is [the icon-color of light]. Turning power off";
            invoke emergency-rules
        end;
end
```

In this example, if the icon-color of light is red, G2 executes the statement specified by the red case-tag; however, if it is yellow, G2 executes the statements specified

by the yellow case-tag. If the icon-color of light is something other than red, yellow, or green, G2 executes the statements specified by otherwise; however, if otherwise were omitted, G2 would signal an error.

collect data

This statement assigns a value, based on an expression containing G2 variables, to a local name. You can access variables *only* with a collect data statement: you cannot reference them directly, as you can a local name. The syntax is:

```
collect data [ (timing out after time-expression) ]
    local-name = value-expression [; ...]
    [; if timeout then statement] [;]
end
```

where:

timing out after *time-expression*	Limits the time that G2 spends seeking values to the specified *time-expression*.
local-name	Is any local name defined within the procedure.
value-expression	Is any expression containing references to variables (*not* parameters).
if timeout then *statement*	Designates a statement that G2 executes if it times out while evaluating the specified expression(s).

Note You *cannot* access parameters with collect data; you can access only variables. The syntax for accessing a parameter is the same as the syntax for referencing a local name.

Executing a Collect Data Statement

When G2 executes a collect data statement, it first attempts to evaluate each expression in the statement. If all of the variables in the expressions have current values, G2 evaluates the expressions and assigns the values to the appropriate local names. It then proceeds to the next statement in the procedure.

If G2 cannot immediately evaluate an expression because a variable does not have a current value, G2 enters a wait state (suspends procedure execution and waits for a value). For information about wait states see Allowing Other Processing. Depending on the characteristics of the variable, G2 may data-seek for values. Procedure execution resumes when the variable receives a value or the statement times out, assuming you have specified a timeout interval.

When the procedure resumes, G2 reevaluates all of the expressions in the collect data statement, as follows:

- If all of the expressions now have values, G2 assigns those values to the local names. It then proceeds to the next statement in the procedure.
- If any of the expressions does not have a value and the statement has not yet timed out, G2 suspends the procedure and waits.
- If any of the expressions does not have a value and the statement has timed out, G2 assigns the values of any expressions that it was able to evaluate to the appropriate local names. It does not change the values of the other local names in the statement.
- If you include the if timeout then syntax, G2 executes that syntax before resuming procedure execution after a timeout.

Tip To make a collect data statement that cannot cause a wait state, use timing out after -1 seconds. G2 then makes one attempt to gather values, but times out without waiting and allowing other processing if some values are not available.

Caution When a procedure returns from a wait state, the environment in which it executes may have changed in ways that invalidate the procedure's assumptions. Be sure to revalidate the environment as needed before continuing execution of the procedure.

do in parallel

This statement allows G2 to execute concurrently all statements that appear between do in parallel and end. The do in parallel statement is very useful when the statements are waiting for conditions in an external system to complete, such as with remote procedure calls. The syntax is:

```
do in parallel [until one completes]
    statement [; ...]
end
```

where:

until one completes	Specifies that G2 stops simultaneously processing the statements when any one of the statements completes. G2 continues executing the procedure, beginning with the statement immediately following the do in parallel block.
statement	Any action or procedure statement.

You cannot use a go to statement to transfer control from one statement to another within a do in parallel block, or into the block from outside.

Do in Parallel and Wait States

A do in parallel statement does not in itself cause a procedure to enter a wait state. However, executing a statement of the form for each ... do in parallel causes a procedure to enter a wait state between the iteration that launches the parallel iterations and the entrance to each parallel iteration. This enables an arbitrary number of threads to be launched through the iteration without risk of the procedure timing out.

For information about wait states see Allowing Other Processing. The for statement is described under for.

Caution When a procedure returns from a wait state, the environment in which it executes may have changed in ways that invalidate the procedure's assumptions. Be sure to revalidate the environment as needed before continuing execution of the procedure.

Using Do in Parallel Effectively

Since the computers on which G2 runs cannot actually do more than one thing at a time, do in parallel offers no speed advantage unless the execution of individual threads is dependent on some external event. For example, the do in parallel in the following is counterproductive:

```
for T = each tank do in parallel
    conclude that the computed-thing of T = compute-thing (T);
end;
```

The do in parallel slows down the application instead of speeding it up, because the body of the loop consists entirely of computations directly within G2. The launching and resynchronizing of all the parallel threads just wastes time. On the other hand, the following is improved by the do in parallel.

```
for T = each tank do in parallel
    thing = call remote-compute-thing (T) across my-interface;
    conclude that the computed-thing of T = thing;
end;
```

In this case, since remote-compute-thing is executed on a remote machine, the loop is improved, if only to reduce the impact of communication latencies on the performance of the loop.

Concurrency and Asynchrony

Don't confuse the concurrency provided by do in parallel with the asynchronous execution provided by start. For example, consider the statements:

```
do in parallel
    call race-car (racer-1);
    call race-car (racer-2);
    call race-car (racer-3);
end
call award-prizes ( );
```

The call to award-prizes will not execute until all three races have finished. Presumably award-prizes uses global data of some kind to obtain the outcomes of the races. However, consider the statements:

```
start race-car (racer-1);
start race-car (racer-2);
start race-car (racer-3);
call award-prizes ( );
```

The call to award-prizes will fail, because none of the invocations of race-car can execute until after the procedure that scheduled them has exited: award-prizes will find no data about outcomes, because the races that will generate it have not yet been run.

exit if

This statement causes G2 to exit from a loop before it would normally terminate. The syntax is:

exit if *truth-value-expression*

where:

truth-value-expression	A logical expression that evaluates to true or false.

If the *logical expression* is true, G2 executes the exit if statement and transfers control to the statement immediately following the loop. If false, G2 does not execute the exit if statement and continues processing the loop.

When G2 encounters an exit if statement within a nested loop, its effect is to transfer control back to the loop immediately enclosing it. For example:

```
for var-1 = 1 to 5
    do
        for var-2 = 1 to 5
            do
                exit if var-2 = 3
            end;
        statement-1;
    end;
statement-2;
```

In the above example, when G2 executes the exit if statement (when var-2 has the value 3), it will continue processing with *statement-1*, rather than *statement-2*.

for

This statement instructs G2 to repeatedly execute a statement or sequence of statements enclosed by do and end. You determine the terminating conditions. You can execute for statements for each instance of a class of items, for each item or value in a list or array, or based on the value of a numeric expression.

Iterating over Each Instance of a Class of Items

You can use the for statement with the each quantifier to execute a sequence of statements once for each instance of a class of items. The syntax is:

```
for local-name = each generic-reference-expression
    do
        statement [; ...]
    end
```

where:

generic-reference-expression	Specifies any generic reference to an item or value that G2 iterates over in the set.
statement	Any procedure or action statement.

The for loop terminates when all the instances of the specified class have been processed. For example:

```
for V = each valve
    do
        change the icon-color of V to red;
    end
```

This example changes the icon-color used for each instance of the class *valve.* To limit the iteration, you could use the following:

```
for V = each valve connected to tank-1
    do
        change the icon-color of V to red;
    end
```

In this example, only instances of valves connected to tank-1 change color.

Iterating Using a Counter

You use a loop variable as a counter to iteratively execute the statements in the body of the for loop. The number of iterations is determined by the initial value of the loop variable, the amount the loop variable is updated at the end of each iteration, and the specified loop termination control value.

The syntax is:

```
for loop-variable = initial-value-integer {to | down to}
    termination-control-integer
 [by loop-variable-pdate-integer]
    do
        statement [; ...]
    end
```

where:

loop-variable	A local integer variable that is initialized to *initial-value-integer* and is incremented or decremented at the end of each iteration of the for loop.
initial-value-integer	An expression that evaluates to an integer. It supplies the initial value for *loop-variable*.
to	When you omit the optional by phrase, to specifies that *loop-variable* is, by default, incremented by 1 at the end of each iteration. If you add the by phrase, it overrides the default incrementing behavior.
down to	When you omit the optional by phrase, down to specifies that *loop-variable* is, by default, decremented by 1 at the end of each iteration. If you add the by phrase, it overrides the default decrementing behavior.

loop-termination-control-integer	An expression that evaluates to an integer. At the end of each iteration, G2 compares its value with the value of the *loop-variable* to determine whether the for loop should continue or should exit. When you use the to phrase, the loop exits when *loop-variable* exceeds its value; when you use the down to phrase, the loop exits when *loop-variable* is less than its value.
by	Overrides the default incrementing and decrementing behavior of the to and down to phrases. Regardless of whether you specify to or down to, the value of *loop-variable-update-integer* is added to *loop-variable* at the end of each iteration. A negative value will decrement *loop-variable*, and a positive value will increment *loop-variable.*
loop-variable-update-integer	An expression that evaluates to an integer. It determines the amount *loop-variable* is updated at the end of each iteration.

The for loop exits and passes control of execution to the next statement in the procedure when:

- The updated value of *loop-variable* is beyond the range of *loop-termination-control-value*.
- The body of the for statement contains an exit if statement that evaluates to true. See exit if for more information.

Caution Your procedure will enter an infinite loop when you specify an update integer that prevents the for loop from exiting.

Here is a for loop that will exit after six iterations:

```
for loop-variable = 0 to 5
    do
        .
        .
        .
    end
```

This for loop will also exit after six iterations:

```
for loop-variable = 0 to 10 by 2
    do
        .
        .
        .
    end
```

In this example, the for loop will enter an infinite loop. If the exit if statement is uncommented and the procedure is recompiled, it will exit after six iterations:

```
for loop-variable = 10 down to 0 by 2
    do
        .
        .
    { exit if loop-counter = 20 }
    end
```

For Each and Wait States

A for each statement does not in itself cause a procedure to enter a wait state. However, executing a statement of the form for each ... do in parallel causes a procedure to enter a wait state between the iteration that launches the parallel iterations and the entrance to each parallel iteration. This enables an arbitrary number of threads to be launched through the iteration without risk of the procedure timing out.

For information about wait states see Allowing Other Processing. The do in parallel statement is described under do in parallel.

Caution When a procedure returns from a wait state, the environment in which it executes may have changed in ways that invalidate the procedure's assumptions. Be sure to revalidate the environment as needed before continuing execution of the procedure.

Transferring Objects in a Loop

In the following example, the first time through the loop, the object is transferred off the workspace and placed in the item list of a container object. The second time through the loop, the same object is placed in the item list of a second container object. Thus, an object that has already been removed from a workspace appears in the item list of two container objects.

```
for O = each myObject upon WS do
    transfer O off;
    create containerObj C;
    insert O in the items item-list of C;
    for T = each myObject upon WS do
        if the destination of T = the destination of O then begin
            transfer T off;
            insert O in the items item-list of C;
        end;
    end;
end;
```

go to

This statement explicitly transfers control to a statement with a specified label. The syntax is:

go to *statement-label*

where:

statement-label	An integer or symbol that labels some statement in the procedure.

The go to statement specifies the label to which G2 transfers control. The specified label must appear in the procedure. If it does not, G2 signals an error. Specifying a label requires the following syntax:

statement-label: *statement*

For example:

```
new-proc (quant1: quantity, quant2: quantity) = (integer)
x, y: integer;
begin
    if x > 4
        then go to tag5
    else
        return
    tag4: x = 8;
    tag5: x = y;
end
```

In this example, G2 transfers control of the procedure to tag5 if x is greater than 4, and returns otherwise. Note that G2 never executes the statement labeled tag4.

if-then

This statement is a conditional statement that executes the then statement if G2 evaluates the *truth-value-expression* to true; otherwise, it executes the else statement, if one exists, or the statement following the if-then, if no else exists. The syntax is:

```
if truth-value-expression
    then statement
    [else statement]
```

where:

truth-value-expression	Is any expression that evaluates to true or false. If the expression is true, G2 executes the then statement. If false, G2 executes the else expression if one exists, or transfers control to the statement following the if statement.
statement	Is any procedure or action statement.

For example:

```
if x = y
    then return 10 else return 12
```

In this example, the program returns 10 if x equals y; however, if x does not equal y, the program returns 12.

You can nest if-then statements by enclosing the nested if-then statement in a begin-end block. For example:

```
if x = y then
    begin
        if x = z then
            return 10
        else
            return 12
    end
else ...
```

If x is not equal to y, G2 ignores the nested if-then statement and passes control to the else portion of the outer if statement. If x does equal y, G2 executes the nested if-then statement.

Note When a then clause contains a begin-end block immediately followed by an else clause, do not put a semicolon after the keyword end. Such a semicolon would terminate the scope of the if statement, causing the else clause to appear as a syntax error.

on error

G2 provides **error handlers**. The default error handler prints a message to the logbook describing the error. You can use the on error statement to define an error handler that is specific to any begin-end block in a procedure, including the outer block that contains all of the executable code.

For complete information on G2 error handling, see Error Handling. For information about the signal statement, see signal.

G2's default error handling capabilities are synchronous: they do not enter a wait state during handling of an error. This protects the context within which the error occurred from asynchronous changes. User-defined error handling capabilities can allow other processing if appropriate. For information on wait states, see Allowing Other Processing.

On Error Statement Syntax

The syntax of the on error statement is:

```
on error (local-name)
    [label: ] statement [; ...] ...
    end
```

where:

local-name	A local name whose type is class error or any subclass of error.
statement	Is any procedure or action statement.

An on error statement appears immediately after the end statement of the block to which it applies. For example:

```
demonstrate-block-error-handler()
error-obj: class error;
begin
    post "calling sigproc now";
    call sigproc(0);
    post "returning from sigproc:"
end
on error (error-obj)
    post "An error of class [the class of error-obj]
        occurred: [the text of the error-description of error-obj]";
    delete error-obj
end
```

An on error statement executes if and only if G2 signals an error within the scope of the statement, or a signal statement executes within the scope of the statement. Otherwise, control skips over the on error block and continues sequentially.

For complete information about the on error statement, see Defining an Error Handler.

Superseded On Error Statement Syntax

For compatibility with earlier versions of G2, the on error statement also accepts two arguments. The syntax is:

```
begin
    statement [; ...]
end
on error (symbolic-local-name, text-local-name)
    [statement [; ...] ]
end
```

where:

symbolic-local-name	A local name that contains a symbol. You must declare this name locally in your procedure.
text-local-name	A local name that contains text. You must declare this name locally in your procedure.

For example:

```
sample-proc()
error-name: symbol;
error-text: text;
begin
    post "This is a sample statement within a begin-end block";
    call proc-with-signal();
end
on error (error-name, error-text)
    post "An [error-name] error occurred. [error-text]";
end
```

The two-argument form of the on error statement should not be used in new code. A KB can mix both forms of the statement, and can use either form in conjunction with either form of the signal statement. G2 automatically interconverts between the two syntaxes, as described under Mixing Error Handling Techniques.

repeat

This statement causes G2 to repeatedly execute a statement or set of statements indefinitely. The grammar for a repeat statement is as follows:

```
repeat
    statement [; ...]
end
```

To exit the loop, you can embed a statement that alters the pattern, such as exit if, return, or go to.

return

A return statement tells G2 to exit the procedure and return values to the calling procedure. This is required when the procedure returns values, since at that time the values are assigned to the specified symbols. The syntax is:

return [*value-expression* [, ...]

The number and types of the values returned must match the return values definition in the procedure header. See Procedure Header Syntax for more information.

The calling procedure may accept as many of the returned values as it needs; however, it may not attempt to accept more values than are returned. For details see call.

signal

You can signal errors that you have defined by using the signal statement. By using the signal statement, you can signal errors that you have named, then use the on error statement to create an error handler for those named errors. The signal statement can be used anywhere within a procedure.

For complete information on G2 error handling, see Error Handling. For information about the on error statement, see on error.

G2's default error handling capabilities are synchronous: they do not enter a wait state during handling of an error. This protects the context within which the error occurred from asynchronous changes. User-defined error handling capabilities can allow other processing if appropriate. For information on wait states, see Allowing Other Processing.

Signal Statement Syntax

The syntax of the signal statement is:

signal *error-object*;

where:

error-object	An instance of the class error or of any subclass of error.

For example:

```
sigproc(index: integer)
ZD: class zerodivide;
begin
    create a zerodivide ZD;
    conclude that the error-description of ZD = "Cannot divide by zero.";
    if index = 0 then signal ZD;
    post "ratio: [45387 / index]
end
```

When a signal statement executes, G2 looks for a block error handler whose class matches that of the *error-object* specified in the statement. If G2 finds such a handler, it invokes the handler, passing it *error-object*. If G2 does not find a block error handler, it invokes the default error handler on *error-object*.

For complete information about the signal statement, see Signaling Errors in a Procedure.

Superseded Signal Statement Syntax

For compatibility with earlier versions of G2, the signal statement also accepts two arguments. The syntax is:

signal *symbolic-expression*, *text-expression*

For example:

```
proc-with-signal()
begin
    if x > 100
        then signal the symbol OVERFLOW,
          "X appears to be overflowing --check X immediately!";
end
```

The two-argument form of the signal statement should not be used in new code. A KB can mix both forms of the statement, and can use either form in conjunction with either form of the on error statement. G2 automatically interconverts between the two syntaxes, as described under Mixing Error Handling Techniques.

wait

This statement suspends a procedure's execution, causing it to enter a wait state, until either a specified amount of time passes, a specified condition is met, or an event occurs. During this time, other processing may take place. For information about wait states see Allowing Other Processing. The syntax is:

```
wait { {for time-expression} |
    {until truth-value-expression checking every time-expression} |
    {until {variable | parameter | {the attribute-name [local-name] of item} |
    {the {class-name | type} that is an attribute of item [named by
    symbolic-expression] } receives a value}
```

where:

for *time-expression*	Specifies that G2 suspends procedure execution for the specified interval.
until *truth-value-expression*	Specifies that G2 suspends procedure execution until a *truth-value-expression* is true. Note that the *truth-value-expression* may contain references to G2 variables and may data-seek for values.
checking every *time-expression*	Species how often G2 checks the value of the specified *truth-value-expression*. The *time-expression* is an expression that returns seconds.

The second until clause defines an **event predicate**. When you specify an event predicate in a wait statement, the statement suspends procedure execution until the event predicate becomes true. An event predicate may contain boolean operators. For example:

```
repeat
    exit if the temp of tank-1 > the high-limit of tank-1;
    wait until the temp of tank-1 receives a value or
      the high-limit of tank-1 receives a value;
end
```

This form of the wait statement does not consume any CPU resources while it waits for an event to occur. If any of the items for which the event predicate awaits a value is deleted, the wait statement terminates, and control proceeds to the next statement.

Caution When a procedure returns from a wait state, the environment in which it executes may have changed in ways that invalidate the procedure's assumptions. Be sure to revalidate the environment as needed before continuing execution of the procedure.

23

Methods

Shows how to define and use G2 methods.

Introduction

Object-oriented programming treats the data and the behavior associated with a class as parts of a single abstraction. G2 uses attributes to contain the data associated with a class, and methods to define the behavior. For information on attributes, see Identifying the Knowledge in Attributes.

This chapter does not cover the theory that underlies methods in object-oriented programming languages. It focuses on practical techniques for designing and implementing methods in G2. For information on the theory of methods, consult a standard text on object-oriented programming.

In order to understand, design, and use methods, you must understand class hierarchy, class inheritance, and class hierarchy paths, as described in Classes and Class Hierarchy. In order to code methods, you must know how to code procedures, as described in Procedures.

About Methods

In G2's object-oriented programming language:

- An **operation** is a function or transformation that can be applied to items.
- A **method** is a specialized procedure that implements an operation for items of a particular class.
- An **operand** is an item on which a method performs an operation.

Methods allow you to customize operations in class-specific ways. In conjunction with class hierarchy, methods allow you to define item behavior with great economy and modularity. Two essential capabilities provide these advantages:

- When you perform an operation, you perform it in the same way regardless of the class of the operand. G2 automatically invokes the correct method for an operand of that class.
- A method defined for a class can invoke the method defined for a superior class by executing a call next method statement. G2 automatically calls the correct method based on the structure of the class hierarchy.

Methods and Procedures

Methods are syntactically and functionally similar to procedures, and procedures can do anything that methods can do. However, methods are typically more convenient than procedures for defining complex behavior, because they are more modular, flexible, maintainable, and reusable.

G2 methods have essentially the same syntax as ordinary G2 procedures. Both procedures and methods:

- Have names.
- Can take arguments and return values.
- Contain statements and actions.
- Can be invoked by call or by start.

The only syntactic differences are:

- The first argument to a method must be of the class to which the method applies.
- A method can execute call next method, while an ordinary procedure cannot.

Thus any procedure that has the first property listed could be used as a method without changing its code in any way. The essential difference between procedures and methods is not in their code, but in the way G2 invokes them.

The Vessel Example

A simple example can demonstrate all the essential features of methods, and show you how they compare with procedures. Suppose that:

- Your KB defines the class vessel, with three subclasses: tank, bottle, and flask.
- You need to fill tanks, bottles, and flasks at various times as your KB executes.
- Before filling a vessel, you must prepare it in a way that depends on its class:
  - tank: Unscrew the tank's cap.
  - bottle: Remove the bottle's cork.
  - flask: Sterilize the flask.
- After preparing the vessel, you fill it in exactly the same way regardless of its class.
- After filling the vessel, you screw on its cap, replace its cork, or do nothing, depending in its class.

The rest of this chapter uses this example at various points to illustrate the properties of methods.

Filling Vessels Using Procedures

To fill vessels by using procedures, you could create a procedure for each of the four classes. For example:

fill-vessel (V: class vessel)	Fill the vessel (tank, bottle, or flask).
fill-tank (T: class tank)	Unscrew the tank's cap, invoke fill-vessel to fill the tank, then replace the tank's cap.

fill-bottle (B: class bottle)	Remove the bottle's cork, invoke fill-vessel to fill the bottle, then replace the bottle's cork.
fill-flask (F: class flask)	Sterilize the flask, then invoke fill-vessel to fill the flask.

Your code would need to know in advance which class of vessel is to be filled, and invoke a different procedure depending on the class, or else use a case statement that selects on class to choose the correct procedure dynamically. The former technique greatly constricts code flexibility. The latter is not too burdensome for three subclasses—but what if there were hundreds of them?

Filling Vessels Using Methods

To fill vessels by using methods, you could create a method for each of the four classes. Each of these methods would be similar to the analogous procedure, with the following differences:

- All four methods would be named fill.
- Each method would be bound to the class to which it applies.
- The methods for tank, bottle, and class would use call next method to invoke the method for vessel.

For example:

fill (V: class vessel)	Fill the vessel (tank, bottle, or flask).
fill (T: class tank)	Unscrew the tank's cap, execute call next method to fill the tank, then replace the tank's cap.
fill (B: class bottle)	Remove the bottle's cork, execute call next method to fill the bottle, then replace the bottle's cork.
fill (F: class flask)	Sterilize the flask, then execute call next method to fill the flask.

Your code would not need to know the class of a vessel to be filled, or use a case statement that selects on class. With the above methods defined, you can invoke fill on any tank, bottle, or flask. G2 then looks at the class of the vessel and invokes the fill method specific to that class. Thus fill means different things for different classes. This property of methods is called **polymorphism**.

When the method G2 selected executes call next method, G2 scans the class hierarchy path of the relevant class, looking for a superior class that also has a fill method. The class vessel is the direct superior of tank, bottle, and flask, and

defines a fill method. G2 invokes that method on the vessel. When the method returns, the lower-level method continues execution.

Encapsulation

Methods allow existing code to be extended more easily than procedures do. Suppose that you now define a fourth subclass of vessel, say tube, which must be washed before it can be filled, and labeled afterwards. You need only define a fill method bound to tube, and code that method to:

- Wash a tube.
- Execute call next method.
- Label the tube.

Existing code already used to fill tanks, bottles, and flasks, can now fill tubes also, yet the code itself has not changed at all. It did not need to change because the knowledge of how to fill an instance of each class resides in the class, in the form of its fill method; not in the code that calls the method, which needs to know only the operation's name. This property of methods is called **encapsulation**.

Duplicate Methods

On occasion, the nature of an operation requires it to do slightly different things under different circumstances to operands of the same class, and these differences require supplying the relevant method with different numbers of arguments. G2 does not provide optional arguments, but you can achieve the same effect by defining two or more methods that:

- Have the same name and perform the same operation.
- Apply to the same class.
- Take different numbers of arguments.

This capability allows you to customize the behavior of operations by giving different numbers of arguments when you invoke them.

Inheriting Methods

When a class defines no method for a particular operation, and a superior class does, the subclass inherits the method defined for the superior.

Suppose that vessel has another subclass, vial, that needs no preparation before filling and no cleanup afterwards. That is, filling a vial requires no customized behavior, but only the behavior characteristic of every vessel.

The vial class would need no fill method of its own. If you invoked fill on a vial, G2 would search vial's class inheritance path looking for a method named fill. The

class vessel is the direct superior of vial, and defines a fill method. G2 invokes that method on the vial.

G2 does the same thing every time you invoke a method, whether directly or with call next method: it scans the class inheritance path of the relevant class, and invokes the first method it encounters that has the correct name and the right number of arguments. Since every class is the first element of its own class inheritance path, this technique gives a locally defined method precedence over any inherited method.

Defining Methods

The steps for defining a set of methods to specify behavior are:

- Design and implement the class hierarchy.
- Create a method declaration for each operation to be implemented using methods.
- Create methods as needed to implement each operation.

You don't have to carry out these steps sequentially, though doing so is often convenient. The following sections give complete information on defining methods.

Designing a Class Hierarchy

Methods are closely linked with the class hierarchy on whose members the methods operate. If the hierarchy is correctly designed, methods can take advantage of its structure to provide very economical and modular behavioral specifications.

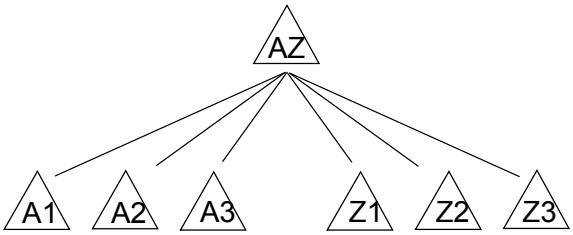
The general principle is to distribute the behavior of the various classes over the hierarchy in a way that takes maximum advantage of any inherent modularity in the behavior. For a given operation:

- The method for each class should do only things that are specific to it and all of its subclasses.
- Any behavior characteristic of more than one sibling class should be factored into a method defined for a parent class.
- Any behavior that differs for different child classes should be specified in a separate method defined for each class.

Class hierarchies designed for use with methods often contain levels of refinement that exist only to modularize behavior. For example, suppose that:

- The class AZ is the direct superior of six subclasses: A1, A2, A3, Z1, Z2.
- A start operation exists that is much the same for all six subclasses.
- The three A classes all customize start in one way, and the three Z classes customize it in another way.

The hierarchy looks like this:

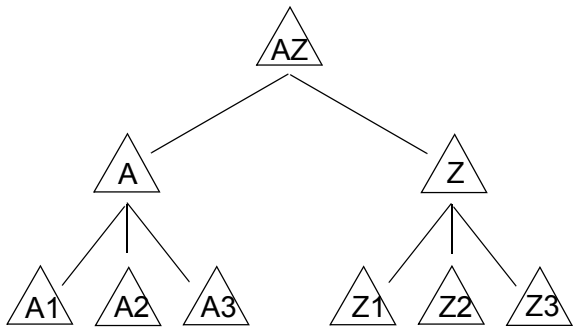


With this hierarchy, you have two possible ways to implement the start operation using methods. Both of them are unsatisfactory because:

1. No subclass defines a start method of its own. Instead, AZ defines a method and each subclass inherits it. This method does everything necessary to start an instance of any of the subclasses, and contains a case statement that selects on class. The case statement customizes start appropriately for the A classes and for the Z classes.
2. Every class has a start method. The method for AZ does everything necessary to start an instance of any of the subclasses. The method for each subclass uses call next method to invoke AZ's method, and provides class-specific customization. The methods for the three A classes are identical, as are the methods for the three Z classes.

The first technique uses a case statement to simulate the effect of customized methods. The second technique requires three copies of each class-specific method. In either case, the improvement over ordinary procedures is small.

The problem is that the structure of the AZ class hierarchy does not correctly reflect the structure of the constituent classes' behavior. The answer is to implement an intermediate class level:



With this hierarchy, you could define:

- A start method for AZ that does everything necessary to start an instance of any of the subclasses.
- A start method for A that uses call next method to invoke AZ's method, and provides the customization needed by an A class.
- A similar method for Z that provides the customization needed by a Z class.

With such methods defined, you can invoke start on any instance of the six lowest-level subclasses. Since none of the subclasses defines a start method of its own, G2 selects the method inherited from A or Z as appropriate. Thus the improved class structure allows you to define the needed behavior without case statements or duplicate code.

Implementing a Class Hierarchy

Techniques for implementing a class hierarchy appear in Classes and Class Hierarchy, and Definitions. This chapter does not repeat the information available there.

You do not have to define the classes for which you need methods before you define the methods themselves: you can implement classes, method definitions, and methods in any order. When you create methods before creating their class, all of the methods become usable as soon as you create the class.

However, you may find it convenient to define classes before defining the methods that use them, because a method defined for a nonexistent class contains a warning in its notes attribute that the class it applies to does not exist. Such warnings can become tiresome.

Class hierarchies rarely provide the ideal structure when first implemented, and methods rarely provide the ideal hierarchy of behavior from their inception. As

you develop your KB, you can iteratively develop and refine your class hierarchy and its associated methods, changing each as needed to reflect changes in the other. G2 immediately updates existing classes and methods as needed to reflect any such changes.

Creating Method Declarations

For every operation that you want to implement using methods, you must create a definition called a **method declaration**. This definition declares the name of the operation: every method that implements it for a particular class has this name.

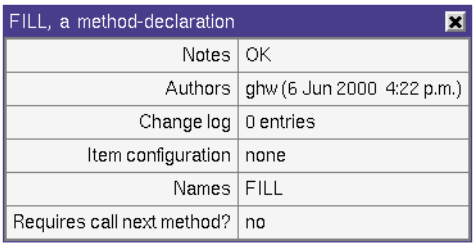
In the vessel example on , you would need one method declaration to define the fill operation. This declaration would suffice for all fill methods on all classes, no matter how numerous the methods, or how the class hierarchy might be structured or restructured. If you also needed to implement an empty operation, you would need a second method declaration to define it; and so on for any number of operations.

To create a method declaration:

1. Select KB Workspace > New Definition > procedure > method-declaration.
2. Open the method declaration's table.
3. Edit the names attribute to specify the name of the operation.

You can use any available name. If you specify more than one name, only the method with the first name is called.

For example, the following method declaration defines the fill operation used in the vessel example:



A method declaration can exist on any workspace. However, you can keep track of method declarations more easily if you store them systematically. One possibility is to keep them all on a workspace dedicated to that purpose. Another is to store each on the subworkspace of the highest-level class that uses the method.

Note If you give a method declaration and a procedure the same name, and use that name in a call statement or a start action, you cannot predict whether G2 will invoke a method or the procedure. Avoid using the same name for both a method declaration and a procedure.

Flagging Call Next Method Requirements

Method declarations have one class-specific attribute:

Attribute	Description
requires-call-next-method?	Flags developers that any method based on this method-declaration must include a call next method statement in order to function correctly.
Allowable values:	yes, no
Default value:	no

G2 does not enforce the restriction implied by requires-call-next-method?. The attribute is strictly informational, and has no effect on method compilation or KB execution.

Defining a Method

A method is syntactically similar to a procedure, and specifies what an operation does when the operand is of a particular class. You must create a separate method for each class on which you want to define a given operation.

To begin defining a method:

➔ Select KB Workspace > New Definition > procedure > method.

To specify the behavior of the method:

➔ Edit the text of the method to specify the desired behavior.

You can use any statement or action you could use in a procedure, as described in Procedures, plus the call next method statement, as described under Invoking a Superior Method.

When you are defining a method, the G2 text editor prompts for procedure and function signatures as described in Using the Procedure Signature Prompts in the Editor.

To bind a method to its method declaration:

➔ Give the method the same name as the method declaration for the operation that the method implements.

The method declaration need not exist when you reference it in the code of a method, but you must create it before you can actually invoke the method.

To bind a method to its class:

➔ Code the method so that its first argument is of the class to which the method applies.

Every method must have at least one argument, which must name the applicable class.

For example, the following code could be used as a fill method for the class flask in the vessel example:

```
fill(F: class flask) = (truth-value)
OK: truth-value;

begin
    call wash-flask(F);
    OK = call next method;
    if OK then call label-flask(F)
        else call empty(F);
    return OK
end
```

Note that, except for the call next method statement, this could be an ordinary procedure. Any method that does not use call next method could be a procedure, and any procedure whose first argument is an instance of a class could be a method defined for that class.

A method can exist on any workspace, but you can keep track of methods more easily if you store them systematically. One possibility is to keep them all on a workspace dedicated to that purpose. Another is to store each on the subworkspace of the highest-level class that uses it.

Method Attributes

A method has exactly the same attributes as an ordinary procedure, plus one more. The common attributes of procedures and methods are:

- tracing-and-breakpoints
- class-of-procedure-invocation
- default-procedure-priority
- uninterrupted-procedure-execution-limit

For information about how to use these attributes, see Defining a Procedure. These attributes serve the same purpose for methods and procedures.

Methods have two attributes that procedures do not:

Attribute	Description
qualified-name	Allows you to reference a specific method.
Allowable values:	*class-name* :: *method-name*
Default value:	none
synchronized	Whether a method should allow locking. For details, see Locking Mechanism for Objects.
Allowable values:	truth-value
Default value:	false

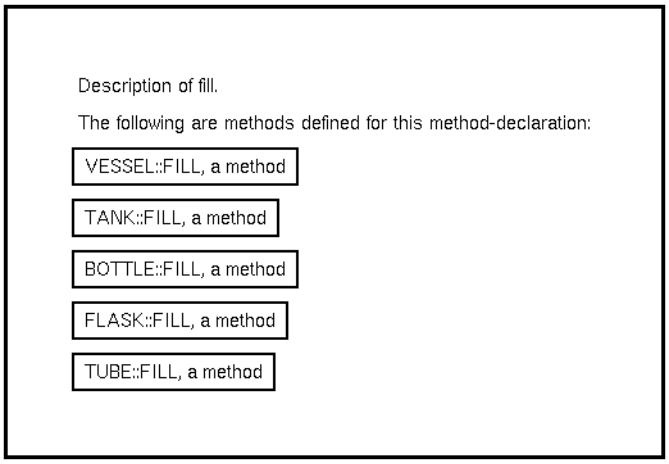
You cannot edit a method's qualified-name attribute. When you first create a method, its qualified-name is none. When you have correctly entered the text of the method, G2 sets its qualified name automatically based on the method's name and the class of its first argument.

When a method appears on a workspace, it has by default an attribute display that shows its qualified name. You can use this name to invoke the method directly, as described under Invoking a Method Directly.

Describing a Collection of Methods

You can use the Describe facility to see a list of all methods defined for that operation. For example, if you defined the classes and methods defined for the

vessel example on , and executed Describe on the method declaration for the fill method via the describe command in its menu, G2 would display:



Invoking a Method

Method invocation is syntactically similar to procedure invocation.

To invoke a method:

➔ Use call or start exactly as you would for a procedure.

For information about the call statement, see call. For information about the start action, see start. The differences between call and start are the same for methods and procedures.

Invoking a Method Generically

When you invoke a method generically, G2 selects the particular method to invoke.

To invoke a method generically:

➔ Specify an operation name in a call statement or start action.

To execute such an invocation, G2 does the following:

- Notes the class of the first argument in the call statement.
- Obtains the class inheritance path of that class.
- Scans the classes on the path looking for one that has a method that:
  - Has the name of the operation named in the invocation
  - Takes the number of arguments given in the invocation.

If G2 finds a class with such a method defined, G2 invokes the method, sending it the argument(s) specified in the invocation. If G2 reaches the end of the inheritance path without finding such a class, it signals an error.

For example, suppose that:

- flask-1 is an object of class flask.
- success is a truth-value.
- The fill method is as described in the vessel example on .

You could then invoke fill on a flask by executing:

```
success = call fill (flask-1)
```

The class flask and its parent vessel each has a fill method. Since every class appears first in its own inheritance path, G2 invokes the method for flask, sending it the arguments flask-1.

Matching Types in Generic Method Invocations

When you invoke a method generically, the class of the first argument and the number of arguments are significant, because they specify the class whose inheritance path G2 scans to search for a matching method, and the number of arguments that a matching method must have.

The types of any additional arguments, and the number and type(s) of any return values, are not significant for selecting which method to invoke. However, they must match whichever method G2 actually invokes, or G2 signals an error, as with a similar mismatch in an ordinary procedure invocation.

Invoking a Method Directly

In most cases, you invoke a method by specifying an operation, leaving G2 to select the correct method as described under Invoking a Method Generically.

Some situations require invoking a specific method and no other, bypassing G2's selection process. To allow such invocation, G2 provides a **qualified name** for every method. This name has the syntax:

class-name::*method-name*

For example, the qualified name of the fill method for flask is:

```
flask::fill
```

To invoke a method directly, you give its qualified name in a call statement or start action. G2 then invokes exactly the designated method. If the method does not exist, G2 signals an error: it does *not* search the inheritance path of the class specified in the generic name.

For example, suppose that:

- flask-1 is an object of class flask.
- The fill method is as described in the vessel example on .
- You want for some reason to bypass the normal sterilizing and labeling of a flask, which flask::fill performs, and fill the flask directly, via vessel::fill.

You could obtain the described effect by executing:

```
call vessel::fill (flask-1)
```

To execute this statement, G2 acts just as it would for an ordinary procedure call: it calls the fill method for vessel on the object flask-1.

The call next method statement has the same effect whether the method that contains it was invoked generically or directly, as described under Invoking a Superior Method.

Optional Direct Invocation

You can use direct invocation even where generic invocation would have the same effect. Such invocation, though initially unnecessary, ensures that your code will always call the particular method despite subsequent changes to the class hierarchy. Using direct invocation does not protect against changes to the effect of executing call next method.

For example, so long as the class hierarchy described for the vessel example remains unchanged:

```
success = call flask::fill (flask-1)
```

has the same effect as:

```
success = call fill (flask-1)
```

but the former will always invoke flask::fill no matter how the class hierarchy changes, while the latter might cease to do so. However, such a change in the hierarchy might in either case change the effect of call next method.

Matching Types in Direct Method Invocations

When you invoke a method directly, all arguments and any returned values must match the invocation, or G2 signals an error, as with any procedure call.

Direct invocation allows you to invoke a method on an item whose class differs from that of the first argument defined by the method. Such an invocation is correct if the item belongs to a subclass of the argument class, but not if it belongs to a superior class.

That is, you cannot directly invoke a method on an item whose class is superior to the class for which the method is defined. If you attempt to violate this restriction, G2 signals an error.

This restriction exists because an instance of a superior class may not have all of the attributes needed by a method or procedure designed for use with an inferior class: additional attributes may be added lower in the hierarchy.

Invoking a Superior Method

You can use the call next method statement to cause one method to call another that is defined for a superior class. The statement allows you to specify behavior hierarchically, as described under Designing a Class Hierarchy.

The syntax of the call next method statement is:

[*return-value* [, ...]] = call next method

Note that this syntax is the same as that of an ordinary call statement, except that no arguments appear. For example, the flask::fill method shown earlier included:

```
OK = call next method
```

G2 executes a call next method statement as follows:

- Scans the class inheritance path of the class to which the calling method applies.
- Checks each class in turn to see if it has an associated method that:
  - Has the same name as the calling method.
  - Takes the same number of arguments as the calling method.

If G2 finds such a method, G2 calls it, passing it the same arguments that the calling method received. If G2 does not find such a method, G2 does *not* signal an error; it proceeds to the next statement in the calling method.

A call next method statement is similar to an ordinary call statement. Specifically:

- The calling method suspends execution until the called method returns.
- The calling method need not be coded to receive any value(s) returned by the called method.
- If the calling method is coded to receive a return value, and the called method cannot be found or does not return a value, G2 signals an error.
- If the number and types of arguments passed to the called method do not match its declared number and types of arguments, G2 signals an error.

The call next method statement applies only to methods: it cannot appear in a procedure.

Duplicate Methods

Sometimes the nature of an operation requires it to do slightly different things under different circumstances to operands of the same class, and these differences require supplying the relevant method with different numbers of arguments. To provide for such cases, G2 allows you to define methods that:

- Have the same name.
- Apply to the same class.
- Take different numbers of arguments.

For brevity, G2 refers to such methods as **duplicate methods**, even though only the methods' qualified names are duplicates, and not the methods themselves.

For example, suppose that in the vessel example on , you sometimes want to supply the label to be pasted onto a flask, and other times want a default label. You could define a second fill method on flask.

fill (F: class flask, L: class label)	Sterilize the flask, execute call next method to fill the flask, than paste the label on the flask.

With both fill methods defined, G2 counts the arguments each time you invoke fill on a flask, and selects the fill method that has that number of arguments.

Duplicate and Superior Methods

When you use call next method in a duplicate method, you must make sure that the superior method is also duplicated, or G2 will not invoke it because it has the wrong number of arguments. For example, the two-argument flask::fill in the preceding example would not work correctly unless a two-argument vessel::fill existed also.

When duplicate superior methods exist, you *must* use call next method to invoke the correct superior method. Trying to call it directly may invoke the wrong method, as in the case of duplicate procedures, resulting in an error due to mismatched argument lists. For further information, see Duplicate Procedure Names.

Inlining a Method

A method declared as inlineable exists as a separate item, but is compiled as part of the calling code when called from another method or procedure. Inlining a method can improve performance by:

- Avoiding the overhead of a procedure invocation when the method is called. Procedure invocations consume runtime memory.
- Reducing the total number of instructions executed between the calling procedure or method and the inlined method.

However, inlined methods increase the size of the executable code, because the method code is copied redundantly to every point from which it is invoked. Inlining is best for small methods that are called frequently, typically from a loop that iterates many times.

Embedded code is incompatible with asynchronous and recursive invocation, so the code for an inlineable method is actually inlined only for a nonrecursive call. Inlineable methods exist as ordinary items; starts and recursive calls to them are handled just as if the method were not inlineable.

Inlining Restrictions

The following situations prevent G2 from compiling the code of an inlineable method into a calling method or procedure:

- The inlineable method and calling method or procedure are not defined in the same module.
- The invocation to the inlineable method is asynchronous.
- The invocation to the inlineable method is recursive.

These restrictions are the same as those applied to procedures. See Inlining Restrictions for the details of how G2 handles existing inlined code when procedures are transferred to different modules and when loading older KBs that violate the current inlining restrictions.

Declaring a Method as Inlineable

Declaring a method as inlineable requires the additional stable-hierarchy and the stable-for-dependent-compilations configuration.

Declaring a method as stable-hierarchy implies that a more specialized method will not be added below the current method in the method hierarchy. If the method includes any return values, the stable-hierarchy declaration additionally guarantees the return value types.

To declare that a method be inlined:

➔ Add these item configurations:

```
declare properties as follows : inlineable, stable-hierarchy,
    stable-for-dependent-compilations
```

Tip As with other configurations, the configuration statements stable-hierarchy and stable-for-dependent-compilations can be applied to a workspace. Whenever configurations are applied to a workspace, their effects propagate to every applicable item (methods in this case) upon the workspace, and all item subworkspaces.

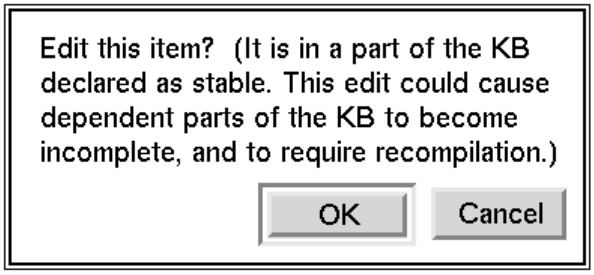
Recompiling an Inlineable Method

When you change the item-configuration of a method to include these properties, the notes attribute of the method changes to read:

```
OK, and note that this item needs to be recompiled to
    generate data needed for inlining
```

To recompile the inlineable method:

1 Choose edit from the table item menu of the method attribute table. G2 displays this dialog, because the method has been declared as stable:



2 Click OK to continue.

3 Save the edited method to recompile it. The method notes status will be OK.

Note After making a method inlineable and recompiling it, other methods and procedures that call the inlined method must be recompiled to incorporate the inlineable code. You can recompile a single procedure method by editing it in the Text Editor and saving any changes, or recompile your entire KB using Inspect.

Testing for an Inlined Method

You can test whether a method has been inlined through the G2 Tracing facility.

To test for an inlined method:

1. In the Debugging Parameters system table, change the tracing-and-breakpoints-enabled? attribute to yes.
2. Change the value of the tracing-message-level attribute to:

   1 (trace messages on entry and exit)
3. Run the procedure that calls the inlined method. If the inlined code has been incorporated into the calling procedure, the inlined method will not appear in the trace messages.

For a further discussion on using the inlineable and stable-hierarchy configurations, see Using Compilation Configurations.

Considerations for Multiple Inheritance

When you call a method for an instance of a multiple-inheritance class, G2 uses the inheritance path of the instance class to determine which methods should be invoked.

G2 uses the class inheritance path for two types of method calls:

- When you invoke a method generically by calling the method name without class qualification, G2 scans the class inheritance path of the item that is the first argument to the method call. The first class on the item inheritance path that has a method defined for it, with the method name and number of arguments in the call, is the method that G2 invokes.
- When you execute call next method, G2 similarly scans the inheritance path of the instance class, this time starting with the class following the class of the first-argument of the calling method.

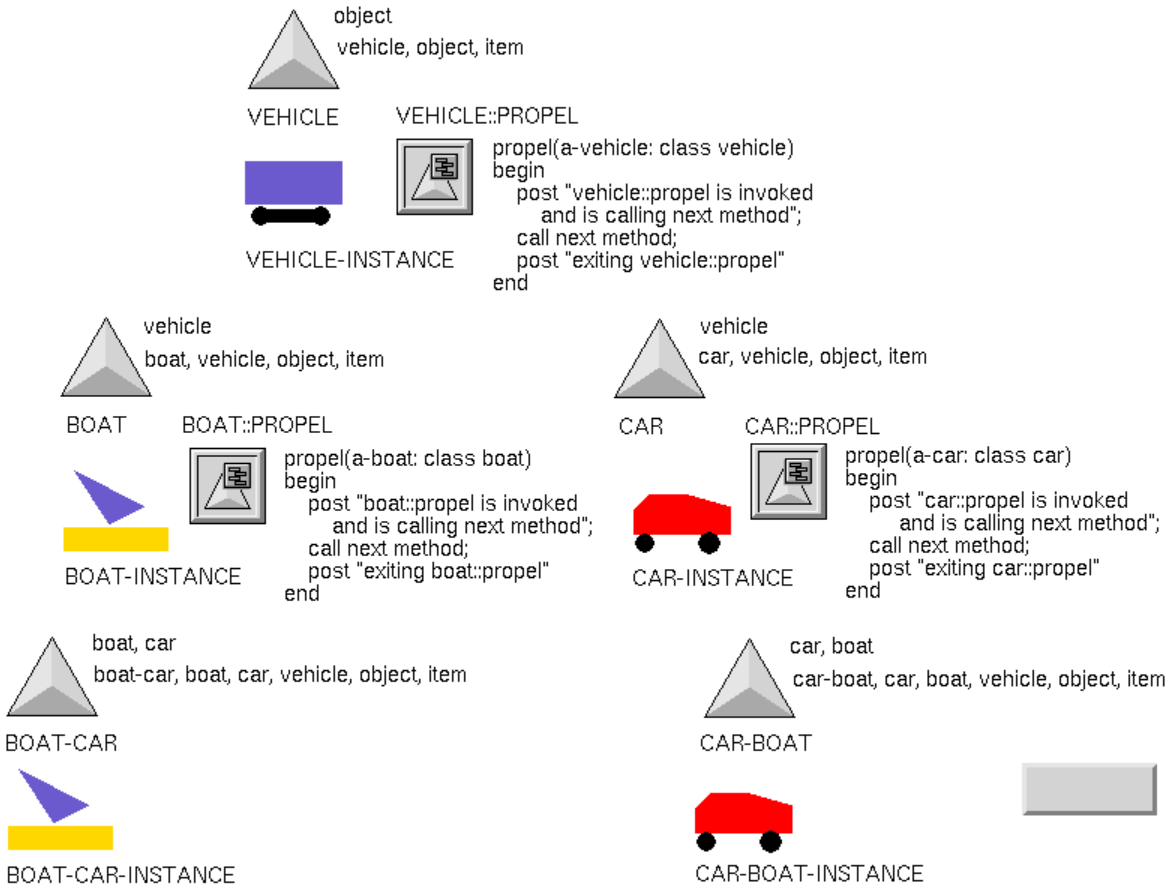
In either case, only the instance class inheritance path determines method invocation. The path might result from single inheritance or from linearizing multiple inheritance. G2 scans the path in exactly the same way, regardless of the underlying class hierarchy.

The effect of nested call next method invocations in multiple inheritance can be appear complex. This complexity does not result from any special action of G2, which invokes methods in exactly the same way irrespective of the particular class hierarchy. It results from the fact that in multiple inheritance, a class's hierarchy path is not a simple extension of a unique superior class's path, but a linearization of two or more different inheritance paths.

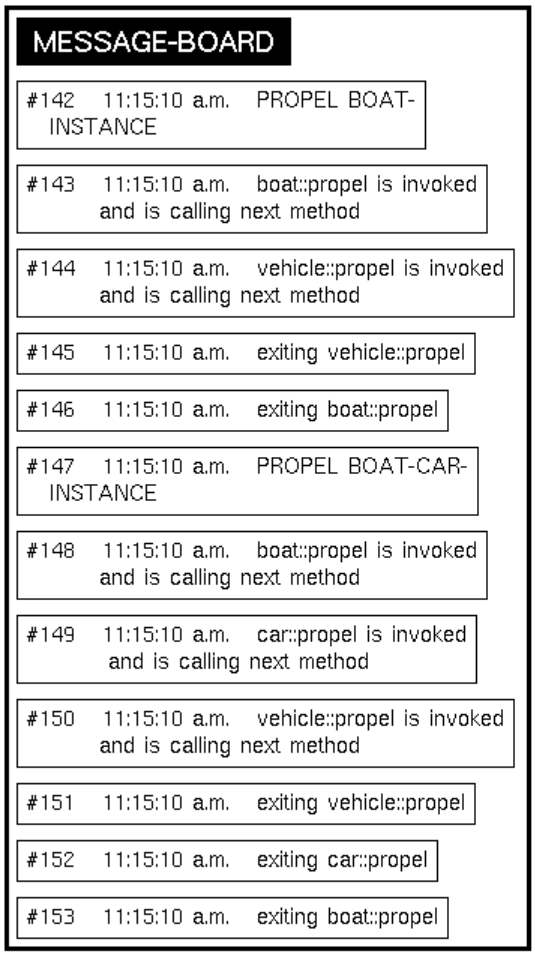
For example, assume that:

- Your KB contains five class definitions.
- Two of these class definitions, car-boat and boat-car, are defined with two direct-superior classes.
- You have defined three propel methods, vehicle::propel, car::propel, and boat::propel. Each of these methods contains a call next method statement placed between two post statements that indicate when the method is being invoked and when it is being exited.
- You have defined a procedure that calls the propel method for boat-instance and then for boat-car-instance.

The figure below shows the vehicle class hierarchy, and the propel methods defined for the hierarchy. To indicate inheritance, each class definition has attribute displays of its direct-superior classes and class inheritance path. There is a single instance of each class.



The next figure shows a procedure which calls the propel method for two instances. The messages posted to the Message Board indicate the order of method invocation:



SHOW-INVOCATION-ORDER

```
show-invocation-order()
begin
   post "PROPEL BOAT-INSTANCE";
   call propel(boat-instance);
   post "PROPEL BOAT-CAR-INSTANCE";
   call propel(boat-car-instance);
end
```

Notice that the methods are invoked strictly in order of the classes on the inheritance path of the instance class. This is due to the call next method statement at the beginning of each method which has the effect of invoking the methods from bottom up and then executing those methods top down, from the most general class method to the most specific class method.

Locking Mechanism for Objects

G2 provides two techniques for invoking a method:

- Using the call procedure statement, which executes the method synchronously. It tells G2 to invoke the method and wait until it returns before continuing to execute the calling procedure or method.
- Using the start action, which executes the method and runs it asynchronously. It tells G2 to schedule the method for execution, then continue processing the calling procedure or method. The method being started does not run until the entity that issued the start action either completes or enters a wait state.

To enter a wait state and allow other processing to occur, a procedure must execute one of the following statements: allow other processing, call ... across, collect data, for each ... do in parallel, or wait.

Historically, to avoid problems due to concurrency when a procedure or method enters a wait state, it has been up to the developer to ensure that only one procedure is accessing the same object at the same time.

G2 provides a locking mechanism, which allows other processing to occur when executing a method with a wait state, while ensuring that no more than one procedure that locks the same object can operate at the same time. This feature is similar to the use of the synchronized keyword in Java.

To support this feature, methods define the synchronized attribute, whose value is one of the following:

- no — When an unsynchronized method executes, the item that is the first argument to the method call is not locked and is, therefore, vulnerable to concurrency problems if the method contains a wait state. This is the default.
- yes — When a synchronized method executes, the item that is the first argument to the method call is locked, which prevents any other synchronized method from obtaining a lock on the item. When the synchronized method completes, either normally or due to an error, the lock is released.

When a synchronized method attempts to execute on a locked item, there are two possible outcomes, depending on how the method is executed, as follows:

When a synchronized method is executed in...	The synchronized method...
A different call chain	Waits to execute until the lock is released. If the synchronized method is invoked via a call statement, then the calling method enters a wait state until the lock is released, allowing other processing to occur while waiting for the lock to be released.
The same call chain	Executes normally.

A synchronized method is executed in a different call chain when:

- Using a call statement in a different procedure or method.
- Using the start action.
- Within a do in parallel, do in parallel until one completes, or for each...do in parallel statement.

A synchronized method is executed in the same call chain when it is executed via a call statement within the same method or within any method in the calling hierarchy of the synchronized method. Note that the call statement can be inside an on error block and is still considered within the same call chain. However, if the call statement is within a do in parallel, do in parallel until one completes, or for each ... do in parallel statement, this is considered a different call chain.

It is possible, due to careless use of synchronization, to have a program in which several methods are permanently in wait states, each waiting for the other to release a lock, in what is usually referred to as a *deadlock*. You can avoid this situation by ensuring that a call chain that obtains locks on multiple items always attempts to lock the items in the same order.

G2 provides a way of detecting and releasing deadlocks when they occur, interactively, programmatically, and automatically.

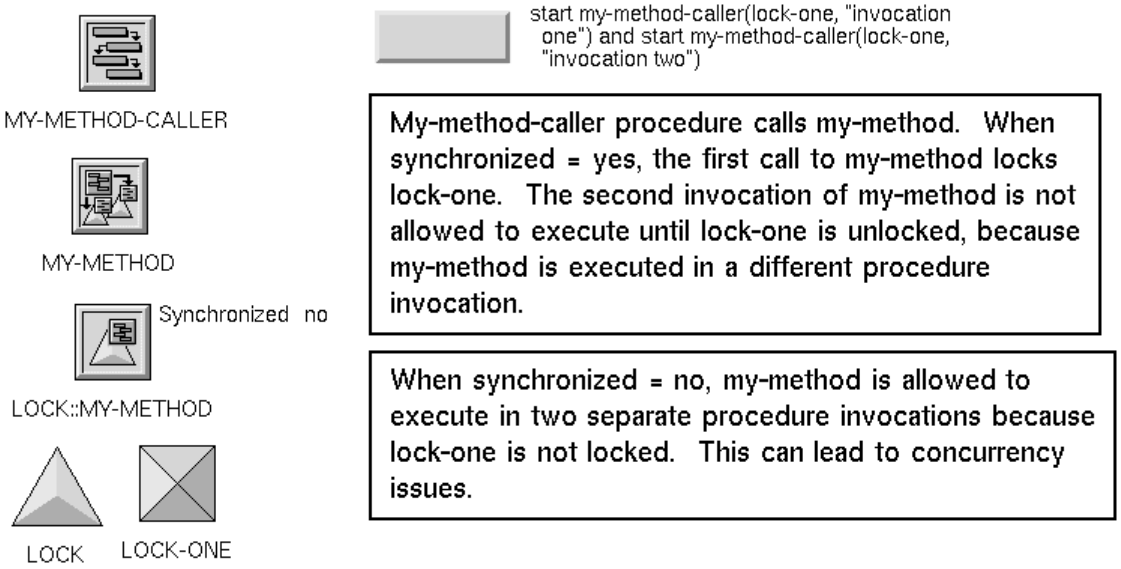
As a result of this change, the call action now causes procedures to enter a wait state when calling a synchronized method and when the first argument to the call is an item that is currently locked by another synchronized method.

The locking mechanism for G2 methods works in the same way as synchronized methods in Java. For more information, see `http://java.sun.com/docs/books/jls/third_edition/html/classes.html#8.4.3.6` and `http://java.sun.com/docs/books/jls/third_edition/html/memory`.

Example: Calling a Synchronized Method from a Procedure

This example demonstrates executing a synchronized method via a call statement in two separate procedure invocations when the method contains a wait state. When the method executes in the first invocation of the procedure, the item that is the first argument to the method is locked. Any other synchronized method call on the same item is not allowed to execute until the first method completes, even though the method contains a wait state, because the method is being executed in a different procedure invocation.

In this example, my-method-caller calls my-method on items of class lock. The action button starts my-method-caller for lock-one in two separate procedure invocations.



Here is the my-method synchronized method, which has synchronized set to yes. This method simply posts to the Message Board that my-method is starting, waits, then posts to the Message Board that my-method is completed. Because the method contains a wait state, it allows other processing to occur; however, because the method is synchronized, the item that is the first argument to the method is locked. Thus, any other synchronized method that attempts to execute on the same item must wait until the first method completes and releases the lock before it can execute.

```
my-method(thing-to-lock: class lock, message: text)
begin
    post "Starting my-method on [the name of thing-to-lock]: [message]";
    wait for 5 seconds;
    post "Completed my-method on [the name of thing-to-lock]: [message]";
end
```

Here is the procedure that executes my-method, using a call statement. It posts to the Message Board that it is about to call my-method, calls my-method, then posts to the Message Board that my-method is complete.

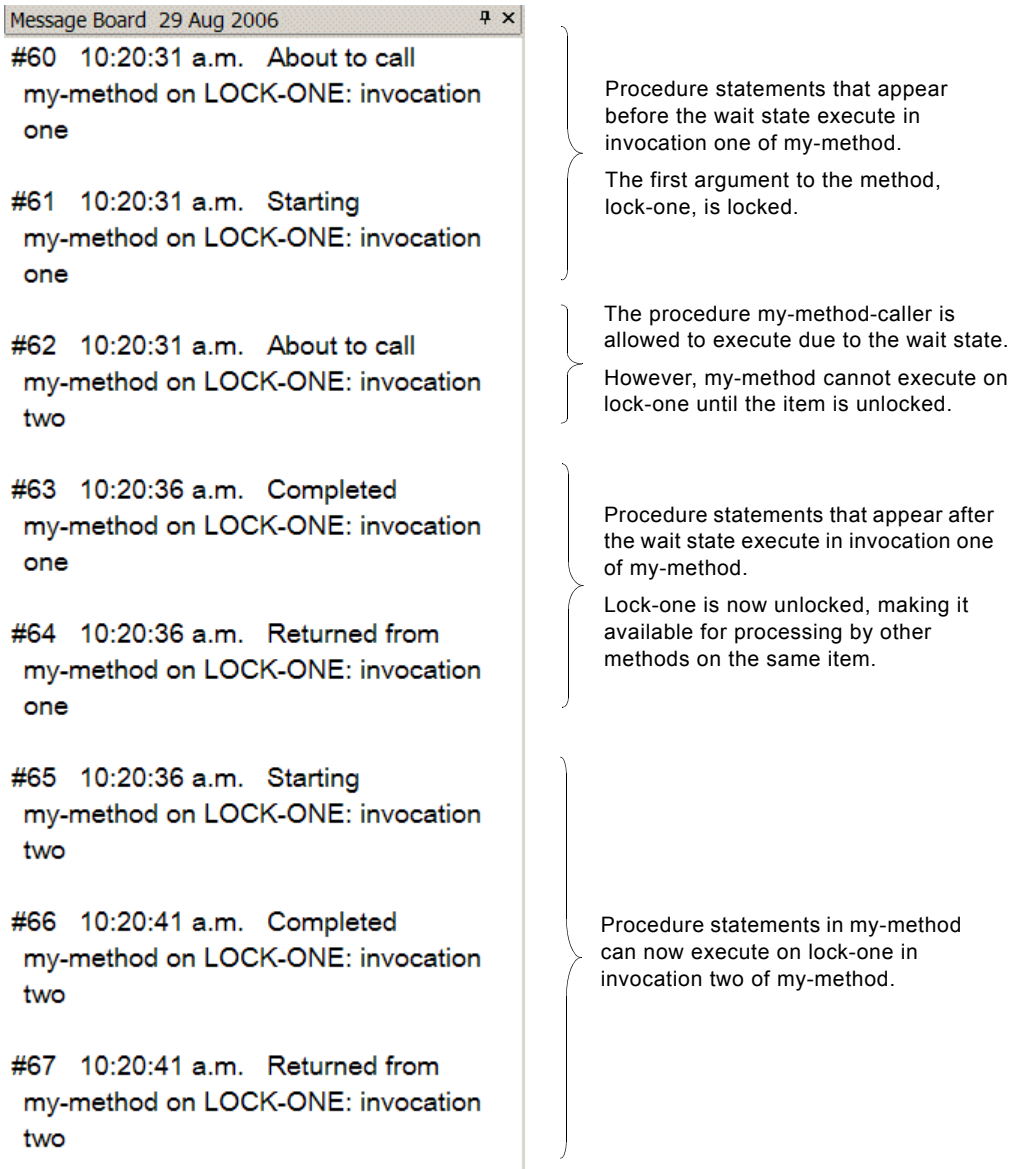
```
my-method-caller(thing-to-lock: class lock, message: text)
begin
    post "About to call my-method on [the name of thing-to-lock]: [message]";
    call my-method(thing-to-lock, message);
    post "Returned from my-method on [the name of thing-to-lock]: [message]";
end
```

The action button starts the my-method-caller procedure twice, which creates two separate procedure invocations, each using the same target object, lock-one:

```
start my-method-caller(lock-one, "invocation one") and
start my-method-caller(lock-one, "invocation two")
```

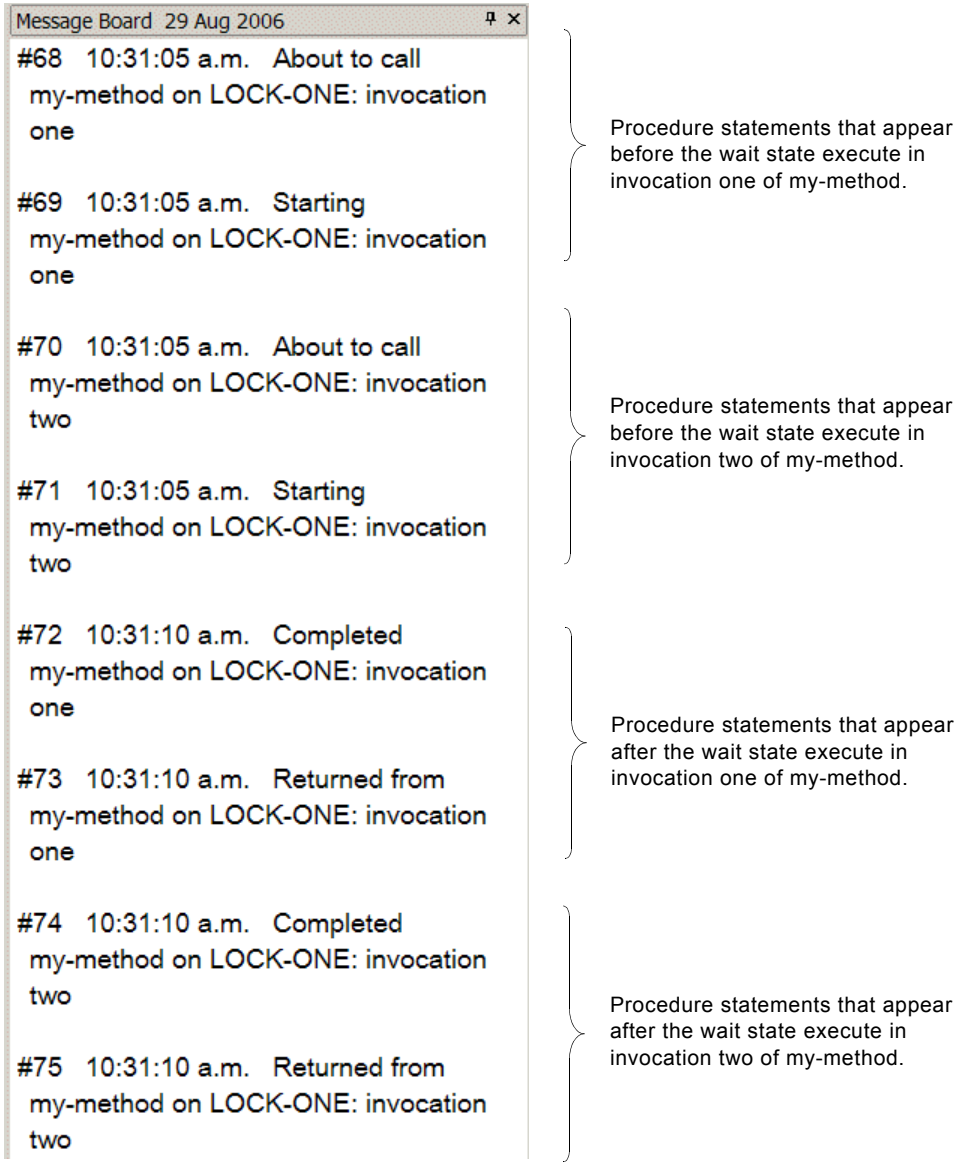
Using method synchronization (synchronized is yes) and a wait state, the synchronized method allows other processing to occur due to the wait state; however, it cannot execute the method on the same item because the item is locked. In this scenario, there are no concurrency issues.

Synchronized, wait state: No concurrency issues



With no method synchronization (synchronized is no) and a wait state, the synchronized method allows other processing to occur due to the wait state, including executing the method on the same item in a different invocation of the method. This is the scenario that can lead to concurrency issues because the item is not locked.

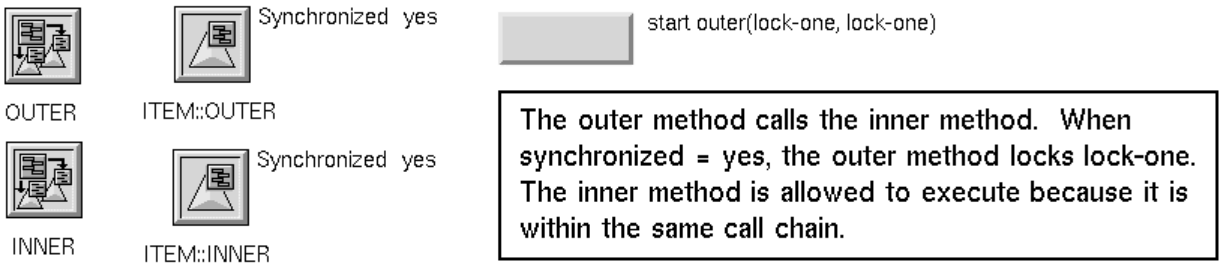
Unsynchronized, wait state: Concurrency issues



Example: Calling a Synchronized Method from the Same Method

This example demonstrates executing two synchronized methods on the same item when one method uses the call statement to execute the other method and both methods contain a wait state. When the outer method executes, the item that is the first argument to the method is locked. However, because the inner method is being called from the same call chain, it is allowed to execute, even though the item is locked.

In this example, outer calls inner on items of class lock. The action button starts outer for lock-one, which starts inner for lock-one:



Here is the outer synchronized method, which has synchronized set to yes. This method posts to the Message Board that outer is starting, waits, calls inner, waits, then posts to the Message Board that outer is completed. The method contains a wait state, which allows other processing to occur. Because the method is synchronized, the item that is the first argument to the method is locked when outer executes. However, because outer executes inner by using a call statement in the same synchronized method, inner is allowed to execute because it is within the same call chain.

```
outer(lock-for-outer: class item, lock-for-inner: class item)
begin
    post "Starting OUTER method on [the name of lock-for-outer]";
    wait for 5 seconds;
    call inner(lock-for-inner);
    wait for 5 seconds;
    post "Returning from OUTER method on [the name of lock-for-outer]";
end
```

Here is the inner synchronized method, which also has synchronized set to yes. This method simply posts to the Message Board that inner is starting, waits, then posts to the Message Board that inner is complete.

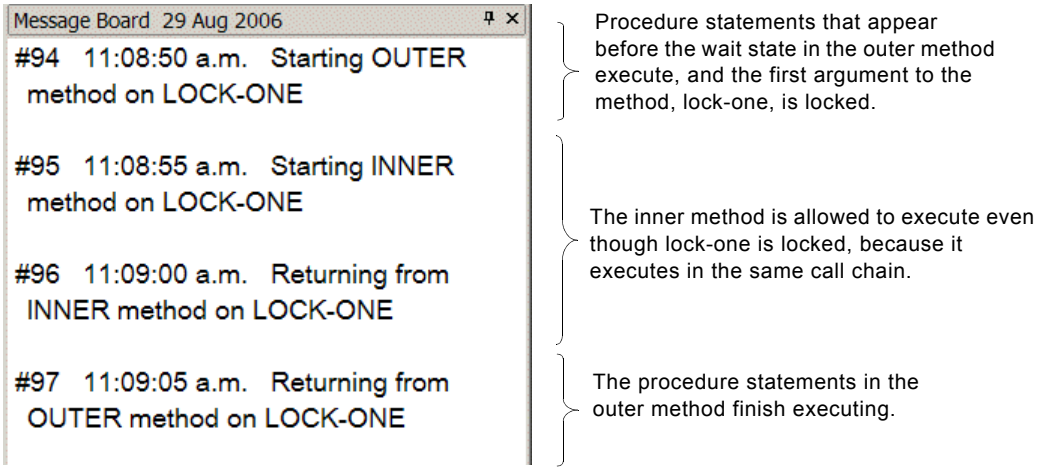
```
inner(lock: class item)
begin
    post "Starting INNER method on [the name of lock]";
    wait for 5 seconds;
    post "Returning from INNER method on [the name of lock]";
end
```

The action button starts outer using lock-one as the argument to both outer and inner:

```
start outer(lock-one, lock-one)
```

Using method synchronization (synchronized is yes) and a wait state, the outer synchronized method allows other processing to occur due to the wait state. Because the outer method calls the inner method, the inner method is allowed to execute, even though the item is locked, because it is in the same call chain. In this scenario, there are no concurrency issues because the methods execute within the same call chain.

Calling a synchronized method in the same call chain: No concurrency issues



Detecting and Releasing Deadlocks

You have two options for detecting deadlocks:

- Choose Miscellany > Detect Deadlocks.
- Use this system procedure:

  g2-detect-deadlocks
  ( )
  -> *return-value*: truth-value

These options simply indicate whether a deadlock exist.

You can also detect and break deadlocks by using one of these options:

- Choose Miscellany > Detect and Break Deadlocks.
- Use this system procedure:

  g2-detect-and-break-deadlocks
  ( )
  -> *return-value*: truth-value

  Returns false if no deadlock is detected; otherwise, returns true and breaks the deadlock.

- Setting the Automatic Deadlock Detection Frequency parameter in the Miscellaneous parameters system table to the frequency, in seconds, with which to check for deadlocks and break them when found.

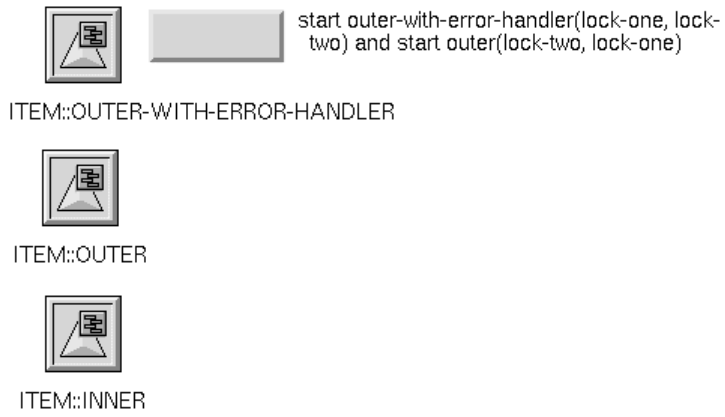
These options detect deadlocks and abort one of the involved methods by generating an instance of g2-deadlock-error, a subclass of g2-error. G2 chooses which method to abort, as follows:

- It chooses a method that contains an on error clause that catches errors of type g2-deadlock-error or one of its superior classes, like g2-error, or a method that was called by a procedure or method that contains such an on error clause, and so forth.
- Otherwise, it arbitrarily chooses one of the methods that is participating in a deadlock. Note that in the absence of an on error clause, all methods in the call chain will be aborted as a result when a deadlock exists.

Note that it always chooses a method that is waiting for a lock to be released, rather than a method or procedure that has called a method or procedure and is waiting for that method to return.

Example: Detecting and Releasing Deadlocks Using an Error Handler

This example shows a simple deadlock in which two synchronized methods are waiting for locks to be released on the same locked objects. The button below starts outer-with-error-handler locking lock-one and passing lock-two as the argument to inner, then it starts outer locking lock-two and passing lock-one as the argument to inner. The result is a deadlock, because each procedure is waiting for the other to complete before it can release the lock on the respective locked objects.



Here is the outer-with-error-handler method:

```
outer-with-error-handler (lock: class item, lock-for-inner: class item)
errobj: class error;
begin
    post "Starting OUTER-WITH-ERROR-HANDLER method on [the name of lock]";
    wait for 5 seconds;
    begin
        call inner(lock-for-inner);
    end
    on error (errobj)
        post "An error of class [the class of errobj] occurred: [the text of the
            error-description of errobj]";
        delete errobj;
    end;
    post "Returning from OUTER-WITH-ERROR-HANDLER method on [the name of
        lock]";
end
```

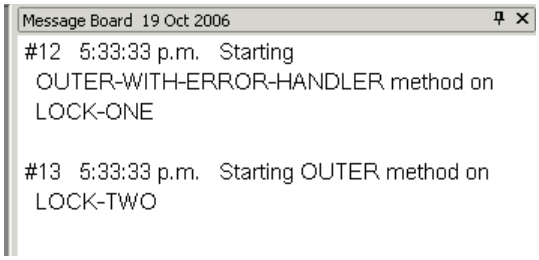
Here is the outer method:

```
outer (lock: class item, lock-for-inner: class item)
begin
    post "Starting OUTER method on [the name of lock]";
    wait for 5 seconds;
    call inner(lock-for-inner);
    post "Returning from OUTER method on [the name of lock]:";
end
```

Here is the inner method:

```
inner(lock: class item)
begin
    post "Starting INNER method on [the name of lock]";
    wait for 5 seconds;
    post "Returning from INNER method on [the name of lock]";
end
```

When you click the button that starts outer-with-error-handler, the outer-with-error-handler method starts on lock-one, then the outer method starts on lock-two. Neither method returns due to the deadlock whereby the first call to inner is waiting for lock-two, which is locked by outer, and the second call to inner is waiting for lock-one, which is locked by outer-with-error-handler and cannot complete.

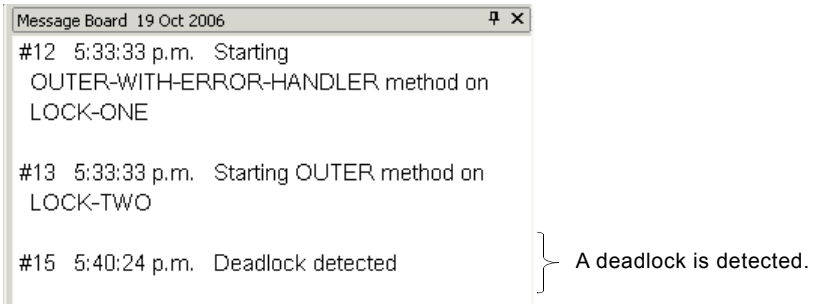


The following button and procedure detect deadlocks:

Here is the detect-deadlocks procedure:

```
detect-deadlocks()
result: truth-value;
begin
     result = call g2-detect-deadlocks();
     if (result) then
          post "Deadlock detected"
     else
          post "No Deadlock detected"
end
```

Clicking the button that detects deadlocks displays this new message in the Message Board:



Alternatively, choosing Miscellany > Detect Deadlocks displays the following message in the G2 Operator Logbook:

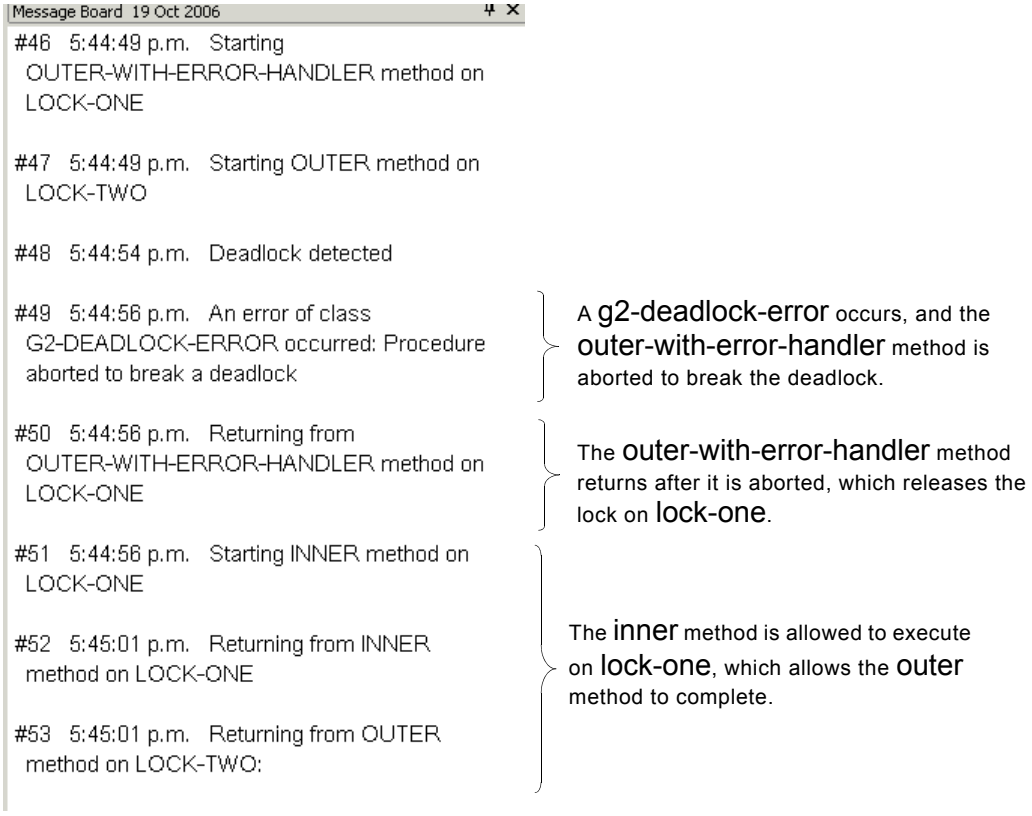
```
#14  5:38:23 p.m.  Deadlock detected: Use
 detect-and-break-deadlocks to abort deadlocked
 procedures
```

The following button and procedure detect and break deadlocks:

Here is the detect-and-break-deadlocks procedure:

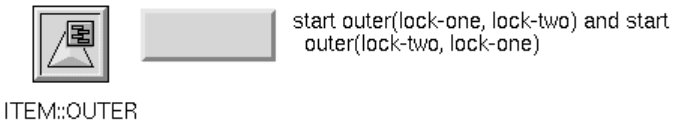
```
detect-and-break-deadlocks()
result: truth-value;
begin
     result = call g2-detect-and-break-deadlocks();
     if not (result) then
          post "No deadlock detected"
end
```

Clicking the button that detects and breaks deadlocks displays these new messages in the Message Board. First, the outer-with-error-handler method is aborted and a g2-deadlock-error occurs and is posted to the Message Board. Next, the outer-with-error-handler method returns, which releases the lock on lock-one. The call to outer is allowed to proceed by executing inner on lock-one and returning.

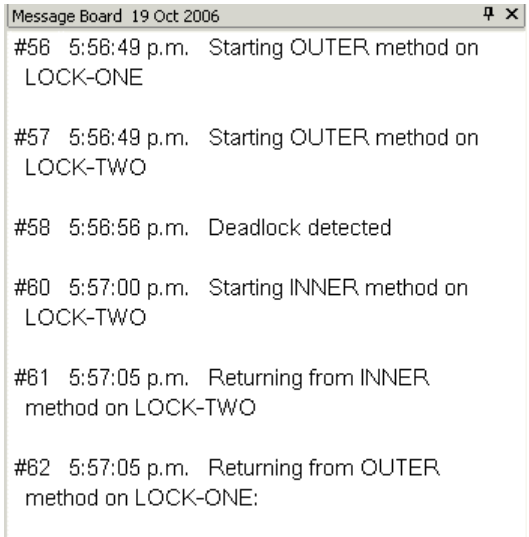


Example: Detecting and Releasing Deadlocks with No Error Handler

This example shows what happens when a deadlock occurs on a method that has no error handler. In this case, the button below starts outer locking lock-one and passing lock-two as the argument to inner, then it starts outer locking lock-two and passing lock-one as the argument to inner. Again, the result is a deadlock, because each procedure is waiting for the other to complete before it can release the lock on the respective locked objects.



In this case, clicking the button that starts outer, then clicking the button that detects deadlocks, then clicking the button that detects and breaks deadlocks results in these messages in the Message Board:



A deadlock is detected.

When the deadlock is broken, it aborts one of the methods, in this case, the second method call to outer, which releases the lock on lock-two, thereby allowing inner to execute on lock-two and the first method call to outer to return on lock-one.

Because neither method defined an error handler, the g2-deadlock-error appears in the G2 Operator Logbook, as follows:

```
#59  5:57:00 p.m.  Error:
 (G2-DEADLOCK-ERROR)

Procedure aborted to break a deadlock


Activity: system call statement
Within: ITEM::OUTER(LOCK-TWO, LOCK-ONE)
Local Names:
 LOCK: class item = LOCK-TWO;
 LOCK-FOR-INNER: class item = LOCK-ONE
Aborting procedure stack from
 ITEM::OUTER(LOCK-TWO, LOCK-ONE).
```

24

Rules, Inferencing, and Chaining

Describes how G2 invokes rules to perform actions.

Introduction

Rules establish how your KB responds to various conditions. Rules describe knowledge in a manner that allows your KB to draw conclusions from existing knowledge, to react to certain kinds of events, and to monitor the passage of time.

A **rule** expresses a programmatic response to a set of conditions. A rule contains a text and a set of attributes. When a rule fires, it executes one or more actions. For information about actions, see Actions.

When you create a new rule, you typically enter a two-part statement in its text. The first part, called the **antecedent**, tests for a condition. The second part, called the **consequent**, specifies the actions to take when the condition returns a value of true. This is an example of the text of a rule:

```
if the level of any tank = 0
    then conclude that the status of the tank is empty
```

G2 offers five kinds of rules:

- if rules are invocable in many ways.
- initially rules respond to the activation of their parent workspace.
- unconditionally rules execute their actions each time they are invoked.
- when rules are like if rules, but they cannot participate in chaining.
- whenever rules respond to events.

You can create:

- **Specific rules** that apply to specific items or values.
- **Generic rules** that apply to a set of items or values. Such rules use a for prefix in the text of the rule.

G2 invokes rules by using:

- Forward and backward chaining.
- Scanning.
- Event detection.
- Activating workspaces.
- Focusing on items.
- Focusing on rule categories.

You can specify whether a rule executes its actions sequentially or in parallel. This determines how G2 schedules the activities of an invoked rule and determines the transaction scope of those activities.

You can use G2's facilities for debugging and tracing to monitor the execution of rules. These facilities notify you when a particular rule has been invoked, suspended, and completed. You can also direct G2 to highlight the text box representation of each rule as it is invoked.

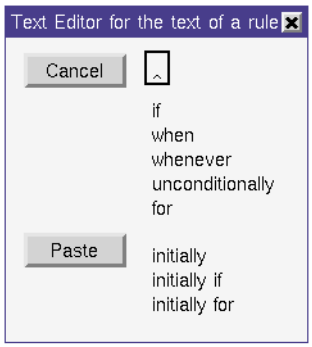
Creating a Rule

You create and edit rules interactively by using the Text Editor. A rule has a text box representation; it does not have an icon.

To create a rule:

1 Select KB Workspace > New Rule.

G2 displays the Text Editor, shown in the next figure, where you enter the text of the rule. The Text Editor displays prompts for entering one of the five kinds of rules, or the for prefix for entering a generic rule.



2 Enter the text of the rule and end the editing session by selecting End or pressing Return.

G2 displays the text box for the rule on the workspace. The text box is attached to the mouse pointer.

3 Move the mouse to position the rule on the workspace, and click to place the rule.

A completed rule has a text box representation, as shown in this figure:

```
if the level of any tank = 0 then
    conclude that the status of the tank is empty
```

To edit the text of an existing rule:

➔ Click anywhere in the text shown in the text box.

G2 displays the Text Editor for editing the text of the rule.

Displaying the Table for a Rule

You display the table for a rule by clicking on the border of its text box.

To display the table for a rule:

1 Click on the border of the text box of the rule to display its menu.

2 Select the table menu choice.

For information about specifying the attributes of a rule, see the sections that follow. For a summary of the attributes of a rule, see The Rule Class.

Cloning a Rule

One way of creating new rules is to clone an existing rule.

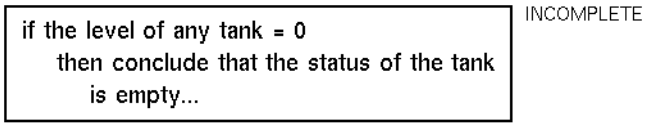
To create a rule by cloning:

1 Display the menu for the rule.

2 Select the clone menu choice.

3 Move the mouse pointer to position the new rule on the workspace, and click to place the rule.

After you clone a rule, G2 assumes that you will edit its text. G2 assumes you do not want more than one rule in your KB with the same text. Thus, after cloning a rule, G2 changes the status of the rule in its notes attribute to incomplete, and displays ... (ellipses) in the text of the cloned rule as shown below:



These two indicators tell you that you must edit or recompile the cloned rule before G2 can invoke it.

Changing the Font Size of a Rule

The default font size for a rule is large.

To change the font size of a rule:

➔ Mouse right on the rule and choose font, then choose extra large, large, or small.

Coding the Text of a Rule

The text of each rule has a two-part structure: an antecedent and a consequent. The antecedent contains a truth-value expression, and the consequent contains one or more actions. In the example below, the truth-value expression is the level of any tank = 0; and the action is conclude that the status of the tank is empty.

antecedent
consequent

```
if the level of any tank = 0
    then conclude that the status of the tank is empty
```

Coding the Antecedent

The **antecedent** of a rule includes one of the five reserved words, such as if, or an optional for prefix, which indicates that it is a generic rule. Following the reserved word in a rule is a truth-value expression, which indicates the conditions under which G2 executes the actions in the consequent of the rule.

Tip For a description of the truth-value type, see Using the Truth-Value Type. For a description of expressions that return truth-values, see Expressions.

Because a truth-value expression can produce a fuzzy truth value, G2 considers the expression in the antecedent of a rule to be true only if its fuzzy truth-value is greater than or equal to a fuzzy truth-value threshold. You specify this threshold for the entire KB in the truth-threshold attribute of the Inference Engine Parameters system table.

By default, truth-threshold contains the value .800 true. The value of the truth-threshold attribute applies to all rules that G2 invokes.

Coding the Consequent

The **consequent** of a rule specifies one or more actions that G2 executes if the antecedent evaluates to true. You can specify the following actions in the consequent of any rule:

abort	focus	move	show
activate	halt	pause	shut down g2
change	hide	print	start
conclude	inform	remove	transfer
create	insert	reset	update

deactivate	invoke	rotate
delete	make	set

For complete information about actions, see Actions. For information about sequential and parallel action execution, see Executing Actions. For a description of each G2 action, see the Dictionary of Actions.

Specifying More than One Action in the Consequent

You can specify more than one action in the consequent of a rule. To do so, include the reserved symbol and after each action in the consequent, except the last action, as follows:

```
if the level of any tank = 0
    then conclude that the status of the tank is empty and
      start flush ( the tank )
```

By default, the actions execute in parallel, and within the same transaction scope. For more information, see Executing Actions in the Consequent in Parallel.

Specifying Sequential Execution

If you specify more than one action in the consequent of a rule, you can specify that the actions execute sequentially, rather than in parallel, by including the phrase in order before the actions. For example:

```
if the level of any tank = 0
    then in order
        conclude that the status of the tank is empty and
        start flush (the tank)
```

When executing actions sequentially, each action executes within its own transaction scope. This is especially important when the expression within one action depends on the result of a previous action. For more information on sequential processing within rules, see Executing Actions in the Consequent Sequentially.

When you are defining a rule, the G2 text editor prompts for procedure and function signatures as described in Using the Procedure Signature Prompts in the Editor.

Kinds of Rules

G2 offers five kinds of rules, each of which serves a different purpose. You refer to a rule by its reserved word: if, initially, unconditionally, when, or whenever.

This table shows which methods of invocation G2 supports for each kind of rule:

Invocation Mechanism	if	initially	uncondi-tionally	when	whenever
Event detection					✓
Scanning	✓		✓	✓	
Workspace activation		✓			
Focusing on or invoking a rule category	✓		✓	✓	

For a detailed description of how G2 invokes each kind of rule, see Invoking Rules.

This table shows whether each kind of rule can participate in backward chaining and forward chaining:

Participation in Chaining	if	initially	uncondi-tionally	when	whenever
Forward chaining from	✓	✓	✓	✓	✓
Forward chaining to	✓		✓		
Backward chaining from	✓	✓		✓	✓
Backward chaining to	✓	✓*	✓		

* Only an initially rule can invoke another initially rule via backward chaining.

For information on how rules participate in backward and forward chaining, see Backward Chaining and Forward Chaining.

If Rules

An if rule specifies a condition, in the form of a truth-value expression, and one or more actions.

The syntax for an if rule is:

```
[for {any | the} generic-reference-expression] ...
    if truth-value-expression
        then [in order]
            action [and action] ...
```

For example:

```
if the level-in-gallons of tank-1 < empty-level-in-gallons of tank-1
    then conclude that the status of tank-1 is empty
```

An if rule is the most flexible kind of rule. An if rule can be invoked using forward and backward chaining, using scanning, and using the focus an invoke actions. An if rule can invoke other rules due to forward chaining and due to backward chaining.

To make an if rule behave like a when rule, edit its options attribute to include the phrases not invocable via forward chaining and not invocable via backward chaining.

Initially Rules

An initially rule optionally specifies a condition, in the form of a truth-value expression, and one or more actions.

G2 automatically invokes an initially rule each time its parent KB workspace is activated. After an initially rule executes, G2 automatically disables it.

A workspace is automatically activated when you start or restart the current KB. Also, the activate action explicitly activates a subworkspace configured as an activatable subworkspace. For information about how and when workspaces are activated, see Activating and Deactivating Workspaces.

Tip If you edit an initially rule after it has been invoked, G2 automatically enables the edited rule, if it otherwise valid to do so.

An initially rule can cause backward chaining and forward chaining to other rules. By default, an initially rule cannot be invoked using forward chaining or backward chaining. You can edit the options attribute of an initially rule so that it can be invoked using backward chaining, but only from other initially rules.

The syntax for an initially rule is:

```
initially
    [for {any | the} generic-reference-expression [unconditionally] ] ...
        [ { if truth-value-expression then} ]
            [in order]
                action [and action] ...
```

Forms of Initially Rules

G2 supports three forms of initially rules:

- initially if: In this form, the antecedent of the rule can optionally specify one or more for expressions to create a generic rule, and must include a truth-value expression that specifies the condition under which G2 executes its consequent. You form this initially rule like an if rule, for example:

```
initially
    for any demonstration-window DW
        if the subworkspace of DW exists
            then change the arrow-region icon-color of DW to yellow
```

- initially unconditionally: In this form, the antecedent of the rule can include one or more for expressions and an unconditionally clause. The antecedent for this initially rule always evaluates to 1.0 true. You form this initially rule like an unconditionally rule, for example:

```
initially
    for any valve V
        unconditionally
            change the name of this window to the symbol gbh-local
```

- initially with implied unconditionally: In this form, the antecedent of the rule cannot include a for expression. For example:

```
initially
    inform the operator that
      "Validation for [the name of the object O superior to this
      workspace] is underway." and
    focus on O
```

Effects on Rule Scanning

When the current KB starts, if any enabled workspace or subworkspace contains initially rules, G2 invokes those initially rules. G2 does not begin scanning other rules until the initially rules upon all newly activated workspaces complete or are suspended (due to backward chaining or data seeking). When the current KB starts, if no enabled workspace contains an initially rule, G2 begins scanning other types of rules immediately.

For an activatable subworkspace that contains initially rules, when G2 explicitly activates that subworkspace due to executing an activate action, G2 does not perform additional scanning of other rules until the initially rules on the newly activated subworkspace complete or are suspended (due to backward chaining or data seeking). If the newly activated workspace has no initially rules, G2 does not interrupt scanning other rules.

When any workspace is activated, G2 invokes each initially rule on the workspace at each rule's declared priority. When each initially rule has completed or is

suspended (due to backward chaining or data seeking), G2 schedules scanned rules for the next G2 clock tick.

Unconditionally Rules

An unconditionally rule specifies one or more actions to perform without a condition. G2 performs the consequent of an unconditionally rule each time the rule is invoked.

The syntax for an unconditionally rule is:

```
[for {any | the} generic-reference-expression] ...
    unconditionally
        [in order]
            action [and action] ...
```

Like an if rule, an unconditionally rule can be invoked using backward chaining, using scanning, and using the focus or invoke actions. Since an unconditionally rule has no antecedent, it cannot be invoked using forward chaining.

For example, this unconditionally rule:

```
unconditionally
    conclude that the status of tank-1 is empty
```

is equivalent to this if rule:

```
if true
    then conclude that the status of tank-1 is empty
```

When Rules

A when rule specifies a condition, in the form of a truth-value expression, and one or more actions.

A when rule is like an if rule that cannot be invoked using forward chaining or backward chaining. In this sense, a when rule is like an if rule that includes these phrases in its options attribute: not invocable via forward chaining, not invocable via backward chaining.

A when rule can be invoked using scanning and using the focus or invoke actions.

The syntax for a when rule is:

```
[for {any | the} generic-reference-expression] ...
    when truth-value-expression
        then [in order]
            action [and action] ...
```

This is an example of a when rule:

```
when the status of any pipe-valve PV1 is broken
    then inform the operator that
      "[ the name of PV1 ] is broken. Corrective action taken."
      and
    invoke repair rules for PV1
```

Whenever Rules

The antecedent of a whenever refers to an **event**. G2 invokes a whenever rule each time the specific event occurs. The syntax for a whenever rule is:

```
[for {any | the} generic-reference-expression] ...
    whenever event-expression [or event-expression] ...
      [and when truth-value-expression]
        then [in order]
            action [and action] ...
```

For example:

```
whenever any object O is moved by the user
    then start align-workspace-objects ( the superior item of O )
```

You can include more than one event expression in the antecedent of a whenever rule by using the reserved symbol or. This expression directs G2 to execute the consequent of the rule when any event specified in any of the event expressions in the antecedent occurs. For example:

```
whenever the level-status TS of any valve V receives a value
  or TS fails to receive a value
    then invoke heat-safety rules for V
```

Optionally, you can include the reserved symbol when and specify a truth-value expression, which represents an additional condition that must be satisfied for G2 to perform the consequent of the rule. For example:

```
whenever the temperature-status TS of any valve V receives a value
  and when TS is too-high
    then invoke heat-safety rules for V
```

Event Expressions

This section describes the various event expressions that a whenever rule can contain. The general principles described previously in this section apply to all event expressions. The detectable events are:

A Variable, Parameter, or Attribute Receives a Value

The syntax is:

whenever {*variable* | *parameter* | *attribute*} receives a value}

See Multiple Invocations Result in a Single Firing for a discussion of value updates and rule firing.

A Variable Fails to Receive a Value

The rule fires when a variable fails to receive a value for the first time after previously having a current value. For information on how variables fail to receive a value, see Handling a Variable Failure. Syntax:

whenever *variable* fails to receive a value

A Variable Loses Its Value

The syntax is:

whenever *variable* loses its value

For example:

```
whenever the miles-per-hour of any racing-car R loses its value
    then start check-auto-speed (R)
```

An Item Is Created

The syntax is:

whenever any instance of *class* is created

For example:

```
whenever any instance of tank T is created
    then conclude that total-number-of-tanks = total-number-of-tanks + 1
    and start initialize-tank (T)
```

G2 fires the rule after the create action is complete. To detect the creation of an item of any kind, specify item itself as the *item*.

An Item Is Moved on a Workspace

The rule can specify whether the item is moved by the user or by G2 executing a move action, or can fire in either case. The syntax is:

whenever *item* is moved [by {the user | G2}]

An Item Is Resized by user on a Workspace

The rule can specify whether the item is resized by the user. The syntax is:

whenever *item* is resized by the user

An Item Is Enabled or Disabled

The syntax is:

whenever *item* is enabled

whenever *item* is disabled

For example:

whenever any currency-exchange CE is disabled
then start exchange-tally (CE)

An Item Is Activated or Deactivated

The syntax is:

whenever {*item* | *subworkspace*} is activated

whenever {*item* | *subworkspace*} is deactivated

For example:

whenever the subworkspace of any shuttle S is activated
then conclude that the operational-status of S is active

The use of whenever rules for detecting item deactivation is an experimental feature. To make the rule fire, you must set the rule's may-refer-to-inactive-items evaluation attribute to true.

To make a whenever rule for detecting item deactivation operative:

➔ Create an action button to conclude the value of the rule's evaluation-attributes, using the following grammar:

conclude that the may-refer-to-inactive-items of the evaluation-attributes of *myrule* is true

---

Caution The evaluation-attributes of items are hidden attributes, which are not fully supported. Do not use them in any way that is not specifically described in the G2 documentation or recommended by Gensym Customer Support.

Two Items Become or Cease to Be Related by a Specific Relation

The syntax is:

whenever *item* {becomes | ceases to be} *relation-name item*

Two Items Become or Cease to Be Related by Any Relation

The syntax is:

whenever *item* becomes related to *item*

whenever *item* ceases to be related to *item*

For details, examples and further information, see Invoking Rules Using Relations.

Two Items Become or Cease to Be Connected

See Detecting Connection and Disconnection Events for information about detecting connection events with whenever rules.

Using Whenever Rules

Whenever rules have a number of special considerations, which this section discusses.

Event Expressions in Whenever Rules

You cannot use an event expression in a whenever rule to respond to events produced due to actions that use *indirect* referencing to affect an item or the value of an attribute. That is, if your KB's processing performs an action that affects an item or an attribute's value, and the action references that item or attribute in an indirect manner, then updating that item's knowledge or that attribute's value does not produce an event that is detectable using a whenever rule.

Referencing an item or attribute by means of the named by *symbolic-expression* expression, or referencing an item that is a member of a list or array, are examples of indirect referencing.

G2 cannot invoke a whenever rule by using scanning, forward chaining, or backward chaining. G2 ignores the focal-objects, focal-classes, and categories attributes of a whenever rule.

G2 does not execute a whenever rule that refers to a variable in an expression that uses the named by or nearest to phrases.

A whenever rule that tests for an attribute receiving a value is fired when an object is instantiated only if the referenced attribute is a variable or parameter with an initial value. The rule is *not* fired if the attribute of the object being instantiated is a simple data type with an initial value. Thus, the rule fires as a result of activating the object and its subobjects, not as a result of the object being instantiated.

Multiple Invocations Result in a Single Firing

Because the processing of rules must compete for priority with all of the other tasks G2 performs, G2 does not execute the actions in the whenever rule consequent every time the antecedent evaluates to true.

For example, a variable being monitored can change values very rapidly, and executing the consequent actions for each change would hinder G2 from performing other tasks in a timely manner. Instead, when a value is received, G2 determines whether there already is a task scheduled to execute the rule actions. If there is, G2 updates the task with the new value; otherwise it creates a new task

and adds it to the queue. This process ensures that when the rule finally executes, it has the latest value.

Reducing the Number of Invocations per Firing

The number of invocations that result in a single firing depend on factors such as what tasks are scheduled, their priority, and how rapidly the monitored events are occurring. Having a rule fire every time an event takes place can be a very difficult task. However, if you would like to reduce the number of rule invocations that result in a single firing, here are four suggestions:

- Add a wait statement after the conclude value statement.

  A wait statement suspends a procedure's execution, providing some time during which the rule has an opportunity to fire.

- Increase the priority of the rule.

  Give the rule more priority than the task that concludes a value. This will give the rule an opportunity to execute before a new value is concluded.

- Require an acknowledgement before a new value can be concluded.

  For example, add an action to the rule consequent that changes the state of some data that can be referenced before concluding another value.

- Use procedure statements instead of a whenever rule.

  Place the conclude statement and the statements that execute the response actions in the same procedure. Because multiple procedure invocations can exist, no values will be missed. This suggestion can also be applied to setting a value via a remote procedure call by G2 Gateway.

Coalescing Multiple Whenever Rule Invocations

This section describes:

- Problems caused by coalescence of multiple whenever rule invocations.
- Design requirements that whenever rules can be expected to satisfy.
- Possible event sequences in a G2 application employing whenever rules.
- Using of delays, priorities, and acknowledgments to resolve problems caused by whenever rule coalescence.
- Additional measures that may be required to ensure the reporting of every value that a variable, parameter or attribute receives.

A whenever rule set to monitor the values received by a variable, parameter, or object attribute exists as a single instance, rather than as multiple instances for each time the variable, parameter, or attribute is updated. Therefore, when such a

rule is scheduled to fire, multiple invocations of the rule are coalesced into the single instance, and the rule reports on the most recent value received.

Such coalescence can cause problems for developers whose applications expect to be informed of every value that the variable, parameter, or attribute takes. This chapter provides guidelines for solving such problems.

Whenever Rule Design Requirements

A whenever rule, when triggered by a change in a value, can be expected to report only the most recent value, not every value that has existed.

- Most recent value

  A whenever rule can trigger whenever a value changes and, via the rule consequent, report only the most recent value. This is the guaranteed behavior of a whenever rule. For rapidly changing information, reporting on the most recent value conserves computational resources.

  The most recent value is typically the desired information for a continuous variable, parameter or attribute that is being monitored. There are also situations where a finite state machine is being monitored. Although the finite state machine may undergo several transitions before a monitoring rule executes, the information desired can still be the most recent state information.

- Every value

  A whenever rule can also trigger whenever a value changes and report on every value. Depending on the conditions that prevail for the execution environment of the knowledge base, this can be difficult requirement to satisfy.

  This requirement applies to finite state machines for which the process of arriving at a final state is as important as the final state itself.

Possible Event Sequences

Two possible conditions can prevail in a knowledge base for the sequence of events relating the concluding of values to the execution of a rule monitoring those values:

- One-to-One

  Priorities or timing can be such that for every time a value is concluded, the whenever rule that monitors that value executes. This situation satisfies the every-value requirement.

- Many-to-One

Values can also be concluded many times before a monitoring rule that has been scheduled finally executes. This situation satisfies only the most-recent-value requirement.

Reporting Every Value

This section describes how a many-to-one situation can be modified to meet the every-value requirement. The techniques that can be used include:

- Delay

  Adding a delay after a value is concluded provides a time interval in which the monitoring rule has an opportunity to fire.

- Priority

  Increasing the priority of the monitoring rule with respect to the priority of the action that assigns a value gives the monitoring rule an opportunity to be executed before a new value is concluded.

- Acknowledgment

  Requiring an acknowledgment before a new value can be concluded ensures that each value is noted. This requires an architectural change to the knowledge base.

- Procedures

  A second architectural approach is to replace the conclude action and whenever rule with a single procedure that combines the setting of a value with the appropriate response action. Because multiple procedure invocations can exist, no values will be missed. This also applies to setting a value via an RPC called by G2 Gateway.

Specifying the Scope of the Rule

For each of the five kinds of rules, you can specify a different scope for the rule:

- A **specific rule** is a rule that applies to one item, as opposed to a set of items.
- A **generic rule** is a rule that applies to a set of items or values, as opposed to a single item or value.

Depending on the scope of the rule, G2 invokes and executes one or more rules for each item specified in the antecedent of the rule.

When creating both specific and generic rules, you typically use local names within the text of the rule.

Creating Specific Rules

You can use any kind of rule to form a specific rule. Specific rules typically refer to items by name. However, there are several expressions that indirectly or generically specify a particular item. For more information, see Item Expressions.

For example, the following rule applies only to the tank named tank-1:

```
initially if the level-in-gallons of tank-1 < the empty-level-in-gallons of tank-1
    then conclude that the status of tank-1 is empty and
      inform the operator that "Tank-1 is empty."
```

Indirect Specific Rules

Using a generic reference expression, a specific rule can refer generically to a particular item in the KB.

For example, the following rule applies only to the tank connected to the valve named valve-1. In this case, if more than one tank is, in fact, connected to valve-1, G2 applies this rule to *none* of them:

```
initially
    if the level-in-gallons of the tank connected to valve-1
      < the empty-level-in-gallons of the tank
        then conclude that the status of the tank is empty and
          inform the operator that "The tank [ the name of the tank ] is empty."
```

Local Names in Specific Rules

The following rule uses the local name T to refer to the tank connected to valve-1. Using local names can make specific rules easier to read. For general information on the use of local names, Using Local Names in Expressions.

```
initially
    if the level-in-gallons of the tank T connected to valve-1
      < the empty-level-in-gallons of T
        then conclude that the status of T is empty and
          inform the operator that "The tank [ the name of T ] is empty."
```

For clarity in rules, we recommend that local names consist of one to three uppercase letters.

Creating Generic Rules

The usefulness of a rule depends on the generic nature of the truth-value expression in its antecedent and of the actions in its consequent. You can use generic rules to capture knowledge about significant features of the items in your KB.

A generic rule includes at least one *generic-reference-expression* in its antecedent. A generic reference expression is preceded by the reserved word for and either

the or any, which identifies a set of items or values to which the rule can pertain. You can use any kind of rule to form a generic rule.

You can also use a generic reference expression in any action in the consequent of a rule. When creating generic actions in the consequent of a rule, you must precede the generic reference expression with the reserved word any.

For example, if your KB contains rules that refer to heat exchangers, you can write a generic rule that pertains to any heat exchanger, such as:

```
if the inlet-temperature-fahrenheit of any heat-exchanger HE
  < the max-inlet-temperature-fahrenheit of HE
        then focus on HE
```

Compare this to this rule that pertains to a particular heat exchanger, such as:

```
if the inlet-temperature-fahrenheit of heat-exchanger-4
  < the max-inlet-temperature-fahrenheit of heat-exchanger-4
    then focus on heat-exchanger-4
```

Two Forms for Generic Rules

You form a generic rule in one of two ways:

- *Before* the identifying reserved symbol (if, when, whenever, and so on), include a for expression that specifies a generic reference expression, as in the following example:

  ```
  for any valve
      if the status of the valve is broken
          then inform the operator that
            "The valve [ the name of the valve ] is broken."
  ```

  The expression any valve is a generic reference expression. You can nest for expressions in a rule, as in the following example:

  ```
  for any tank
      for any valve connected to the tank
          if the status of the valve is broken
              then inform the operator that
                "The valve [ the name of the valve ] is broken."
  ```

- *After* the identifying reserved symbol, include one or more generic reference expressions in the antecedent of the rule, as in the following example:

  ```
  if any valve is broken
      then inform the operator that "[ the name of the valve ] is broken."
  ```

  The expression any valve is a generic reference expression.

You can express a generic rule equivalently in either form. For example, G2 performs the following three generic rules equivalently:

```
for any valve connected to any tank
    if the status of the valve is broken
        then inform the operator that
          "The valve [ the name of the valve ] is broken."

for any tank
    if the status of any valve connected to the tank is broken
        then inform the operator that
          "The valve [ the name of the valve ] is broken."

if the status of any valve connected to any tank is broken
    then inform the operator that
      "The valve [ the name of the valve ] is broken."
```

For more information, see Using Generic Reference Expressions.

Using Local Names in Generic Rules

Local names can increase the expressive power of generic rules, as well as make the rules easier to read. In rules, we recommend that local names consist of one to three uppercase letters. For general information about local names, see Using Local Names in Expressions.

Use a local name to represent an item or value that is referenced more than once in the rule, as in the following two examples:

```
for any heat-exchanger HE
    if the inlet-temperature-fahrenheit of HE
      < the max-inlet-temperature-fahrenheit of HE
        then focus on HE

if the status of any valve V connected to any tank is broken
    then inform the operator that "The valve [ the name of V ] is broken."
```

You can use a local name to represent an item or value in the set, as follows:

```
for any tank T
    for any valve V connected to T
        if the status of V is broken
            then inform the operator that
        "The valve [ the name of V ] connected to [ the name of T ] is broken."
```

To refer generically to the same class in different parts of a generic rule, you must use different local names, as in the following example:

```
for any tank T
    if the sum over each valve VI
      connected at an input to T of ( the flow-rate of VI )
      > the sum over each valve VO connected at an output of T of
      ( the flow-rate of VO )
    then in order
        conclude that the expected-level-status of T is rising and
        inform the operator that
          "The level of the [ the class of T ] named [ the name
          of T ] should be [ the expected-status-level
          of T ], because the sum of rates-of-flow of all inputs to [ the name
          of T ] is greater than the sum of rates-of-flow of all outputs from [ the
          name of T ]."
```

In this case, the local name VI represents valves connected at an input of any tank, and the local name VO represents valves connected at an output of any tank.

Generic Rules and the Class Hierarchy

When a generic reference expression in the antecedent of a generic rule specifies any and the name of a class, G2 includes in the resulting set all items that are instances of the specified class that meet the condition. G2 also includes in the set all items that are instances of any subclass of that class that meet the condition.

Therefore, in a G2 application with a robust class hierarchy, you should develop generic rules carefully. You must consider the position in the class hierarchy of any class referenced in a generic rule. That is, for each generic rule, determine whether the action in the consequent is appropriately generic for the level of the class whose items the rule references.

Also, you should consider the pertinence of generic rules for each new class that you add to your class hierarchy. That is, the consequent of each rule that refers to items generically by class must be meaningful and appropriate for instances of each new subclass added to your KB.

Determining the Number of Generic Rules That Are Invoked

By default, when a generic rule is invoked, G2 actually executes one copy of the rule for each item or value in the set identified by the generic reference expressions in the antecedent of the rule.

For example, when this rule is invoked, G2 invokes a separate copy of the rule for each item upon its parent workspace:

```
if any item upon this workspace is moved
    then start update-leftmost-item-position ( this workspace )
```

As another example, when this rule is invoked, G2 determines the number of connected pairs of objects and water pipes. For each such pair, G2 determines

whether the notes attribute of the object in that pair contains the text "OK", then invokes a separate copy of the rule for each such object.

```
for any object O upon this workspace
    for any water-pipe connected to O
        if the text of the notes of O /= "OK"
            then start validate-objects ( this workspace )
```

Scanning Generic Rules

When scanning generic rules, G2 invokes a separate copy of that rule at the beginning of each scan interval, *for each item or value in the set* specified by the generic reference expressions in the antecedent of the rule.

For example, if you set a scan interval for a rule that applies to any terminal connected to any modem, G2 invokes that rule once every scan interval for each connected terminal/modem pair.

Tip Because scanned generic rules can cause the invocation of very many rules per scan interval, employ them carefully.

For more information about invoking rules by scanning, see Scanning Rules.

Using Generic Rules with Focal Objects

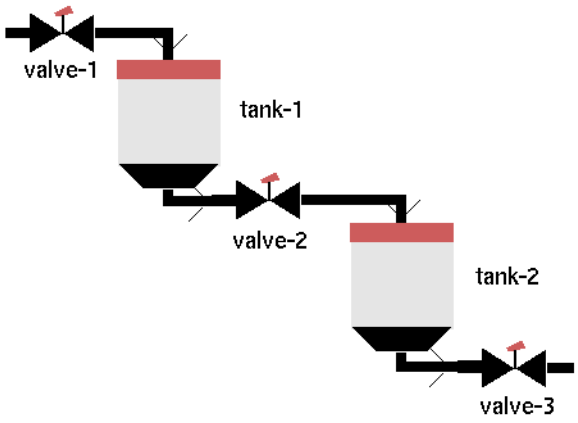
You can invoke rules with a focal object, as described in Focusing on Items.

For example, when G2 invokes the following generic rule with a focal object, by executing the action focus on valve-1 anywhere in the KB, the rule applies to only *one valve*, namely, the one that the focus action specifies. An example is:

```
for any valve V1
    if V1 is broken
        then inform the operator that "[V1] is broken"
          and conclude that the is-maintenance-required of V1 is true
```

In contrast, here is a more complicated rule that refers to the valves and tanks upon the workspace shown in the next figure:

```
for any tank T
    for any valve V1 connected to T
        if V1 is broken ...
```



If this rule is invoked with tank-2 as a focal object, G2 applies the rule to only the connected pairs of tanks and valves of which tank-2 is a member. Given the items on the workspace shown above, G2 applies this rule to the connected pair tank-2 and valve-2, and to the connected pair tank-2 and valve-3. For these two cases, G2 determines whether the valve in the connected pair is broken, and, if so, executes the consequent of the rule.

Getting Focal Classes for Rules

G2 provides system procedures for getting the focal classes to which a generic rule applies and for getting the rules to which a variable would backward chain.

For details, see Rule Operations in the *G2 System Procedures Reference Manual*.

Invoking Rules

By default, each rule in your KB is idle. To *invoke* a rule means to begin evaluating the antecedent of the rule. G2 can invoke a rule when:

- Data referenced in the antecedent of an if rule changes via forward chaining.
- The KB requires a value for a variable that is concluded in the consequent of any rule via backward chaining.
- The parent KB workspace of an initially rule is activated.
- G2 detects an event that is tested in the antecedent of a whenever rule.
- A time interval passes for a rule that specifies scanning.
- The KB executes a focus or invoke action that names an object, object class, or rule category associated with the rule.

If G2 is paused, and you interactively make a change that would invoke a rule if G2 were running, the rule is invoked when G2 resumes execution.

Based on the kinds of reasoning that you desire, you should develop rules that use the appropriate mechanisms for rule invocation.

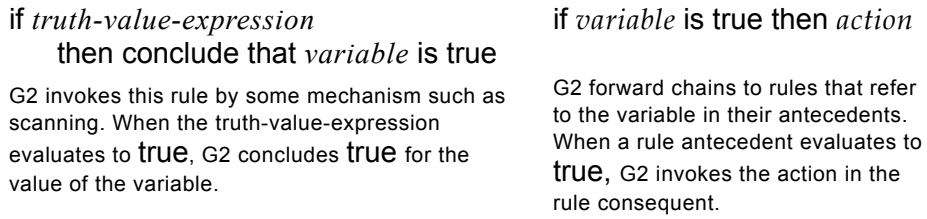
Forward Chaining

G2 uses **forward chaining** to invoke if rules when the value referenced in the antecedent of the rule changes.

Forward chaining is a form of deductive reasoning. Your KB can use rules invoked using forward chaining to draw conclusions from other rules. Similarly, your KB can use forward chaining to initiate actions from conclusions drawn in other rules.

When forward chaining takes place, G2 identifies all rules whose antecedents refer to the changed value. If a rule concludes the new value, the rule must be able to cause forward chaining. The rules that refer to the value in their antecedents must be invocable via forward chaining. G2 invokes these rules in parallel, and schedules each for execution at its declared priority.

The following figure illustrates forward chaining by using an abstract example:



G2 invokes rules by using forward chaining when the value referenced in the antecedent changes for any reason, not just by concluding a value in another rule. Also, the value referenced in the antecedent can be an attribute, a variable, or a parameter.

By default, G2 only forward chains to rules when a variable or parameter referenced in the rule receives a new value that is *different* from its previous value; whereas, G2 forward chains to rules whenever an attribute referenced in the rule receives a new value, regardless of whether the value has changed. You can also configure variables and parameters to cause forward chaining even when the value does not change. For details, see Forward Chaining on Unchanged Variables and Parameters.

To illustrate, suppose the following rule is invoked by scanning:

```
if valve-is-broken of valve-1
    then conclude that the temperature-is-too-hot of tank-1 is true
```

If this rule is declared to participate in forward chaining, when G2 concludes a value for the temperature-is-too-hot of tank-1, G2 forward chains to any rule that

1) refers to the temperature-is-too-hot of tank-1 in its antecedent, and 2) can be invoked using forward chaining. For example, G2 could forward chain to this rule:

```
if temperature-is-too-hot of tank-1
    then invoke safety rules
```

The antecedent of this rule refers to the temperature-is-too-hot of tank-1, which is given by a logical variable that is declared to use forward chaining.

Note To create a rule that is invoked when the value of a history expression changes, use a whenever rule. For efficiency reasons, G2 does not invoke an if rule that is configured to forward chain when the rule antecedent refers to a history expression.

To use forward chaining in rules:

1. In the rule that concludes a value in its consequent, specify the following phrase in the options attribute:

   may cause forward chaining

2. In the rule that refers to the value in its antecedent, specify the following phrase in the options attribute:

   invocable via forward chaining

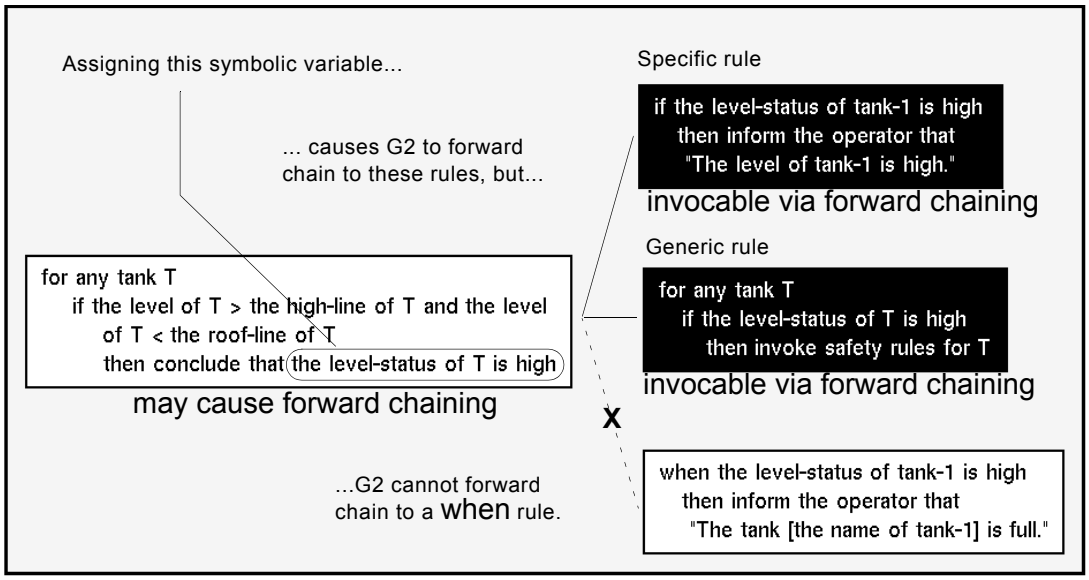
3. If the value that is concluded is a variable or parameter, include this phrase in the options attribute for the variable or parameter, whose value is obtained via forward chaining:

   do forward chain

By default, symbolic and logical variables and parameters have the setting do forward chain in their options attributes. Text and quantitative variables and parameters have the default setting do not forward chain in their options attributes. You can edit these settings so that any variable or parameter in your KB can support or not support forward chaining.

This figure illustrates how concluding a new value for an item in one rule causes forward chaining to other rules whose antecedents refer to that item. In this example, level-status is an attribute whose value is given by a symbolic variable.

Forward Chaining



In general, forward chaining only works with if rules. Although G2 never allows forward chaining *to* a whenever rule, you can potentially forward chain *from* a whenever rule. In this case, forward chaining can occur when a whenever rule concludes a value for a variable that is declared to participate in forward chaining. All other rules can be set to be invocable or not invocable by forward chaining.

Ordering of Rules Invoked by Forward Chaining

When two or more rules are invocable due to forward chaining based on a change to the same attribute or current value of the same item, G2 does not specify the order in which those rules are invoked.

To specify such ordering, you can set the rule-priority attributes of the rules differently. When this has been done, G2 schedules the rules for invocation according to their priority.

Implementing Loops Using Forward Chaining

A rule can potentially forward chain to itself. For example, given that the value of X is greater than zero, the first time the following rule is invoked it forward chains to itself many times within a second, until X equals 20:

```
if (x > 0 and x <= 19)
    then conclude that x = x + 1
```

Such rules can implement a processing loop, but such loops must have some way of ending. For example, the rule above forward chains to itself only until X equals 20.

In contrast, the next rule can forward chain to itself repeatedly without stopping, thereby heavily loading G2's ability to run all the current activities:

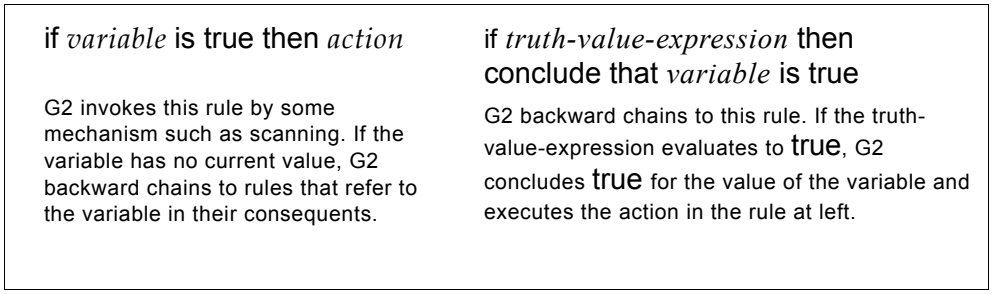
```
if x > 0
    then conclude that x = x + 1
```

Notice that this behavior cannot occur when this rule is no longer declared to allow forward chaining. Take care not to construct rules that forward chain to themselves without an appropriate termination condition.

Backward Chaining

Backward chaining takes place when an item in the KB references a variable that does not have a current value. G2 can invoke via backward chaining one or more rules whose consequent provides a value for the variable.

The following figure illustrates backward chaining by using an abstract example:



To illustrate with a simple example, suppose the following rule is invoked by scanning:

```
if valve-is-broken of valve-1
    then focus on repair rules for valve-1
```

If this rule is declared to participate in data seeking, when G2 evaluates the antecedent of the rule and valve-is-broken has no current value, then G2 backward chains to any rule that 1) concludes a value for the valve-is-broken attribute of valve-1, and 2) can be invoked using backward chaining. For example, G2 could backward chain to this rule:

```
for any valve V
    if valve-is-closed of V and tank-is-overflowing of the tank connected to V
        then conclude that valve-is-broken of V
```

The consequent of this rule refers to the valve-is-broken of valve-1, which is given by a logical variable that is declared to use backward chaining.

To evaluate the antecedent of this rule, G2 might be required to backward chain to other rules that provide values for valve-is-closed and tank-is-overflowing, which are also variables. G2 continues to backward chain to other rules until it can fully evaluate the antecedents of each invoked rule.

To use backward chaining in rules:

1 In the rule that refers to the variable in its antecedent, specify the following phrase in the options attribute:

    may cause data seeking

    Backward chaining is one form of data seeking using variables. For information on other forms of data seeking, see Obtaining Values for Variables.

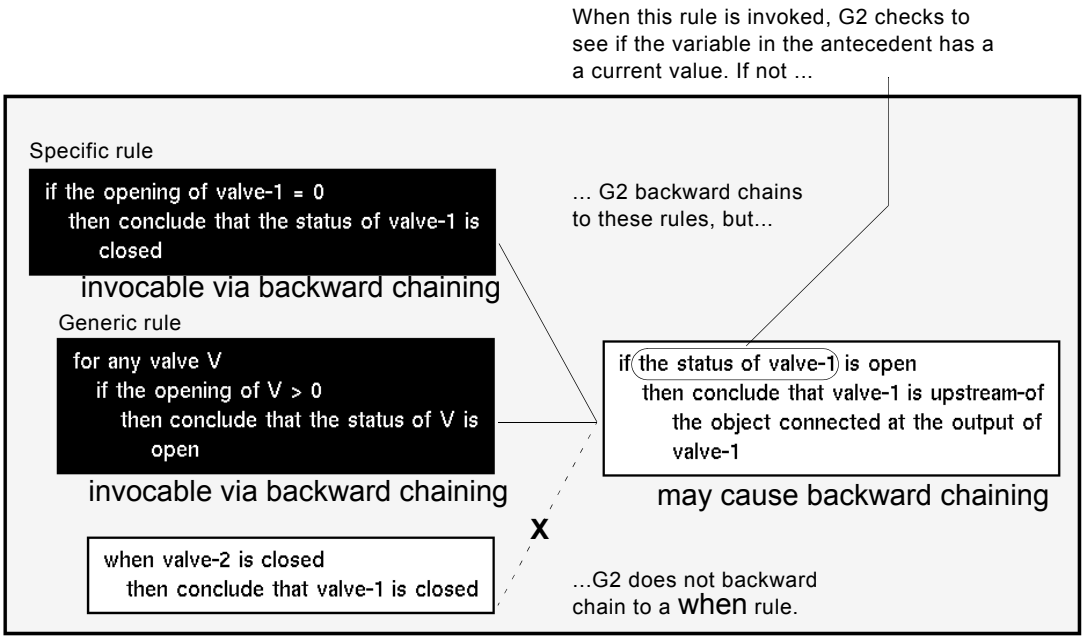
2 In the rule that supplies a value for the variable in its consequent, specify the following phrase in the options attribute:

    invocable via backward chaining

You can define subclasses of variables whose value is computed by a formula specified in the formula attribute of the variable. Such a variable cannot be used in the consequent of a rule, but can be used in the antecedent. For further information, see Variables, Parameters, and Rules and Obtaining Values for Variables.

The following figure shows another example of backward chaining. Note that a rule cannot backward chain to a when rule, because when rules do not support backward chaining.

Backward Chaining



Using Breadth-First Backward Chaining

If you have a set of rules that provide a new current value for a variable, you can organize the set of rules to be invocable by using **breadth-first backward chaining**.

When G2 invokes a rule whose antecedent requires a value for a variable that specifies breadth-first backward chaining, G2 invokes in parallel every rule that provides a value for that variable in its consequent. G2 schedules each rule for execution at its declared priority. As soon as any rule in that set provides a value for the variable, G2 cancels execution of the other rules.

To use breadth-first backward chaining:

1 Include this phrase in the options attribute for the variable whose value is obtained via backward chaining:

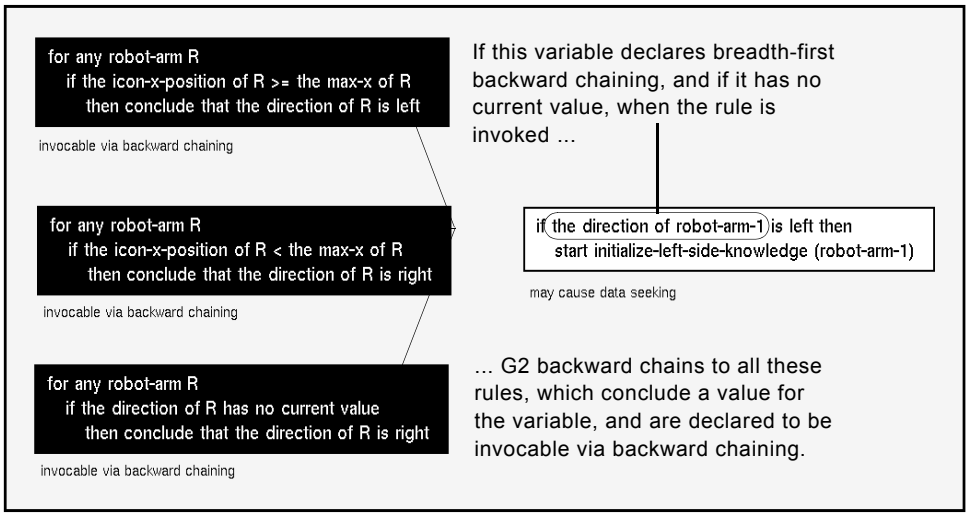
```
breadth first backward chain
```

2 Specify backward chaining in the rules by following the steps under Backward Chaining.

This is the default setting for all subclasses of the system-defined variable class.

In the following figure, in the rule on the right, suppose the direction of robot-arm-1 is given by a variable. If the variable declares breadth-first backward chaining, G2 invokes each rule that can provide a value for that variable.

Breadth-First Backward Chaining



Note When G2 invokes rules that can conclude a value for a variable that declares breadth-first backward chaining, G2 does *not* cancel the invocation of the same rules invoked due to some other rule, procedure, or function.

Using Depth-First Backward Chaining

Often, rules provide different techniques for determining the value of a single variable. These techniques are called **heuristics**. When more than one rule provides a value for a variable via backward chaining, you might want to specify the order in which G2 executes each rule by specifying a **precedence**. This is called **depth-first backward chaining**.

When G2 concludes a value for a variable that declares depth-first backward chaining, G2 identifies the rules that can conclude a value for the variable, then invokes those rules in sequence until one concludes the required value.
G2 invokes these rules in an order based on the precedence set in each rule.

To use depth-first backward chaining:

1 Include this phrase in the options attribute for the variable whose value is obtained via backward chaining:

```
depth first backward chain
```

2 Specify a value for the depth-first-backward-chaining-precedence attribute in each rule that concludes a value for the variable in its consequent.

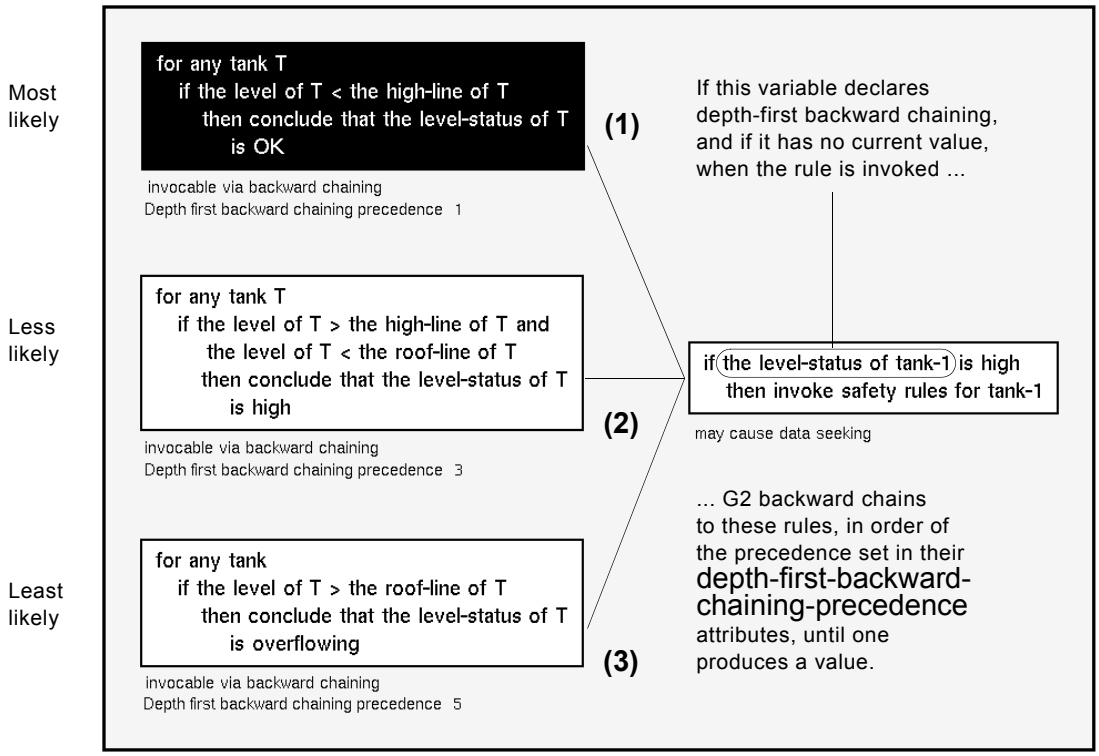
G2 invokes rules with the higher precedence (lowest number) first.

3 Specify backward chaining in the rules by following the steps under Backward Chaining.

For example, suppose a valve has a level, which is a measure of its flow. You might want to determine the status of its flow, that is, whether the flow is ok, high, or overflowing. You might use distinct rules, each implementing a different heuristic, to set the level-status attribute of the valve.

This figure shows three rules that implement heuristics for determining the level-status of a valve. The precedence order is shown in the figure and indicates the likelihood of finding a result.

Depth-First Backward Chaining



G2 invokes a rule with the next lower precedence only if rules with higher precedence have completed without concluding a new value for the variable.

G2 waits for a rule to complete before invoking the rule with the next lower precedence, even if that rule must backward chain to other rules. After each rule is invoked, G2 schedules it for execution at its declared priority.

Note The *precedence* for a rule for depth-first backward chaining differs from its *priority*. Precedences determine the order in which G2 invokes a set of rules that provide a new value for the same variable. Priorities determine which tasks can wait if G2 is fully loaded with activities to perform in a particular G2 clock tick.

Activating the Parent Workspace of a Rule

G2 invokes an initially rule each time the parent KB workspace of a rule is activated. For a description of initially rules, see Initially Rules.

When G2 starts the current KB, G2 activates all enabled top-level workspaces. Next, G2 activates in turn all enabled KB workspaces that are in the workspace hierarchy below the activated top-level workspaces. However, when G2 starts the current KB, G2 does not automatically activate subworkspaces that are configured as activatable subworkspaces.

For more information about how G2 activates workspaces, see Activating and Deactivating Workspaces.

Detecting Events

G2 can invoke certain rules after detecting an event. This mechanism is called **event detection**.

G2 invokes whenever rules after detecting the event specified in the antecedent of a rule. For example:

```
whenever the temperature of tank-1 receives a value
    then start update-maximum-temperature (tank-1)
```

G2 can invoke a whenever rule in response to these events:

- A variable, parameter, or attribute receives a value.
- A variable fails to receive a value.
- A variable loses its value.
- An item is created.
- An item is moved on a workspace.
- An item is resized by user on a workspace.
- An item is enabled or disabled.
- An item is activated or deactivated.

- Two items become or cease to be related by a specific relation.
- Two items become or cease to be related by any relation.
- Two items become or cease to be connected.

For the syntax used for each of these events, see Whenever Rules.

Scanning Rules

For rules that contain a value in the scan-interval attribute, G2 invokes rules once per the specified time interval. This mechanism is called **scanning**, because G2 invokes rules due to the passage of time, as opposed to based on the state of knowledge in your KB.

To invoke a rule by scanning:

➔ Specify a time interval in the scan-interval attribute of the rule.

For example, if you want G2 to check the temperature of tank-4 every five minutes, you can assign a scan-interval of 5 minutes for the following rule:

```
if the temperature of tank-4 > 40 F
    then inform the operator that "Tank-4 is overheating."
```

G2 invokes this rule every five minutes; thus, every five minutes, G2 finds the current temperature of the tank, compares it to 40 F, and tells the operator if the tank is overheating. In this case, F (representing degrees fahrenheit) has been defined as a unit of measure.

Note You cannot set the scan-interval attribute for an initially or whenever rule.

Determining the Scan Interval to Use

The time interval at which G2 starts performing a new set of scheduled tasks is determined by the value of the minimum-scheduling-interval attribute of the Timing Parameters system table. G2 cannot perform rule scanning more often than the minimum scheduling interval, when this value is a number.

This means that you must determine which time interval is most significant for your KB processing: the minimum-scheduling-interval attribute, which affects all activities that G2 performs for your KB, or the scan-interval attribute of a scanned rule. If some scanned rule must be invoked more often than the minimum scheduling interval, you must either decrease the setting of the minimum-scheduling-interval attribute or increase the scan-interval setting for the most frequently scanned rule or rules.

To determine the rate at which G2 must service scanned rules, you should first identify those rules. Next, determine which of those rules has the shortest time interval (that is, the smallest value) in its scan-interval attribute. This value

represents the slowest rate that the G2 scheduler can schedule its activities and still invoke scanned rules.

Note You should not attempt to set the value of the scan-interval attribute of a rule to a time interval shorter than (smaller than) the time interval found in the minimum-scheduling-interval attribute of the Timing Parameters system table.

Scanning Versus Event Detection

Scanning is a less efficient way of invoking rules than event detection. Invoking rules by scanning causes G2 to perform activity that might not have any relationship to the condition expressed in each scanned rule.

If G2 scans many rules, in some circumstances, the performance of the KB might be constrained. An important goal of analyzing your application is to identify the items whose knowledge directly depends upon the passage of a fixed time interval. Only those items should be manipulated due to invoking scanned rules.

Your application will be easier to understand and maintain, and will perform with less overhead, if you define the majority of the items in your application so that they respond to events other than the passage of a fixed time interval. For more information, see Detecting Events.

Scanning Generic Rules

Avoid coding generic rules that are invoked via scanning. For each generic rule that is scanned, at the beginning of that rule's scan interval, G2 invokes one copy of that rule for each item or value in the set identified in the generic reference expressions in the antecedent. For more information, see Determining the Number of Generic Rules That Are Invoked.

Focusing on Rules and Invoking Rules by Category

If the majority of your rules are generic, you can invoke these rules so that they apply only to a particular item or set of items. This is called **focusing**. G2 supports two mechanisms for invoking rules by focusing:

- Focusing on rules associated with particular items or classes of items
- Focusing on rules defined to be within a particular category

Focusing on Items

The focus action names as its argument a **focal item** or **focal class** of items. Executing the focus action causes G2 to invoke all rules that are associated with that focal item or focal class.

To associate a rule with a focal item:

➔ Specify one or more items in the focal-objects attribute of the rule.

Specifying a focal object might be appropriate for a specific rule that refers only to particular items. However, you can also associate a generic rule with a focal object.

To associate a rule with a focal class:

➔ Specify one or more item classes in the focal-classes attribute of the rule.

Specifying a focal class might be appropriate for a generic rule that refers to a set of items or values.

When G2 invokes a generic rule due to focusing, G2 applies the rule to each item that is an argument of the focus action.

For example, suppose you have this generic rule:

```
if the status of any tank T is ready
    then change the background-region of T to the ready-color of T
```

Suppose water-tank-1 is specified as the focal-objects attribute of the rule. When G2 executes this action, G2 invokes the rule for the instance named water-tank-1:

```
focus on water-tank-1
```

Now suppose tank is specified as the focal-classes attribute of the rule. If water-tank is a subclass of tank, then when G2 executes this action, G2 invokes the rule for all instances of the water-tank class:

```
focus on water-tank
```

The focus action can also act upon a set of items. For example, when G2 executes the following action, G2 invokes the rule for every water tank upon the parent workspace of the item that specifies this focus action:

```
focus on every water-tank upon this workspace
```

Invoking Rules by Category

The invoke action takes as its argument one or more rule categories. This action causes G2 to invoke all rules that are associated with the named category.

To associate a rule with a rule category:

➔ Specify one or more symbols in the categories attribute of the rule.

Note A rule category exists only when it is named in the categories attribute of any enabled rule.

For example, suppose you have the following rule and have specified safety and quality in its categories attribute:

```
if any tank T is overheated
    then inform the operator that
      "The tank [the name of T] is hot."
```

By associating this rule with the safety rule category, you capture the knowledge that an overheated tank is unsafe. By associating this rule with the quality rule category, you capture the knowledge that an overheated tank reduces the quality of the product that passes through that tank.

Executing the following action causes G2 to invoke all rules that specify safety in their categories attribute, and that specify tank-1 in their focal-objects attribute or tank-1 or any superior class of tank-1 in their focal-classes attribute.

```
invoke safety rules for tank-1
```

If G2 executes an invoke action whose argument is a set of items, G2 invokes one copy of the rule for each named rule category and for each item in the set.

For example, executing the following action causes G2 to invoke one copy of each rule associated with the safety and quality rule categories for each item upon the parent KB workspace of the item that contains the action:

```
invoke safety and quality rules for any item upon this workspace
```

If executing an invoke action causes G2 to invoke a generic rule, G2 invokes one copy of that rule for each item in the set of items named as the argument to the invoke action.

Waiting for Rules to Complete When Invoked from a Procedure

G2 provides the ability to invoke a category of rules, or to focus on a class or object from a procedure and have the procedure wait until the invocation completes before continuing. If the left-hand side of the rule matches the given category as specified by the invoke action or the given focal class or object as triggered by the focus action, the procedure waits until the right-hand side of the rule completes before continuing execution. The right-hand side of the rule might include actions that occur immediately, such as conclude, or they might include scheduled actions, such as start. Note that any scheduled side-effect of the right-hand-side of the rule might or might not occur before the invoking procedure wakes up, which is based on the relative priorities of the various tasks. Side-effects include the start action or any additional forward-chaining caused by a conclude action.

To cause the procedure to wait until the rule completes, use the , awaiting completion after the invoke action.

For example:

```
invoke safety rules, awaiting completion
```

You can also use this new grammar with the focus action, for example:

```
focus on tank, awaiting completion
```

Note that if the rule never completes, the invoking procedure never wakes up. To cause the procedure to wake up if the rule never completes, add a do in parallel statement, as follows:

```
do in parallel until one completes
     invoke safety rules, awaiting completion;
     wait for 10 seconds;
end
```

Debugging Rules

You can use G2's system-defined facility for debugging and tracing rules. In addition, you can highlight invoked rules, which is helpful for debugging.

Debugging and Tracing Rules

G2 provides a facility for producing messages that indicate when the invocation and execution of a rule begins and ends. This facility is described in Debugging and Tracing.

To specify debugging and tracing for rules:

1. Specify a custom settings in the tracing-and-breakpoints attribute of the rule.
2. Set the attribute tracing-and-breakpoints-enabled? to yes in the Debugging Parameters system table.

This setting overrides the global settings for tracing and warning messages and for breakpoints found in the Debugging Parameters system table, as described in Debugging Parameters.

Highlighting Rules

You can direct G2 to highlight each rule that it invokes. *Highlighting* a rule means to change momentarily the appearance of its text box representation, so that its text appears in white on a dark background.

To enable rule highlighting:

➔ Select Main Menu > Run Options > Highlight Invoked Rules.

To disable rule highlighting:

➔ Select Main Menu > Run Options > Do Not Highlight Invoked Rules.

You enable and disable highlighting for all rules in the current KB.

If rule highlighting has been enabled, for each rule that G2 highlights, G2 pauses for three-tenths of a second, to allow you to identify that rule. Thus, G2 runs the current KB at a slightly slower speed when rule highlighted is enabled.

If invoking a rule causes backward chaining to other rules, G2 leaves that rule highlighted until its antecedent is fully evaluated (or until that rule times out), then highlights in turn each rule to which it chains. G2 restores the appearance of the chained to rules in the reverse order in which they were invoked.

Note Highlighting is not intended to portray all invocations of the current KB's rules. Due to G2's optimizations for evaluating the antecedents of rules, G2 does not always highlight each rule whose antecedent is, in fact, checked but not fully evaluated. Simple condition expressions, such as the status of my-variable is ok, allow G2 to optimize its own behavior and not evaluate the entire antecedent.

Understanding Rule Invocation and Execution

To *invoke* a rule means to begin evaluating its antecedent. G2 can invoke a rule by using the mechanisms described in Invoking Rules.

When G2 invokes a rule, G2 creates a task called a **rule invocation**. Each rule invocation is a copy of the information in the invoked rule.

By default, G2 creates one rule invocation for each item or value in the set that the antecedent of the rule identifies. Thus, for a specific rule, G2 creates one rule invocation; for a generic rule, G2 creates one rule invocation for each item or value in the set identified in the generic reference expressions of the antecedent.

To *execute* a rule means to begin performing the actions in the consequent of a rule. G2 executes a rule if the antecedent of the rule produces a value of true. If you are using fuzzy truth values, the value of true must be greater than or equal to the truth-threshold attribute of the Inference Engine Parameters system table.

Prioritizing Rules

Every rule has a priority that determines how G2 schedules rule invocations for that rule. By default, the scheduler executes rules at a priority of 6.

To override the default priority for executing a rule:

➔ Specify a value for the rule-priority attribute of the rule.

This attribute accepts a value from one (1) to ten (10), with one indicating the highest priority.

Scheduling tasks by priority becomes important in the unusual case when G2 is working at maximum capacity. G2 can postpone the execution of the lowest

priority tasks until the next G2 clock tick. Therefore, you should use priorities to identify what tasks can safely be postponed if necessary.

Each time G2 completes a task, G2 starts executing the next highest task in the task list for the current G2 clock tick. If a task with priority 1 comes in while G2 is performing a priority 2 task, then after the priority 2 task completes, G2 starts executing the priority 1 task.

Rules and Scheduler Tasks

An internal component of G2, called the scheduler, is responsible for managing which activities G2 actually performs starting at each tick of the G2 clock.

At each G2 clock tick, G2 begins performing the tasks that the scheduler has associated with that clock tick. For a given clock tick, G2 begins performing tasks with a higher priority before those with a lower priority. However, G2 must suspend some tasks, or put them into a waiting state, before they can complete. This means that a task with a high priority that has already begun, and that must be suspended for some reason, might not complete before another task with a lower priority begins *and* completes.

One example of this is the activity of evaluating the antecedent of a rule that can backward chain to other rules. Evaluating the antecedent of a rule might require invoking another rule that concludes a value for a variable or that causes data seeking.

Rule Priorities and Rule Completion

The priorities of rules do not affect the order in which G2 *completes* particular rule invocations. That is, after G2 invokes a rule with a high priority, G2 might require its rule invocation to wait for a value as G2 evaluates the antecedent of the rule. While this rule invocation waits, another rule invocation with lower priority (that is, one that is not required to wait for the values that it uses) can begin to execute *and* can complete.

Propagation of Rule Priorities

When backward chaining takes place, the priority of rules propagates from the invoking rule to the invoked rule. This means that if two rules with different priorities backward chain to the same rule, G2 invokes a rule invocation for the third rule at the higher of the two invoking rules' priorities.

For example, if rule R1 is declared to have a rule-priority of 3, and rule R2 is declared to have a rule-priority of 5, and both rule R1 and R2 backward chain to rule R3 that declares a rule-priority of 8, then G2 creates and invokes a rule invocation for rule R3 that has the priority 3.

If G2 cancels the rule invocation for rule R1 for any reason, G2 does not reschedule rule R3 with a rule-priority of 5; rule R3 retains its rule-priority setting of 3.

Note The priorities of rules do not propagate when forward chaining takes place or when a rule is invoked due to a focus or invoke action.

Setting the Timeout Interval for a Rule

You can specify a timeout interval for a rule to specify how long G2 allows the rule invocation to execute after it is invoked. If a rule did not specify a timeout interval, then after being suspended, it could reawaken long after the conditions that caused G2 to invoke it have passed from the KB's knowledge.

To set the timeout interval for a rule:

➔ Specify a time interval for the timeout-for-rule-completion attribute of the rule.

The value of this attribute in the rule overrides the value of the timeout-for-inference-completion attribute in the Inference Engine Parameters system table. G2 uses the latter attribute as the timeout interval for rules whose timeout-for-rule-completion attribute contains the value use default.

Depending on the nature of the values that your rules and procedures manipulate, setting long time intervals for the completion of your rules can lead to inconsistencies in your KB's knowledge.

For instance, if the values of variables are periodically unavailable to your rules, you should create whenever rules that respond to the event expression does not receive a value. A variable does not receive a value when its validity interval has expired, as declared in its validity-interval attribute.

Creating and Managing Rule Invocations

G2 begins performing a rule invocation when that task reaches the top of the task queue for that rule's declared (or propagated) priority. Note that one rule's priority can propagate to other rules, as described in Prioritizing Rules.

G2 executes each rule invocation in several stages, as follows:

1. G2 evaluates the truth-value expression (whether it is explicit or implied) in the antecedent of the rule.
2. If the evaluation of the antecedent cannot be completed within the timeout-for-rule-completion attribute of the rule, G2 performs the appropriate time-out processing. See Evaluating the Antecedent.
3. If the antecedent evaluates to a valid truth-value, the evaluation does not time out, and the consequent of the rule specifies the in order phrase, then G2 performs the actions in the consequent sequentially.

4 If the antecedent evaluates to a valid truth-value, the evaluation does not time out, and the consequent of the rule does not specify the in order phrase, then G2 executes the actions in the consequent in parallel.

5 If any action in the consequent cannot be completed within the declared timeout-for-rule-completion time interval, G2 performs the appropriate time-out processing.

6 If the antecedent evaluates to false, G2 does not perform the actions in the consequent, and the rule invocation completes.

The rest of this section describes how G2 performs each portion of a rule invocation.

Evaluating the Antecedent

G2 begins performing a rule invocation task by attempting to evaluate the antecedent of the rule. When the rule has a compound antecedent, G2 will first look for any variable that has an immediate value, regardless of where it is in the expression; then if any other variables require data-seeking, these will be evaluated in left-to-right order. Evaluating the antecedent brings about one of three results:

- The antecedent evaluates to true, so G2 begins executing the actions in the consequent.
- The antecedent evaluates to false, and the rule invocation task completes without executing any action in the consequent.
- One or more variables in the antecedent do not have a current value. G2 suspends this rule invocation and sets a wake-up flag on each variable that needs a new current value. If any of the variables receives a value, G2 wakes up the rule invocation and tries again to evaluate the entire antecedent.

In the unusual case where the time required to evaluate the antecedent exceeds the timeout-for-rule-completion attribute setting, G2 performs time-out processing. This means G2 makes one final attempt to evaluate the complete antecedent:

- If this final attempt succeeds, G2 continues performing this rule invocation by executing the actions in the consequent of the rule, as described in the next two sections.
- If this final attempt fails, G2 cancels the entire rule invocation without executing the consequent. Note that G2 does *not* consider this cancellation to be an error condition; G2 does *not* signal an error.

Executing Actions in the Consequent in Parallel

When G2 begins executing the consequent of a rule, it first determines whether the phrase in order is present. If the phrase is not present, G2 first attempts to

evaluate all the expressions in all the actions in the consequent. After G2 successfully evaluates all expressions in the consequent, G2 schedules all actions to be performed in parallel, in other words, within the same transaction scope. For information about the scope of a transaction, see Understanding Transactions and Transaction Scopes.

Evaluating all the expressions in the consequent might require G2 to suspend this rule invocation, so that G2 can perform data seeking to obtain values.

Time-Out Processing

If G2 cannot evaluate all the expressions in the consequent within the declared timeout-for-rule-completion time interval, G2 performs time-out processing. This means that G2 makes a final attempt to evaluate all the expressions in the consequent as follows:

- If this final attempt succeeds, G2 performs the actions in the consequent as described in the next section.
- If this final attempt fails, G2 cancels this rule invocation without completing its execution of the consequent. Note that G2 does *not* consider this cancellation to be an error condition; G2 does *not* signal an error.

Single Transaction Scope for All Consequent Actions

When G2 executes the actions in the consequent in parallel, all actions execute in the same transaction scope.

This means that each action begins executing with the same context of information. For example, if the expressions in any two consequent actions refer to the same item or value, G2 evaluates those expressions by starting with the same set of items or values.

Example 1: The following rule includes two actions that increment the variable X:

```
unconditionally
    conclude that X = X + 1 and conclude that X = X + 1
```

In this case, after this rule completes successfully, the value of X will be incremented only by 1. G2 executes the actions in the consequent in parallel and evaluates all expressions in the consequent by using values in effect at the time the rule was invoked.

Example 2: The two actions in the next rule increment the variable X by differing amounts:

```
unconditionally
    conclude that X = X + 1 and conclude that X = X + 3
```

However, even though the two conclude actions start with the same source value for X, when the rule completes successfully, the value of X will be incremented by 3. this is because, among actions that execute in parallel and that update the same

value, the action specified *last* in the rule determines the new value after the rule completes.

Example 3: The consequent actions in this rule refer to the variable that gives the value of the minimum-temperature of tank-1:

```
if the temperature-measured of tank-1
    then conclude that
      the minimum-temperature of tank-1 = min (the temperature of tank-1,
      the minimum-temperature of tank-1)
      and conclude that the global-minimum-temperature of tank-monitor
      = min (the minimum-temperature of tank-1, the minimum-temperature of
      tank-monitor)
```

Because G2 executes the actions in this consequent in parallel, G2 supplies the same value for the minimum-temperature of tank-1 for each conclude action.

If the purpose of the global-minimum-temperature of tank-monitor is to store the minimum temperature ever recorded for any tank monitor in the KB, then this rule concludes the value of the global-minimum-temperature of tank-monitor incorrectly. This is because the second conclude action uses a value for the minimum-temperature of tank-1 that does not include the calculation performed in the first conclude action. Therefore, this rule should be rewritten by using sequential execution (described below).

Also, because G2 executes consequent actions within one transaction scope, after the actions start executing, G2 allows no other KB processing to take place until all the consequent actions finish executing.

Executing Actions in the Consequent Sequentially

When G2 begins executing the consequent of a rule invocation, it first determines whether the phrase in order is present. If it is present, G2 prepares to perform the actions in the consequent in sequence. G2 evaluates the expression in the first action in the consequent, then executes the action, and so on for every action in the consequent.

When performing the actions in the consequent of a rule sequentially, G2 executes each action within its own transaction scope, as described in One Transaction Scope per Consequent Action.

If executing any consequent action depends upon obtaining a new current value for any variable, G2 suspends this rule invocation and sets a wake-up flag on each variable without a current value that is referenced in that action.

When any variable with a wake-up flag receives a new current value, G2 wakes up the rule invocation and reevaluates the expressions in the action whose execution was suspended. In this case, G2 does *not* reevaluate the antecedent of the rule, and it does not reevaluate the same consequent actions that have already been performed.

Time-Out Processing

If G2 cannot perform all consequent actions in sequence within the declared timeout-for-rule-completion time interval, G2 performs time-out processing. For instance, if a rule invocation is suspended because an action contains an expression that refers to a variable, and the action has been waiting for a new current value for that variable, G2 wakes up the rule and performs time-out processing, as follows:

- G2 attempts to execute any inform actions without having the values of all variables; G2 displays expired current values for variables with asterisks.
- After discarding any consequent actions that have not yet been performed, G2 completes the rule.

One Transaction Scope per Consequent Action

When performing the consequent actions of a rule sequentially, G2 executes each action within its own transaction scope. This means that before executing each consequent action, G2 evaluates the expressions within the actions by using values that reflect the execution of any previous consequent action in this rule.

For example, G2 performs these consequent actions sequentially:

```
for any demonstration-window DW
    if the subworkspace of DW exists
        then in order
            change the arrow-region icon-color of DW to yellow and
            start populate-demonstration-frame ( the initial-item-count
              of DW )
```

This rule helps prepare the subworkspace of an item to display a dynamically generated set of items. In this case, the initial-item-count attribute of a demonstration window is given by an integer variable.

Because each consequent action in this rule takes place within its own transaction scope, this rule cannot prevent other KB processing from taking place between the completion of the change action and the completion of the start action.

In this case, if the variable that gives the value of the initial-item-count of the demonstration window requires a new current value, G2 might be required to suspend this rule invocation while backward chaining or other data seeking takes place.

This means that G2 cannot prevent other KB processing from changing the color of the demonstration-window's arrow-region to a color other than yellow while the start action is suspended. Depending on the activity that the populate-demonstration-window procedure performs, this possibility might leave some of the current KB's knowledge in an inconsistent state.

The Rule Class

Rules have the following class-specific attributes:

Attribute	Description
options	Declares whether the rule can be invoked using chaining, and also declares whether the rule can invoke other rules by using chaining. For any rule except a when or whenever rule, edit this attribute to declare participation in chaining to or from this rule.
Allowable values:	invocable via forward chaining not invocable via forward chaining invocable via backward chaining not invocable via backward chaining may cause data seeking may not cause data seeking may cause forward chaining may not cause forward chaining
Default value:	if rules, initially rules, and unconditionally rules: invocable via forward chaining invocable via backward chaining may cause data seeking may cause forward chaining when rules and whenever rules: not invocable via forward chaining not invocable via backward chaining may cause data seeking may cause forward chaining
Notes:	See Backward Chaining and Forward Chaining. By setting this attribute to not invocable via forward chaining and not invocable via backward chaining, you can make an if rule behave like a when rule.

Attribute	Description
tracing-and-breakpoints	Declares the message levels for warning messages and tracing messages, and declares the breakpoint level. These settings pertain only to this rule. Use warning messages, tracing messages, and breakpoints when debugging your KB.
Allowable values:	default or, optionally, one of the following warning message levels: warning message level 0 (no warning messages) warning message level 1 (KB errors only) warning message level 2 (KB errors and deficiencies) warning message level 3 (KB errors, deficiencies, and other conditions) and, optionally, one of the following tracing message levels: tracing message level 0 (no trace messages) tracing message level 1 (trace messages on entry and exit) tracing message level 2 (trace messages at major steps) tracing message level 3 (trace messages at every step) and, optionally, one of the following breakpoint levels: breakpoint level 0 (no breakpoints) breakpoint level 1 (breakpoints on entry and exit) breakpoint level 2 (breakpoints at major steps) breakpoint level 3 (breakpoints at every step)
Default value:	default
Notes:	See Debugging and Tracing. A value of default in this attribute directs G2 to use the settings in the warning-message-level attribute, tracing-message-level attribute, and breakpoint-level attribute of the Debugging Parameters system table. The tracing-and-breakpoints-enabled? attribute of the Debugging Parameters system table must have the value yes for G2 to produce tracing messages and to recognize breakpoints.

Attribute	Description
scan-interval	Specifies how often G2 should invoke this rule. G2 ignores this attribute for initially rules and whenever rules.
Allowable values:	none Any time interval
Default value:	none
Notes:	See Scanning Rules. If a generic rule has a scan interval, G2 invokes each generic rule invocation at the beginning of each scan interval.
focal-classes	Associates the rule with one or more classes of items. By executing a focus action that names a class of items, your KB can invoke all rules whose focal-classes attribute names that class.
Allowable values:	none item Name of any subclass of item class
Default value:	none
Notes:	See Focusing on Rules and Invoking Rules by Category.
focal-objects	Associates the rule with one or more named items. By executing a focus action that names an item, your KB can invoke all rules whose focal-objects attribute names that item.
Allowable values:	none Name of any item
Default value:	none
Notes:	See Focusing on Rules and Invoking Rules by Category.

Attribute	Description
categories	Names one or more rule categories that pertain to this rule. When executing an invoke action that names a rule category as its only argument, G2 invokes all rules whose categories attribute names that category. When executing an invoke action that names both a rule category and an item or an item class, G2 invokes all rules whose categories attribute names that category *and* whose focal-objects attribute name that item or whose focal-classes attribute names that class. *Allowable values:* none Any non-reserved symbol *Default value:* none *Notes:* A rule category exists only if named in at least one enabled rule. See Focusing on Rules and Invoking Rules by Category.
rule-priority	Specifies the priority for the rule invocation that G2 creates when this rule is invoked. *Allowable values:* Literal integer, from 1 (highest priority) to 10 (lowest priority) *Default value:* 6 *Notes:* See Prioritizing Rules. A rule's priority does not control the *ordering* of G2 scheduler tasks. Priorities propagate from rule to rule due to backward chaining. Priorities do not propagate due to forward chaining, or due to invoking a rule by executing a focus or invoke action.

Attribute	Description
depth-first-backward-chaining-precedence	Specifies the precedence among a group of rules that G2 identifies for execution by using depth-first backward chaining. This attribute does not affect breadth-first backward chaining.
Allowable values:	Literal integer, 1 or greater
Default value:	1 (highest priority)
Notes:	By default, depth-first backward chaining causes G2 to invoke rules with lower precedence after rules with higher precedence have completed. Precedence differs from rule priority. For more information, see Using Depth-First Backward Chaining.
timeout-for-rule-completion	Determines how long G2 attempts to invoke and execute a rule invocation before performing time-out processing.
Allowable values:	none (the rule never times out) use default (use the value in the timeout-for-inference-completion attribute of the Inference Engine Parameters system table) Any time interval
Default value:	use default

Actions That Manipulate Rules

To invoke rules associated with an item or class:

➔ focus on {*item* | *item-class*}, awaiting completion

Invokes all rules that are associated with the specified item or item class.

invoke *rule-category-name* [{ , | or } *rule-category-name*] ... rules
for { *item* | *item-class* }, awaiting completion

Expressions That Refer to Rules

To refer directly to a rule:

➔ this rule
-> *rule*

This expression produces the invoked rule within which this expression is evaluated. This expression is valid only in a rule. For example:

```
if the status of any custom-object O
    upon the workspace of this rule is not ok
    then inform the operator that
     "An error has occurred; press the DIAGNOSIS button."
```

This whenever rule detects events that are associated with items on its own workspace.

25

Formulas

Describes generic and specific formulas and their use.

Introduction

A **formula** is an equation that provides values for a variable or parameter. G2 computes a formula only when a value is needed. G2 provides these kinds of formulas:

- **Generic formulas**, which you can specify for a class of variables by creating a generic-formula definition. The generic-formula class is a subclass of the statement class.
- **Specific formulas**, which you specify in the formula attribute of a single variable.
- **Simulation formulas**, which are used with the G2 Simulator, a superseded capability. For more information, see Appendix F, Superseded Practices.

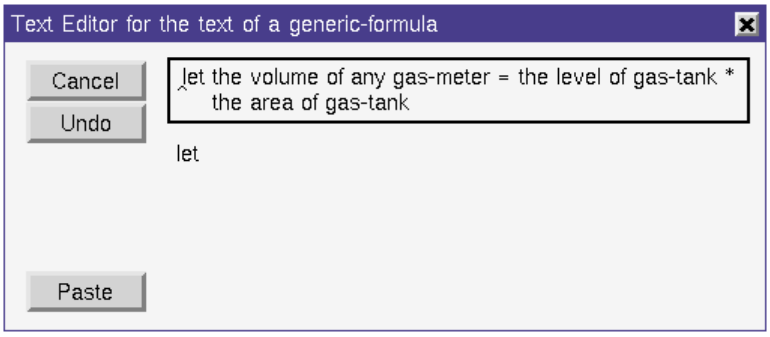
Creating Generic Formulas

A **generic formula** applies to an entire class of variables. You can specify that an attribute of any class evaluates to a certain expression.

To create a generic formula:

➔ Select KB Workspace > New Definition > generic-formula.

G2 invokes the Text Editor so that you can enter a formula beginning with the word let, as follows:



When you complete the edit, the new formula appears in a statement box connected to your cursor. Press to place it on a workspace. G2 uses this generic formula to calculate the volume of any gas-meter, if no specific formula exists for the variable.

When writing formulas, a general rule of thumb is to use generic rather than specific formulas whenever possible. This lets you write one formula for an entire class of variables rather than a formula for each variable in the class. A generic formula for a class of variables applies to all classes of variables below it in the class hierarchy; thus, you should try to write generic formulas for classes as far up in the item hierarchy as possible.

Creating Specific Formulas

A **specific formula** is a formula that applies to just one variable. You give a variable a specific formula by specifying the formula attribute in its attribute table. G2 then uses this specific formula to calculate a value for the variable.

To create a specific formula for a variable:

➔ Edit the formula attribute of the variable.

You create a specific formula for numeric, truth-valued, symbolic, and text variables by using any arithmetic, logical, symbolic, or text expression. Here is a partial attribute table with the formula attribute filled in:

MILES-PER-GALLON, a quantitative-variable	
Options	do not forward chain, breadth first backward chain
Notes	OK
Item configuration	none
Names	MILES-PER-GALLON
Tracing and breakpoints	default
Data type	quantity
Initial value	none
Last recorded value	no value
History keeping spec	do not keep history
Validity interval	1 second
Formula	the miles-travelled of tractor-1/the gas-used of tractor-1
Simulation details	no simulation formula yet
Initial value for simulation	default
Data server	inference engine
Default update interval	none

Specific formula for a variable

G2 uses this formula to calculate a value for the variable if the data-server for the variable indicates inference engine. If the data-server attribute is G2 simulator, G2 uses the formula specified in the simulation-formula attribute in the subtable of the simulation-details attribute.

The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices.

26

Text Parsing and Manipulation

Describes capabilities for manipulating text and substrings, parsing and tokenizing text using regular expressions, and interconverting text between the Gensym and Unicode character sets.

Introduction

G2 provides a variety of capabilities for manipulating text:

- Functions for searching for, extracting, and replacing text in strings.
- A syntax for regular expressions.
- Functions that use regular expressions for searching and modifying text.

- A facility that uses regular expressions to parse strings into tokens.
- Text-processing functions for use with encoding character codes for importing and exporting text.

This chapter describes all of the capabilities listed, and shows you how to use them to process text in a knowledge base.

G2 Text Manipulation Functions

G2 provides text manipulation functions that:

- Perform simple text manipulation, as described under Ordinary Text Manipulation Functions
- Use regular expressions to perform more complex text manipulation, as described under Text Functions Using Regular Expressions
- Use regular expressions to parse a string and identify tokens, as described under Locating Tokens in a String.

The text functions available in G2 are non-destructive: they leave their input arguments intact. G2 functions in general are described in Functions.

G2 Conventions for Manipulating Text

G2 provides a character string data type, text, but lacks a built-in type to refer to characters. However, conventions exist that provide a consistent way to deal with substrings and individual characters:

- The first position in a string is position 1 (*not* position 0).
- The last position in a substring reference is *inclusive*. For example:

  ```
  get-from-text ("abcdef", 2, 3) -> "bc"
  ```

- The length of a string does not include any escape (@) characters that appear in the string when G2 displays it in the Text Editor. Such characters exist only in the editor display: they are not stored as part of the string.

  ```
  length-of-text ("@"") -> 1
  ```

- The length of a string *does* include any extra characters kept internally to represent diacritics (Unicode non-spacing marks).
- The index 0 is used as a sentinel to indicate failure:

  ```
  position-of-text ("A", "XYZ") -> 0
  ```

Arguments that provide character positions to G2 text-manipulation functions must be of type integer. Using a quantity or a float causes G2 to signal an error, even of the value is integral, for example, 2.0.

For information on using square brackets to get the Unicode character code of a single character in a text, see Getting Unicode Character Codes.

Ordinary Text Manipulation Functions

G2 provides ordinary text-manipulation functions similar to those that exist in most programming languages, as described in this section.

Text functions in general are described under G2 Text Manipulation Functions and G2 Conventions for Manipulating Text. G2 functions in general are described in Functions.

For the functions get-from-text, replace-in-text, and omit-from-text, the following conditions must be true; otherwise, the function signals an error:

- 1 <= *start-index* <= *length-of-text* + 1
- *start-index* - 1 <= *end-index* <= *length-of-text*

Obtaining Text Length

length-of-text
(*text-expression:* text)
-> *length:* integer

Returns the number of characters in *text-expression*. Examples:

```
length-of-text ("message") = 7
length-of-text ("") = 0
```

Testing for a Substring

is-contained-in-text
(*text-expression1:* text, *text-expression2:* text)
-> *substring-exists:* truth-value

Returns the true if *text-expression1* is a substring of *text-expression2*, and false if it is not. This function is not case sensitive. Example:

```
is-contained-in-text ("your", "Your flight") = true
```

Locating a Substring

position-of-text
(*text-expression1:* text, *text-expression2:* text)
-> *start-position:* integer

Returns the starting position of the first occurrence of *text-expression1* in *text-expression2*, or 0 if no such occurrence exists. Example:

position-of-text ("fli", "Your flight") = 6

Obtaining a Substring

get-from-text
(text-expression: text*, start-index:* integer*, end-index:* integer*)*
-> *substring:* text)

Returns the string of characters extracted from *text-expression* beginning at *start-index* and ending at *end-index*. Spaces between words are included in the count from left to right. Assuming the *start-index* and *end-index* are both valid, this function returns the empty string if and only if *end-index* = *start-index* - 1; otherwise, the function returns a non-empty string. Examples:

get-from-text ("one two three", 5, 7) = "two"

get-from-text ("abcd",5,4) = ""

get-from-text ("abcd",3,2) = ""

Inserting a Substring

insert-in-text
(text-expression1: text*, text-expression2:* text*, insert-index:* integer*)*
-> *combined-text:* text

Returns a text value consisting of *text-expression2* with *text-expression1* inserted before the character at *insert-index*. Example:

insert-in-text (" not ", "do enter", 3) = "do not enter"

Replacing One Substring with Another

replace-in-text
(*text-to-substitute*: text, *source-text*: text,
start-index: integer, *end-index*: integer)
-> *modified-text*: text

Returns the *source-text* with the substring from *start-index* to *end-index* replaced by *text-to-insert*. Assuming the *start-index* and *end-index* are both valid, this function simply inserts the *text-to-substitute* into the *source-text* when *end-index* = *start-index* - 1. Examples:

replace-in-text ("exit", "do not enter here", 8 , 12) = "do not exit here"

replace-in-text("come ","to",1,0) = "come to"

replace-in-text (" come","to",3,2) = "to come"

replace-in-text ("and","nip tuck",5,4) = "nip and tuck"

Deleting a Substring

omit-from-text
(text-expression: text*, start-index:* integer*, end-index:* integer*)*
-> *remaining-text*: text

Returns the *text-expression*, with the range of characters between *start-index* and *end-index* omitted. Spaces between words are included in the count from left to right. Assuming the *start-index* and *end-index* are both valid, this function makes no change when *end-index* = *start-index* - 1. Examples:

omit-from-text ("do not enter",4,7) = "do enter"

omit-from-text("do not enter",8,7) = "do not enter"

Capitalizing Text

capitalize-words
(*text-expression:* text)
-> *capitalized-string:* text

Returns *text-expression* with the first letter of each word capitalized. Example:

capitalize-words ("this is a test") = "This Is A Test"

Converting Text to Uppercase

upper-case-text
(*text-expression:* text)
-> *uppercased-text:* text

Returns *text-expression* with all alphabetic characters in upper-case. Example:

upper-case-text ("123AbcDef") = "123ABCDEF"

Converting Text to Lowercase

lower-case-text
(*text-expression:* text)
-> *lowercased-text*: text

Returns *text-expression* with all alphabetic characters in lowercase. Example:

lower-case-text ("123AbcDef") ="123abcdef"

Testing for a Quantity

text-begins-with-quantity
(*text-expression:* text)
-> *quantity-present:* truth-value

Returns true if *text-expression* begins with one or more numeric characters; else false. Leading spaces, signs, or exponents are ignored. Examples:

```
text-begins-with-quantity ("123abc") = true
text-begins-with-quantity ("+456def") = true
text-begins-with-quantity ("abc123") = false
```

Regular Expression Syntax

G2 provides regular expressions that can be used to describe text flexibly and succinctly. The G2 regular expression syntax is similar to that used by UNIX utilities such as `lex`, `grep`, `sed`, `awk`, and others. G2 regular expressions are text strings (type text), and therefore appear in quotes when embedded in G2 code.

A regular expression is a sequence of characters and/or meta-characters that specifies a pattern that matches one or more possible character strings. The meta-characters used by G2 regular expressions are:

{ } \ | * + ? ^ . ( )

Characters represent themselves; meta-characters are operators that define expressions. For example, the meta-character '|' represents logical or, so the regular expression a|b matches either the character 'a' or the character 'b'. The following table gives the syntax of G2 regular expressions.

Basic Regular Expression Constructs

Example	Interpretation
abc	a followed by b followed by c.
(abc)	a followed by b followed by c; i.e., a synonym for abc. The difference is that, in larger strings, the meaning of abc can be affected by the context, whereas (abc) always means the same thing. See Precedence.
a|b	Either a or b. The vertical bar (|) means alternatives. A string with | as the first character is an invalid regular expression, as is a string with | as the last character, unless the next-to-last character is \.
a*	Zero or more occurrences of a. A string with an asterisk (*) as the first character is an invalid regular expression.
a+	One or more occurrences of a. A string with a plus sign (+) as the first character is an invalid regular expression.
a?	Zero or one occurrences of a. A string with a question mark (?) as the first character is an invalid regular expression.

Basic Regular Expression Constructs

Example	Interpretation
.	Any single character, including the newline character.
\.|*	A period (.) followed by | followed by \, followed by *. The \ character specifies that the character following the \ is to be interpreted literally. It is typically used to escape meta-characters, although it will work for any character (e.g., \a is a synonym for a.)
["@"]	This is just an ordinary string, with no meta-characters. It matches the sequence of characters '[' '"' '@' '"' ']'. However, remember that the standard meta-characters of G2 text will apply to strings even if they are intended to be regular expressions, so to enter this string in the editor, one would use the string "@[@"@@@"@]".
$(foo)	The $ character is used in the attributes of the Tokenizer class. The construction has no special meaning in other contexts.
^abc	The sequence of characters a, b, c, but only if found at the start-position given. The caret "anchors" the search. This feature only has meaning for system-defined functions. The Tokenizer's search is always anchored.

Caution Be careful not to confuse regular expressions with wildcard expressions for designating filenames. The two syntaxes use some of the same meta-characters, but their meanings are somewhat different.

Character Classes

Character classes provide a terse notation for indicating large sets of characters. G2 regular expressions use { and } as delimiters for character classes. G2 also provides several system-defined character classes, as described under System-Defined Character Classes.

Note Character classes are unrelated to item classes, which are classes in the object-oriented sense. Character classes are just sets of characters specified with a terse notation.

Character Class Constructs

Example	Interpretation
{abc}	The character a, or b, or c. Essentially, this usage is a shorthand for the notation (a|b|c).
{a-z}	Any character between a and z inclusive. Inside curly braces, the hyphen becomes a meta-character meaning a range of characters. Since "between" refers to the numerical values assigned to the characters in the character encoding, problems may arise with encodings, such as EBCDIC, that intersperse alphabetic and non-alphabetic characters.
{^a-z}	Any character which is *not* between a and z inclusive. A caret (^) immediately following a left curly brace introduces an inverted character class. The inversion refers to the entire class; i.e., the characters following the caret determine a match space, and the match space for the class becomes the set difference between the full alphabet and the computed match space. A caret inside a character class which is not the first character has no special meaning.
{}	The null string. A null string in a regular expression is legal but has no effect on the meaning of the string. Thus car, {}car, c{}ar, ca{}r and car{} are all equivalent.
<*charclass*>	Any alphabetic character in the system-defined character class named by *charclass*. All characters in between a < and its corresponding > are read as a symbol. If that symbol does not name a system-defined character class, an error results.

Caution A null string matches anything, so searching for it can cause an infinite loop. Guard against inadvertently creating and searching for it in iterative constructs that assemble regular expressions dynamically.

System-Defined Character Classes

G2 provides several system-defined character classes. Such a classes consists of the union of the appropriate characters over all natural languages supported by our character encoding. To refer to a system-defined character class in a regular expression, give the name of the class in <brackets>.

The system-defined character classes are as follows. Ranges are specified with semicolons, because no semicolon appears in any character class.

Character Class Name	Character Class Definition
alphabetic	A:Z a:z
numeric	0:9
alphanumeric	A:Z a:z 0:9
g2symbol	A:Z a:z 0:9 - _ " .
hexadecimal	0:9 A:F a:f
whitespace	space, newline, return, linefeed, line-separator, tab
double-quote	"
left-bracket	[
right-bracket	]

Precedence

The constructs used to indicate G2 regular expressions have a precedence order. This order defines the correct interpretation when constructs appear consecutively. Several levels of precedence exist. Constructs at the same precedence level are evaluated left-to-right.

The following table shows the construct(s) at each precedence level, and gives the correct interpretation of a sample regular expression that uses the construct(s):

Precedence

Level	Example	Interpretation
\	\{a-z}	The sequence of characters {, a, -, z, }
{...}	c{ad}+r	The character c, followed by one or more occurrence of either character a or character d, followed by character r.
(...)	(dog)*	Zero or more consecutive occurrences of the character sequence d, o, g, e.g., dogdogdog or "".
*, +, ?	dog*	d followed by o followed by zero or more g characters, e.g., dogggg or do.

Precedence

Level	Example	Interpretation
Implicit Sequence	ca|dr	Either (1) c followed by a or (2) d followed by r.
|	a|b	Either a or b.

Text Functions Using Regular Expressions

G2 provides various system-defined functions that find and/or replace text as designated by numeric arguments and regular expressions. For brevity, the names of such functions call a regular expression a **pattern**.

Text functions in general are described under G2 Text Manipulation Functions and G2 Conventions for Manipulating Text. G2 functions in general are described in Functions.

Locating a Substring Using a Regular Expression

find-next-pattern
(*search-pattern*: text, *source-text*: text, *start-position*: integer)
-> structure (*token-type*: the symbol goal,
start-index: integer, *end-index*: integer)

Searches through *source-text*, starting at *start-position*, for a substring that matches *search-pattern*, a regular expression, and returns a structure containing the *start-index* and *end-index* of the match, or:

```
structure (token-type: the symbol goal, start-index: 0, end-index: 0)
```

if no match exists. For example:

```
find-next-pattern ("{A-Z}{a-z}*" "according to Will Rogers", 1)
-> (token-type: the symbol goal, start-index: 14, end-index: 17)
```

Many substrings match the pattern. "W", "Wi", "Wil", and "Will" begin at position 14, and "R", "Ro", etc., begin at position 19. The function returns the start and end positions of the leftmost longest matching substring.

```
find-next-pattern ("{A-Z}{a-z}*" "according to Will Rogers", 15)
-> (token-type: the symbol goal, start-index: 19, end-index: 24)
```

"Will" is not a candidate, because the search starts at position 15. The function therefore returns the start and the end positions of "Rogers".

The token-type attribute can be ignored.

Extracting a Substring Using a Regular Expression

find-next-substring-matching-pattern
(*search-pattern*: text, *source-text*: text, *start-position*: integer)
-> *substring*: text

Identical to find-next-pattern, except that the function returns the matching substring itself, rather than the substring's start and end positions. If no match exists, the function returns the empty string (*""). Example:

```
find-next-substring-matching-pattern
("{A-Z}{a-z}*", "according to Will Rogers", 1)
-> "Will"
```

Replacing a Substring Using a Regular Expression

find-and-replace-pattern
(*search-pattern*: text, *text-to-substitute*: text, *source-text*: text,
start-position: integer, *end-position*: integer)
-> *modified-text*: text

Replaces each occurrence of *search-pattern*, a regular expression, in *source-text* between *start-position* and *end-position*, with *text-to-substitute*, a text string, and returns the complete *source-text* with the changes made. After each replacement, the search continues from the first character unaffected by the change. Example:

```
find-and-replace-pattern ("abcd", "newchars", "abcdcdef", 1, 8)
-> "newcharscdef"
```

If no substring between *start-position* and *end-position* matches *search-pattern*, the function returns *source-text* unchanged.

Parsing Strings into Tokens

A **token** is an atomic unit of a language, consisting of a syntax description and a type name. Every instance of a token is represented by a **lexeme**: a string whose sequence of characters conforms to the syntax characteristic of the token.

A **parser** is a utility that inputs a string and a set of token definitions, scans the string for lexemes, and outputs the token that each lexeme in the string represents. G2 provides two capabilities that can be used together to implement a parser:

- tokenizer: An item that contains regular expressions that define a set of tokens.
- get-next-token: A function that uses a tokenizer to locate lexemes in strings and return the token that each represents.

The difference between tokens and lexemes is analogous to the difference between numbers and character strings that represent numbers. Where the meaning is clear, strings that represent numbers are often referred to as if they were the numbers themselves. Similarly, lexemes that represent tokens are often referred to informally as if they were the tokens themselves.

Specifying the Syntax for Extracting Tokens

To parse a string into tokens, a parser must know:

- The syntax of every type of token in the language that it parses.
- What to do when it locates a token of each type.
- The syntax of anything it should ignore when searching for tokens,

G2 uses tokenizers to specify this information.

To create a tokenizer:

➔ Select KB Workspace > New Definition > tokenizer.

The class-specific attributes of a tokenizer are:

Attribute	Description
patterns-definition	One or more named regular expressions.
Allowable values:	Pairs of the form *name regular-expression*
Default value:	No value
tokens-definition	One or more regular expressions and action to take when each is encountered.
Allowable values:	Pairs of the form *regular-expression action*
Default value:	No value

Defining Patterns

A **pattern** is a named regular expression. Patterns allow you to:

- Use the same regular expression in multiple locations by giving its name.
- Create complex regular expressions by combining the names of simpler expressions.

A **pattern definition** is a pair of the form:

name regular-expression

where *name* is any symbol.

The patterns-definition attribute of a tokenizer specifies a set of patterns that are available in a tokenizer. The value of the attribute is zero or more consecutive pattern definitions. No patterns need be defined in a tokenizer; they are strictly a convenience.

G2 reads and compiles pattern definitions in sequential order. Once a pattern has been defined, it is available for use in subsequent pattern definitions. The syntax for referencing a pattern definition is:

"$(*name*)"

The following could be the value of a tokenizer's patterns-definition attribute:

```
nonzero     "{1-9}"
digit       "{0-9}"
numseq      "$(nonzero)$(digit)*
int         "{\+\-}?$(numseq)"
real        "($(int)\.$(digit)*)|({\+\-}?\.$(digit)+)"
name        "{A-Z}{a-z}*|{A-Z}\."
```

The rest of the examples in this section assume the preceding pattern definitions.

To specify a tokenizer's pattern definitions:

➔ Make the desired definitions the values of the tokenizer's patterns-definition attribute.

The scope of a pattern is the tokenizer that specifies it. Hence the same name can be used to represent different expressions in different tokenizers.

Defining Tokens

A **token definition** is a pair of the form:

regular-expression response

where *regular-expression* specifies the syntax of some class of token, and *response* tells what to do on encountering a token that matches *regular-expression*. The *regular-expression* can use any regular expression construct, including patterns referenced via $(*name*). The possible types of *response* are:

- The symbol do-nothing.
- A symbol other than do-nothing.

The meaning of each of these responses is described under Responding to a Match.

The tokens-definition attribute of a tokenizer specifies one or more token definitions. These define the tokens that the tokenizer is to scan for. The following could be the value of a tokens-definition attribute.

```
"$(numseq) $(name) Street"    address
"$(name) $(name)"             person
" "                           do-nothing
"$(int)                       zip-code
```

The rest of the examples in this section assume the preceding token definitions.

To specify a tokenizer's token definitions:

➔ Make the desired definitions the values of the tokenizer's tokens-definition attribute.

Locating Tokens in a String

Once you have specified a tokenizer's pattern definitions (if any) and token definitions, you can use the tokenizer to locate the tokens that it recognizes. G2 provides a text manipulation function for this purpose:

get-next-token
(*tokenizer*: class G2-tokenizer, *source-text*: text, *start-position*: integer)
-> structure (*token-type*: symbol, *start-index*: integer, *end-index*: integer)

Scans *source-text* beginning at *start-position* for tokens as defined in *tokenizer*. Returns a structure that gives the results of the search, as described in this section.

Text functions in general are described under G2 Text Manipulation Functions and G2 Conventions for Manipulating Text. G2 functions in general are described in Functions.

Searching for a Token

Get-next-token is similar to find-next-pattern, as described under Locating a Substring Using a Regular Expression, but is much more general. The action of get-next-token is as follows:

- Look for substrings that begin at *start-position* and match one or more of the token definitions in *tokenizer*.
- If no substring matches any definition, return:

  structure (token-type: FALSE, start-index: 0, end-index: 0)

- If substrings of different length match the same definition, use the longest such substring.
- If longest substrings of equal length match different definitions, use the substring whose definition appears earlier in the list of definitions.

- If a match is found, proceed as specified by the *response* associated with the definition that matched.

Responding to a Match

Every token definition specifies a response, as described under Defining Tokens. The possible types of response, and the meaning of each, are:

- The symbol do-nothing continues scanning from the first character after the end of the matching substring.
- A symbol other than do-nothing returns:

    structure (*token-type*: symbol, *start-index*: integer, *end-index*: integer)

where:

token-type: The symbol given by the response in the matching definition.

start-index: The character position where the matching token begins.

end-index: The character position where the matching token ends.

Example

If a tokenizer has the pattern and token definitions described earlier in this section, the call:

```
get-next-token (tokenizer, " 10461 Steve Street", 10)
```

returns:

```
structure (token-type: FALSE, start-index: 0, end-index: 0)
```

and the call:

```
get-next-token (tokenizer, " 10461 Steve Street", 1)
```

returns:

```
structure (token-type: address, start-index: 2, end-index: 19)
```

Note that start-index is 2 even though start-position was 1. This occurred because the tokenizer specifies that a blank (" ") has a response of do-nothing. Therefore get-next-token skipped over the initial blank at position 1 of " 10461 Steve Street", and continued scanning from position 2.

Extracting Tokens from a String

The get-next-token function does not return the lexeme that it found, because a token's type is usually all that is needed: returning the lexeme would be a needless overhead. In some cases, the lexeme is also needed.

To extract a token identified by get-next-token:

➔ Use get-from-text as described under Obtaining a Substring.

G2 Character Representation

G2 represents all symbol and text in a knowledge base using the Unicode Worldwide Character Standard, which provides support for multiple languages as described in G2 Character Support.

Working with Multiple Character Sets

While G2 supports the Unicode character set, numerous other character sets exist that are neither Unicode-based, nor Unicode compliant.

To coexist with diverse character sets, KBs require the ability to translate:

- Imported characters into Unicode.
- Exported characters to a given character set of a non-Unicode environment.

G2 provides the text-conversion-style class, described next, to perform character translation to and from Unicode, along with numerous text conversion functions, described in Character Set Conversion Functions.

Working with Text Conversion Styles

The text-conversion-style class lets you specify certain text conversion parameters to represent different character sets for importing or exporting text. For example, if your KB requires translation for three different character sets:

- Gensym
- Cyrillic
- Japanese

you could create three text-conversion-style items. Each of the text-conversion-style items would represent a particular character set that you required. For example, to facilitate the Gensym, Cyrillic, and Japanese character sets, you could create these three text conversion styles:

Use this text-conversion-style item...	For importing and exporting...
gensym-text-style	Gensym character set text
cyrillic-text-style	Cyrillic text
shift-jis-text-style	Japanese text

Once you create the text conversion styles your KB requires, any item that interacts with text conversion can use them, as described in Using a Custom Text Conversion Style.

To create a text-conversion-style item:

1 Select KB Workspace > New Definition > text-conversion-style.

2 Position the new item on the workspace.

3 Click on the item to display its menu.

4 Choose table.

Naming the Conversion Style

You must name each text-conversion-style item. Other items refer to text conversion styles by name.

Determining the External Character Set to Use

The external-character-set attribute lets you choose from the following character sets, where the symbol gensym is the default. The external character set determines how G2 encodes characters whenever the text conversion style is in use.

Note We recommend that you use the us-ascii character set for XML text. For more information, see Providing the XML Code as Text.

Character Set	Description
us-ascii	7-bit, single byte character set
latin-1	8-bit, single byte character set ISO-8859-1
latin-2	8-bit, single byte character set ISO-8859-2
latin-3	8-bit, single byte character set ISO-8859-3
latin-4	8-bit, single byte character set ISO-8859-4
latin-cyrillic	8-bit, single byte character set ISO-8859-5
latin-arabic	8-bit, single byte character set ISO-8859-6
latin-greek	8-bit, single byte character set ISO-8859-7
latin-hebrew	8-bit, single byte character set ISO-8859-8
latin-5	8-bit, single byte character set ISO-8859-9
latin-6	8-bit, single byte character set ISO-8859-10

Character Set	Description
jis	7-bit, JIS X 0208 (Japanese)
jis-euc	8-bit, JIS X 0208
shift-jis	Shift-jis encoded JIS X 0208 (Japanese)
ksc	7-bit, KS C 5601 (Korean)
ksc-euc	8-bit, KS C 5601
unicode	Unicode as series of-16 bit character codes.
unicode-byte-swapped	Unicode as series of-16 bit character codes, but as byte-swapped codes.
unicode-ucs-2	8-bit byte sequences of Unicode in UCS-2 format, most significant byte first.
unicode-ucs-2-byte-swapped	8-bit byte sequences of Unicode in UCS-2 format, least significant byte first.
unicode-utf-7	Standard 7-bit encoding of Unicode.
unicode-utf-8	Standard 8-bit encoding of Unicode.
gensym	The Gensym character set, as used in G2 and related products since Version 1.0, modified to handle Unicode.
x-compound-text	X compound text with subset of ISO 2022 escapes.

Using a Replacement Character

You can specify a replacement character to use in the event that Unicode does not have a character code for any imported character, or for any exported character that Unicode cannot represent. In the replacement-character attribute, specify a one-character string or character code. The default is none, which means that any unrepresented characters will be omitted.

Specifying the Han-Unification Mode

You can specify whether Japanese, Korean, or Chinese is preferred when translating Chinese characters into non-Unicode character sets such as gensym.

In the han-unification-mode attribute, choose:

- japanese
- chinese (traditional Chinese, simplified)
- korean

The default mode is japanese.

Specifying the External Line Separator

Line separators vary among different character sets. The external-line-separator attribute lets you specify what characters are used to indicate the end of one text line and the beginning of the next.

The external-line-separator choice is valid only when exporting text. When importing text, G2 separates lines of text whenever it sees any of the available options. An exception is for the Unicode line separator options, which G2 only searches for when the current text-conversion-style is using one of the Unicode character sets. Character set options are described in Determining the External Character Set to Use.

These are the six possible line separators:

Line Separator	Description
per-platform	This is the default value. With this value, G2 determines the current operating system and selects a line separator as follows: CRLF: Intel NT LF: UNIX If G2 cannot determine the operating system, or it is not one of those listed, the default option is LF.
CR	The carriage return character, which is ASCII 13 decimal and Unicode 000D hexadecimal.
LF	The linefeed character, which is ASCII 10 decimal and Unicode 000A hexadecimal.
CRLF	The two character carriage return and linefeed sequence.

Line Separator	Description
unicode-line-separator	Code 2028.
unicode-paragraph-separator	Code 2029.

While you can choose the line separator of your choice, not every option is applicable to every external character set. For example, the unicode-line-separator or the unicode-paragraph-separator cannot be expressed in ASCII.

Using a Custom Text Conversion Style

The text-conversion-style attribute appears in all items that interact with text conversion:

- Language Parameter system table
- g2-stream
- gfi-output-interface and gfi-input-interface

GFI is a superseded capability. For more information see Appendix F, Superseded Practices.

When at least one text conversion style exists in a KB, you can direct any one of the previous items to use that particular style by including its name in the text-conversion-style attribute.

The text-conversion-style attribute is value-writable and value-readable. This means you can get and set the text conversion style of language-parameters, or an instance of g2-stream, gfi-input-interface, or gfi-output-interface by exporting the attribute text-conversion-style, using the attribute access facility. For example, this example code changes the text-conversion-style of the local variable FS, which is a g2-stream, to MY-TEXT, which is a text-conversion-style:

```
conclude that the text-conversion-style of FS = the symbol MY-TEXT
```

For more information, see Attribute Access Facility.

Using the Default Text Conversion Style

If you do not provide your own text-conversion-style and an item requires one, G2 uses a system-defined text-conversion-style. The relevant attribute values of the system-defined class are as follows:

This attribute...	Has this value...
external-character-set	gensym
replacement-character	8-bit replacement char: none
han-unification-mode	japanese
external-line-separator	per platform

The system-defined text-conversion-style is generally designed to import and export text as it was done in G2 Version 4.0. For text whose external encoding was not specified in 4.0 such as Greek, Hebrew, Arabic, and Georgian, such a comparison is meaningless, but the definition of the Gensym character set clarifies the interpretation that should be assigned.

The reason japanese is used as the default han-unification-mode is that Han (Chinese) characters are infrequently used in Korean writing, but frequently used in Japanese writing.

Working with G2-Stream Items

Several text-oriented system procedures create g2-stream items as part of opening and closing files external to G2. You can specify a particular text-conversion-style for the g2-stream item to use as described in Using a Custom Text Conversion Style.

If the g2-stream contains characters that are not included in the default character set, you might need to programmatically change the text-conversion-style of the g2-stream to a character set that supports all the characters in the file. You should change the text-conversion-style after you open the file and before you begin reading it.

If you do not specify a particular text-conversion-style, G2 uses the system-defined style, which is defined in the text-conversion-style attribute of the Language Parameters system table. The default character set is Gensym.

Note Changing the text-conversion-style attribute while reading from a file, writing to a file, or setting file positions in a g2-stream is not recommended; if you do, the results are undefined.

Character Set Conversion Functions

These are the text processing functions to support character conversion between the standard Gensym character set and the Unicode character set.

Converting Character Codes to Unicode Text

character-codes-to-text
(*character-codes*: sequence)
-> *text-of-unicode-characters*: text

Accepts a sequence of Unicode character codes and returns the appropriate Unicode text that the codes represent.

The next example shows a procedure that accepts a sequence as its single argument, and displays the returned characters that the Unicode codes represent:

```
convert-codes(codes: sequence)
T: text;
begin
    T = character-codes-to-text(codes);
    post "The unicode codes: [codes] represent these characters: [T]
end
```

```
#171   4:09:15 p.m.   The unicode codes:
  sequence (97,
  98,
  99,
  100,
  101,
  102,
  103,
  104) represent these characters: abcdefgh
```

Converting Text to Unicode Character Codes

text-to-character-codes
(*input-text*: text)
-> *sequence-of-unicode-character-codes*: sequence

Accepts a series of text characters and returns the sequence of Unicode character codes that represent the characters of *input-text*.

For information on using square brackets to get the Unicode character code of a single character in a text, see Getting Unicode Character Codes.

The next example shows a procedure that accepts a text value as its single argument, and displays the returned sequence of Unicode character codes for the text:

```
convert-it (T: text)
codes: sequence
begin
     codes = text-to-character-codes(T);
     post "The text string [T] is represented by these unicode codes: [codes]"
end
```

```
#179   4:27:55 p.m.   The text string gensym is
  represented by these unicode codes:
  sequence (103,
  101,
  110,
  115,
  121,
  109)
```

Comparing Text

compare-text
    (*text-1*: text, *text-2*: text)
    -> *greater-than-or-equal-to*: integer

    Compares the *text-1* and *text-2* arguments and returns an integer greater than, equal to, or less than 0, depending on whether the numeric value of the Unicode code that represents the character of *text-1* is greater than, equal to, or less than *text-2*.

The next example shows a procedure that accepts two text values as its arguments, and displays the result of comparing the text:

```
comparing-text(text1: text, text2: text)
result: integer;
begin
     result = compare-text(text1, text2);
     post "The result of [text1] compared to [text2] is: [result]."
end
```

```
#182   4:35:00 p.m.   The result of gensym
  compared to GENSYM is: 1.
```

Exporting Unicode Text

export-text
(*unicode-text*: text, *conversion-style*: class text-conversion-style)
-> *export-text*: text

Takes a Unicode text and returns an equivalent text in an encoding and character set specified by *conversion-style*, and with various other parameters specified there.

Importing Unicode Text

import-text
(*external-text*: text, *conversion-style*: class text-conversion-style)
-> *unicode-text*: text

Takes a Unicode text in an encoding and character set specified by *conversion-style*, and with various other parameters specified there, and returns the corresponding Unicode string.

Determining Unicode Digits

is-digit
(*character-code*: integer)
-> *unicode-digit*: truth value

Returns true if *character-code* is not in the range 0x2000-0x2FFF and the Unicode name contains the word DIGIT.

This is true of both:

- The normal digits that have always been accepted as digits in G2, and which correspond to ASCII digit characters with codes 0x30-0x39.
- The decimal digit characters in various other scripts.

These are the ranges of Unicode characters that are considered digits:

0x0030 through 0x0039	ASCII/ISO-LATIN-1 digits ('0' through '9')
0x0660 through 0x0669	Arabic-Indic digits
0x06F0 through 0x06F9	Extended Arabic-Indic digits
0x0966 through 0x096F	Devanagari digits
0x09E6 through 0x09EF	Bengali digits
0x0A66 through 0x0A6F	Gurmukhi digits

0x0030 through 0x0039	ASCII/ISO-LATIN-1 digits ('0' through '9')
0x0AE6 through 0x0AEF	Gujarati digits
0x0B66 through 0x0B6F	Oriya digits
0x0BE7 through 0x0BEF	Tamil digits
0x0C66 through 0x0C6F	Telugu digits
0x0CE6 through 0x0CEF	Kannada digits
0x0D66 through 0x0D6F	Malayalam digits
0x0E50 through 0x0E59	Thai digits
0x0ED0 through 0x0ED9	Lao digits
0xFF10 through 0xFF19	Full width digits

Determining Lowercase Characters

is-lowercase
(*character-code*: integer)
-> *lowercase*: truth-value

Returns true if *character-code* is lowercase.

Determining Readable Digits

is-readable-digit
(*character-code*: integer)
-> *readable-as-decimal*: true

Returns true if *character-code* corresponds to a digit that can be read by the G2 reader to make a decimal number in G2. These are the character codes in the ASCII range for 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, and are the ASCII digit characters that range from 0x30 to 0x39.

Determining Readable Digits in Radix

is-readable-digit-in-radix
(*character-code*: integer, *radix*: integer)
-> *is-digit*: true

Returns true if character-code corresponds to a digit in the specified radix and if character code is the ASCII range. Radix must be between 2 and 36. Depending on *radix*, the character codes may be any of the ranges 0x30-0x39 (for digits 0-9), 0x41-0x5A (for A-Z), or 0x61-0x7A (a-z).

Determining Titlecase Characters

is-titlecase
(*character-code*: integer)
-> *titlecase*: truth-value

Returns true if *character-code* is in titlecase.

Titlecase is true of those few precomposed characters in Unicode that encode multiple Latin letters, and so appear to be a capital followed by small letter. For example, there is a character that looks like Lj named LATIN CAPITAL LETTER L WITH SMALL LETTER J.

Determining Uppercase Characters

is-uppercase
(*character-code*: integer)
-> *uppercase*: truth-value

Returns true if *character-code* is uppercase.

Obtaining a Readable Symbol from Text

readable-symbol-text
(*printed-text*: text)
-> *readable-text*: text

Returns a text value that is readable as a symbol version of *printed-text*. For example, if *printed-text* is:

SYMBOL WiTH SPACES++

this function returns the text:

symbol@ w@ith@ spaces@+@+

If this returned text is read into the G2 tokenizer, it would be read in as a single symbol consisting of the letters in the printable text. Notice that the space characters and the plus (+) signs require an escape character (@), as does the lowercase i character in the word WiTH. The escape characters insure that the spaces and plus (+) signs are accepted, and that the letter i is not made uppercase by tokenizing. Note also that this representation only guarantees that it will be readable as the symbol with the same printed representation.

Obtaining a Readable Text

readable-text
(*printable-text*: text)

Converts text input in its printable representation to a form that is readable internally by G2. For example, if *printable-text* is:

```
Use quotes around email addresses: "mhd@gensym.com"
```

This function converts the text to:

```
"Use quotes around email addresses: @"mhd@@gensym.com@"
```

Converting a Value into a Readable Representation

readable-text-for-value
(*value-for-text*: value)

Changes a value into that value's readable representation, such that reading back such text through the G2 tokenizer would result in the same value.

This function converts *value-for-text* only if the value has a readable representation. Value types with a readable representation are:

- Text
- Quantity
- Symbol
- Truth-value

If the value has no readable representation, G2 signals an error.

Converting Characters to Lowercase

to-lowercase
(*character-code*: integer)
-> *unicode-equivalent*: integer

Returns the character code that is equivalent to lowercase in Unicode. If there is no equivalent, returns *character-code*.

Converting Characters to Titlecase

to-titlecase
(*character-code*: integer)
-> *unicode-equivalent*: integer

Returns the character code that is equivalent to titlecase in Unicode. If there is no equivalent, returns *character-code*.

Converting Characters to Uppercase

to-uppercase
(*character-code*: integer)
-> *unicode-equivalent*: integer

Returns the character code that is equivalent to uppercase in Unicode. If there is no equivalent, returns *character-code*.

For example, the next procedure returns information about lowercase, uppercase, and titlecase of a single character code:

```
text-case(unicode-code: integer)
lowercase, uppercase, titlecase: integer;
begin
    lowercase = to-lowercase(unicode-code);
    uppercase = to-uppercase(unicode-code);
    titlecase = to-titlecase(unicode-code);
    post "The lower case code is [lowercase],
        the uppercase code is [uppercase],
        and the titlecase code is [titlecase]"
end
```

```
#185   4:41:27 p.m.   The lowercase code is 97,
  the uppercase code is 65, and the titlecase
  code is 65.
```

Transforming Text for Unicode Comparison

transform-text-for-unicode-comparison
(*text-to-transform*: text, *consider-case*: truth-value)

If *consider-case* is *false*, this is just the identity operation. Otherwise, this returns text with all characters changed to their uppercase equivalents, if any, and if not, just kept as themselves.

Note that:

```
(text-1 > text-2)
```

is equivalent to (is true if and only if it is the case that)

```
compare-text
    (transform-text-for-unicode-comparison (text-1, false),
     transform-text-for-unicode-comparison (text-1, false))
> 0
```

Transforming Text for G2 4.0 Comparison

transform-text-for-G2-4.0-comparison
(*text-to-transform*: text, *consider-case*: truth-value)

If *consider-case* is false, transforms text so that, for text that was possible to have in G2 4.0, it is compared exactly as it would have been in G2 4.0. This difference between version 4.0 and 5.0 and later is mostly in the handling of accented and special characters, and in the handling of Japanese. Latin characters that are in ASCII and the Korean and Russian characters allowed in G2 4.0 are sorted in an internally consistent manner in G2 5.0 and later. The sorting of text that mixes any of the above mentioned subsets of characters, as well as the linebreak character, is quite different in G2 5.0 and later as well. This transforming function makes any such distinctions go away.

Note that if in 4.0:

```
(text-1 > text-2)
```

is true, then in 5.0 and later:

```
g2-compare-text
     (g2-transform-text-for-G2-4.0-comparison (text-1, false),
      g2-transform-text-for-G2-4.0-comparison(text-1, false))
> 0
```

is true.

27

XML Parsing

Describes how to parse XML code and make callbacks to user-defined procedures.

Introduction

G2 provides a way to parse XML code and execute user-defined callbacks. G2 uses the SAX (Simple API for XML) standard, which provides an event-based XML parser.

The G2 XML parser consists of a parser class and a set of G2 system procedures that parse the XML text and execute the callbacks. You provide the XML code as a text string, whether from a file, from a bridge, or directly in G2. You then write G2 code that sends the text to the parser and sequentially executes each callback, based on the parsed text. The callback can be any user-defined procedure with the required signature.

You can associate callbacks with the start and end of the XML document, the start and end of an XML element, characters between XML elements, and comments. You can also associate callbacks with warnings, errors, and fatal errors that occur when parsing the XML code.

You can parse the entire XML text at once, or you can parse the XML text in "chunks." As G2 parses the text, it queues the specified callbacks for each event. Your procedure executes each callback in the queue sequentially by calling a system procedure.

Because G2 parses the text and queues the callbacks as a single process, no other processing can occur while the parsing takes place. Therefore, you should ensure that the text strings you are parsing are not too long.

You can also parse XML text directly from a file, which queues all events.

For the definition of the API procedures, see XML Parser API in the *G2 System Procedures Reference Manual*.

For an example, see `xml.kb` in the `g2\kbs\samples` directory of your G2 installation directory.

For information on SAX, visit `www.saxproject.org`.

For information on XML, visit `www.w3.org/TR/REC-xml`.

Providing the XML Code as Text

To use the XML parser, you provide the XML code as a G2 text. Depending on the source of the XML code, you might use one of several techniques:

- Use G2 system procedures to read the entire XML file into a text string or to read one line at a time. For information on reading text files, see File Operations in the *G2 System Procedures Reference Manual*.

  We recommend that you use the us-ascii character set for XML text. For more information, see Working with Text Conversion Styles.

- Use G2 Gateway to write a bridge that receives XML text.
- Write the XML code directly in G2.

G2 limits the size of text strings to 1,000,000 bytes; therefore, you should limit the size of your XML code accordingly. Note that while reading or transferring very large strings, G2 performs no other processing.

Rather than reading the file in chunks, you can also parse an entire XML file. Although this approach takes up more memory than reading the file in chunks, it avoids the limitation on the maximum size of a text and is easier than parsing the file in chunks. Note that this technique does not allow other processing while reading the file, which might be an issue for very large files or slow file systems.

SAX-Parser Class

A SAX parser is an instance of the sax-parser class. It parses XML text and executes user-defined callbacks. The SAX parser provides the following types of callbacks:

- Start and end document callbacks execute when the parser begins and ends parsing the XML text.
- Start and end element callbacks execute when the parser begins and ends parsing an XML element.
- Characters callbacks execute when the parser encounters characters between XML elements, which are not comments or malformed XML code.
- Comment callbacks execute when the parser encounters a comment in the XML code.
- Warning, error, and fatal-error callbacks execute when the parser generates a warning, error, or fatal error, based on malformed XML code.

For each callback, you provide a symbol that names a user-defined callback procedure with a given signature, which performs the desired action when the event occurs. You can provide any or all of the available callbacks.

For a description of the callback procedure, see SAX Callback Procedure.

The following table summarizes the class-specific attributes of the sax-parser class:

Attribute	Description
start-document-procedure	The user-defined callback procedure associated with the start of the XML text.
Allowable values:	The callback procedure name as a symbol
Default value:	none
end-document-procedure	The user-defined callback procedure associated with the end of the XML text. This callback procedure executes only after g2-sax-finish-parsing is called.
Allowable values:	The callback procedure name as a symbol
Default value:	none

Attribute	Description
start-element-procedure	The user-defined callback procedure associated with the start of an XML element.
Allowable values:	The callback procedure name as a symbol
Default value:	none
end-element-procedure	The user-defined callback procedure associated with the end of an XML element.
Allowable values:	The callback procedure name as a symbol
Default value:	none
characters-procedure	The user-defined callback procedure associated with characters that appear between XML elements.
Allowable values:	The callback procedure name as a symbol
Default value:	none
comment-procedure	The user-defined callback procedure associated with comments in the XML code.
Allowable values:	The callback procedure name as a symbol
Default value:	none
warning-procedure	The user-defined callback procedure associated with an XML warning.
Allowable values:	The callback procedure name as a symbol
Default value:	none

Attribute	Description
error-procedure	The user-defined callback procedure associated with an XML error.
Allowable values:	The callback procedure name as a symbol
Default value:	none
fatal-error-procedure	The user-defined callback procedure associated with a fatal XML error.
Allowable values:	The callback procedure name as a symbol
Default value:	none
number-of-pending-callbacks	The number of callbacks remaining to execute. This attribute is decremented each time a callback is executed by calling g2-sax-execute-next-callback. The value is read-only.
Allowable values:	integer
Default value:	0

For example, this code creates a sax-parser and associates user-defined callback procedures with the start and end of the XML document, the start and end of each XML element, and the characters between XML elements.

```
my-sax-parser: class sax-parser;

create a sax-parser my-sax-parser;
conclude that the start-document-procedure of my-sax-parser is
     my-start-document-callback;
conclude that the end-document-procedure of my-sax-parser is
     my-end-document-callback;
conclude that the start-element-procedure of my-sax-parser is
     my-start-element-callback;
conclude that the end-element-procedure of my-sax-parser is
     my-end-element-callback;
conclude that the characters-procedure of my-sax-parser is
     my-character-callback;
```

SAX Callback Procedure

The syntax for the callback procedure of a sax-parser is:

sax-parser-callback-procedure (*sax-parser*: class sax-parser, *data*: structure)

The syntax for the structure argument depends on the callback, as follows:

Callback Procedure	Structure
start-document-procedure	structure (callback-type: start-document)
end-document-procedure	structure (callback-type: end-document)
start-element-procedure	structure (callback-type: start-element element-name: *element-name* attributes: sequence (structure (attribute-name: *attribute-name* attribute-value: *attribute-value*), structure (attribute-name: *attribute-name* attribute-value: *attribute-value*) . . .))
end-element-procedure	structure (callback-type: end-element element-name: *element-name*)
characters-procedure	structure (callback-type: characters string: *character-string*)
comment-procedure	structure (callback-type: comment comment: *comment-string*)
warning-procedure	structure (callback-type: warning error-message: *warning-string*)

Callback Procedure	Structure
error-procedure	structure (callback-type: error error-message: *error-string*)
fatal-error-procedure	structure (callback-type: fatal-error error-message: *fatal-error-string*)

This table describes the attribute values of the structure:

Attribute Value	Description
start-document end-document start-element end-element characters comment warning error fatal-error	A symbol that identifies the type of callback.
element-name	The name of the XML element, as a string.
attribute-name	The name of the XML attribute, as a string.
attribute-value	The value of the XML attribute, as a string.
character-string	The value of the text string of non-XML characters.
string-length	The length of the text string of non-XML characters, as an integer.
comment-string	The value of the text string that appears as a comment in the XML code.
warning-string	The value of the text string generated as a warning when parsing the XML code.
error-string	The value of the text string generated as an error when parsing the XML code.
fatal-error-string	The value of the text string generated as a fatal error when parsing the XML code.

Example

The following procedure shows how to parse XML code in chunks of 128 characters. The XML code is stored in a free text named xml-text. The procedure executes start and end document, start and end element, characters, comment, and error callbacks. The SAX Parser API functions are shown in bold.

```
do-my-sax-parsing ()

my-sax-parser: class sax-parser;
text-input: text;
start-index, end-index, length, events-ready, i: integer;
my-preferred-chunk-size: integer = 128;

begin
    create a sax-parser my-sax-parser;
    conclude that the start-document-procedure of my-sax-parser is start-doc;
    conclude that the end-document-procedure of my-sax-parser is end-doc;
    conclude that the start-element-procedure of my-sax-parser is start-elmnt;
    conclude that the end-element-procedure of my-sax-parser is end-elmnt;
    conclude that the characters-procedure of my-sax-parser is sax-chars;
    conclude that the comment-procedure of my-sax-parser is sax-comment;
    conclude that the error-procedure of my-sax-parser is sax-error;

    text-input = the text of xml-text;

    start-index = 1;
    end-index = my-preferred-chunk-size;
    length = length-of-text(text-input);

    if (end-index > length) then end-index = length;

    repeat
        call g2-sax-parse-chunk(my-sax-parser, text-input, start-index,
            end-index);
        exit if end-index = length;
        start-index = end-index + 1;
        end-index = end-index + my-preferred-chunk-size;
        if (end-index > length) then end-index = length;
    end;
    call g2-sax-finish-parsing(my-sax-parser);

    repeat
        exit if the number-of-pending-callbacks of my-sax-parser = 0;
        call g2-sax-execute-next-callback(my-sax-parser)
    end;
    delete my-sax-parser

end
```

28

Functions

Lists system-defined functions and describes how to create new functions.

Introduction

A **function** is a named sequence of operations that compute and return a value. Functions and procedures are similar in many ways, but G2 invokes them differently, and they have different advantages and disadvantages.

G2 provides three kinds of functions:

- User-defined functions
- Tabular functions of one argument
- System-defined functions

You can also invoke functions that run outside G2, as described in Chapter 68, Foreign Functions on page 2009.

Invoking Functions

G2 executes a function when the function's name and arguments (if any) appear as part of an expression that G2 is evaluating. The function executes synchronously, and the value that it returns is used just as if it had appeared literally at the point of the invocation. A function reference can appear in any statement at any point where a literal value of the type returned by the function could appear.

By contrast, a procedure executes only when a reference to it appears in a call statement or a start action; executes synchronously when called and asynchronously when started; and may or may not return a value.

For information on procedures, see Chapter 22, Procedures on page 865.

Executing Functions

A function executes exactly as if the body of the function appeared literally at the point of invocation. Therefore:

- Function invocation is always synchronous and cannot cause a wait state.
- Function execution time is tallied to the cumulative time of the invoking context.
- Functions do not time out independently of their context.

For information on execution time and timeouts, see Limiting Procedure Execution Time on page 881 and Setting the Timeout Interval for a Rule on page 996.

User-Defined Functions

You create a **user-defined function** by using a function definition.

To create a user-defined function:

➔ Select KB Workspace > New Definition > function-definition.

G2 invokes the Text Editor automatically so that you can enter your function. The syntax is:

function-name ([*argument*] [,...]) = (*expression*)

where:

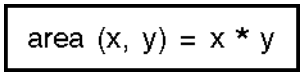
function-name	Specifies the name of the function.
argument [,...]	Describes the arguments for the function, and are substituted for the arguments in the function definition.
expression	Defines the computation of the function using a symbolic, arithmetic, or logical expression, and can include other user-defined or system-defined functions.

When you close the text editor, G2 attaches a rectangle representing the new function definition to the mouse. Click on any workspace to place the definition on that workspace. G2 transfers the definition rectangle to the workspace at the point of the click, and displays the text of the function in the rectangle. You can click the definition to edit it, display its table, transfer it to a different workspace, or delete it.

For example, the following specifies the area function:

```
area (x, y) = x * y
```

On a workspace, the area function definition looks like this:



After creating this function, you can refer to it by name or arguments in procedures or expressions. An example of using the area function is:

```
if area (the length of floor-1, the width of floor-1)  > 25 then
  inform the operator that
  "The area of floor-1 is greater than 25 square feet."
```

You can reference any system-defined or user-defined function in a function definition. You can also use functions recursively; however, be aware that you run the risk of exceeding the recursion limit specified in the Inference Engine Parameters system table or the memory allocated to the stack for recursive functions. In general, we recommend that you use procedures for recursive computation, rather than functions, because procedures are not handled using a stack and as such are not subject to the same memory limitations. Also, the error handling of procedures is more robust.

Tabular Functions of One Argument

Tabular functions of one argument are items of the tabular-function-of-1-arg class, referred to as **tabular functions**.

Tabular functions begin with a single-argument, user-defined function of the form:

f (x)

From such a function, you enter one or more arguments and values to derive, or cause G2 to compute, multiple comparative or associative values. G2 presents such comparisons and associations in a tabular format. Tabular functions and their arguments can consist of any type (integer, float, symbolic, logical, or text).

If you create a tabular function of arithmetic values, you can direct G2 to interpolate a return value for an argument that is not given in the table. G2 performs a straight-line interpolation whenever it interpolates one or more missing values. If you do not direct G2 to interpolate values, and a value of x is not given in the table, function evaluation fails.

Two examples illustrating different uses of tabular functions are:

Table of values

add or delete rows

x	the-color-of (x)
1	black
2	purple
3	blue
4	green
5	yellow
6	orange

Table of values

add or delete rows

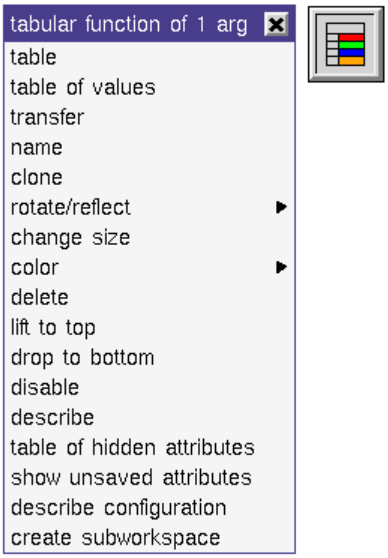
x	percent-of-max-flow (x)
10	30.0
20	60.0
30	75.0
40	80.0
50	87.0

This example...	Shows how to...
1	Translate values from one type into another. After specifying the series of numbers with an associated color, you could then use this tabular function in an expression such as: the-color-of (3) = blue
2	Define the percentage of maximum flow through a valve as a function of the percent of the valve opening. Thus, given a valve opening, x, you can use the function percent-of-max-flow to calculate the percent of maximum flow through a valve. You can use this function in an expressions such as: if the percent-of-max-flow (the percent-of-max-valve-opening of valve-1) > 80 and the outflow of valve-1 < 20 gpm then conclude that valve-1 is broken Percent-of-max-flow is defined as a tabular function, rather than as a user-defined function, because the relationship between the two variables is non-linear and cannot be expressed readily in an algebraically.

To create a new tabular function of one argument:

➔ Select KB Workspace > New Definition > tabular-function-of-1-arg.

The tabular function's icon and its menu are:



Notice that a tabular function has two table menu choices:

- table, which displays the attribute table.
- table of values, which lets you add and delete rows and display the table contents once they exist.

These are the attributes of a tabular function:

Attribute	Description
names	The name of the function as you will use it in expressions.
Allowable values:	*symbol*
Default value:	none
keep-sorted	Specifies whether G2 sorts the values in the table.
Allowable values:	{no | by args | by values}
Default value:	by args

Attribute	Description
interpolate?	Defines whether G2 will interpolate values that are not provided in the table.
Allowable values:	{no | yes}
Default value:	yes

Naming the Tabular Function

The names attribute provides the name to use when referring to the function within expressions. After completing this attribute, the name replaces the f(x) label in the table of values for the function.

Sorting the Items in the Table

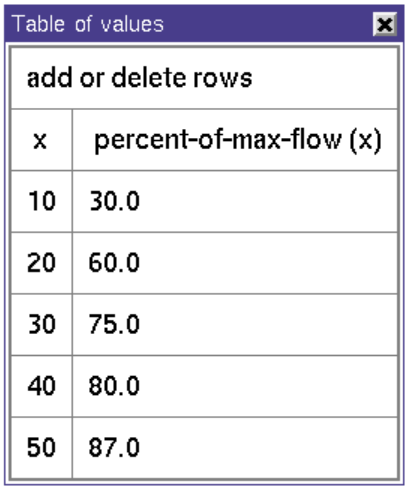
The keep-sorted attribute specifies whether G2 sorts the function arguments, referred to as the values of x, or the values that correspond to the arguments, referred to as f (x).

If G2 is interpolating values, you must specify by values for this attribute.

Interpolating Function Values

The interpolate? attribute specifies whether G2 interpolates a value for f (x) when the value of x is not explicitly given in the table.

When G2 is interpolating values, it performs a straight-line interpolation. For example, assume that the percent-of-max-flow function has these values:



G2 interpolates for all values of *x* in the range of 10 to 50. For example, if *x* is 15, G2 interpolates the percent-of-max-flow as 45.0. For interpolation to occur, the value of *x* must be greater than or equal to the smallest argument in the table and less than or equal to the largest argument in the table. If the argument is not in this range, G2 fails to evaluate the function.

Adding and Deleting Values and Arguments

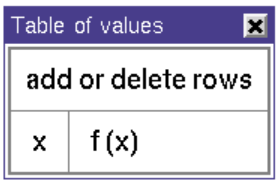
For each tabular function, you enter one or more arguments, and a corresponding number of return values. G2 can complete a list of multiple arguments and return values, after you supply the first two entries.

During function execution, each argument is a possible value to pass to the function; every value is the complement of an argument that G2 can return. When using the optional interpolation capabilities, a tabular function can accept arguments and return complementary values that are not listed in the table.

To show the tabular function values table:

➔ Choose table of values from the menu for the function's icon.

A new table of values contains the values of x and f(x):



To record values in the tabular function table:

1 Choose add or delete rows from the table of values.

2 Enter the syntax to add or delete rows in the edit workspace.

Adding Rows of Values

Use this syntax to add values to a tabular function:

```
add integer [unfinished] row[s]
  [ {at the {beginning | end} } | { {before | after} x = datum} ]
   with x = [datum1] [datum2] [,...] | etc.}
  and f (x) = [ datum] [datum2] [,...] | etc.} ] ]
```

Element	Description
integer	A positive integer specifying the number of rows to add.
at the beginning	Adds one or more values to the beginning of the table.
at the end	Adds one or more values to the end of the table.
before x = *datum*	Adds one or more values immediately before the row defined by *x* = *datum*.
after x = *datum*	Adds one or more values immediately after the row defined by *x* = *datum*.
with x = {*datum1* [*datum2*] [,...] | etc.}	Provides a list of possible argument values that you can pass to the tabular function. If the difference between successive values is constant, you can indicate a series of values for *x* by entering the first two values in the series, followed by etc. G2 computes the remaining values for you.
and f (x) = {*datum1* [*datum2*] [,...] | etc.}	Provides a list of the values that correspond to each of the arguments you provide. If the difference between successive values is constant, you can indicate a series of values for f(*x*) by typing the first two values in the series, followed by etc. G2 computes the remaining values for you.

As an example, entering this statement results in the tabular function that follows:

```
add 7 rows with x = 0, 10, etc. and f(x) = 0, 30, 60, 75, 80, 87, 91
```

Table of values

add or delete rows	
x	f (x)
0	0
10	30
20	60
30	75
40	80
50	87
60	91

Note that if you have indicated that G2 should sort by either argument or value, where you add the rows does not matter; G2 immediately sorts them. For example, the table shown above is sorted by argument. If you add a row to it with x = 55, G2 automatically places that row after the row that has 50 as an argument value.

Deleting Rows of Values

Use this syntax for deleting rows:

```
delete { integer | unfinished} row[s]
  [ {at the {beginning | end} } | { {before | after} x = datum} ]
  [with x =  [datum-1] [datum-2] [,...] | etc. ]
  and f (x) = [datum-1] [datum-2] [,...] | etc. ]
```

Element	Description
integer	A positive integer specifying the number of rows to delete.
unfinished	Specifies any row in which the value of x, or the value of f (*x*) has no value.
at the beginning	Deletes one or more values from the beginning of the table.

Element	Description
at the end	Deletes one or more values at the end of the table.
before x = *datum*	Deletes one or more values immediately before the row defined by *x* = *datum*.
after x = *datum*	Deletes one or more values after the row defined by *x* = *datum*.
with x = [*datum-1*] [*datum-2*] [,...] | etc.]	Provides a list of the argument values to delete. If the difference between successive values is constant, you can indicate a series of values for *x* by typing the first two values in the series, followed by etc. G2 computes the remaining values for you.
and f (x) = [*datum-1*] [*datum-2*] [,...] | etc.]	Provides a list of the return values to delete. If the difference between successive values is constant, you can indicate a series of values for f(x) by typing the first two values in the series, followed by etc. G2 computes the remaining values for you.

Some examples are:

Example	Result
delete unfinished rows	Deletes rows that do not have a value for either x or f (x).
delete row	Deletes the first row in the table.
delete row after x = 60	Deletes the row after the one that has 60 as the value for the x argument.
delete rows with x = 30 and f (x) = 40	Deletes all rows that have the value 30 for the x argument and 40 for the f(x) argument.
delete 5 rows at the end	Deletes the last five rows in the value table.

Changing Tabular Functions Programmatically

Previous G2 releases did not provide programmatic access to the x and f(x) fields of a tabular-function-of-1-arg item, referred to here as a tabular function.

Tabular functions include a hidden attribute, values-for-table-of-values, consisting of a sequence of structures as follows:

```
text-readable = false, text-writable = false,
value-readable = true, value-writable = true,
is-system-defined = true, defining-class = tabular-function-of-1-arg

type =
  sequence
  .  (structure
  .  .  (x: none | symbol | integer | float | text,
  .  .   f-of-x: none | symbol | integer | float | text)
  .   [, ...])
```

Using this hidden attribute, you can now conclude values directly into a tabular function.

To change a tabular function programmatically:

➔ conclude that the values-for-table-of-values of tab1 =
sequence (structure (x:2, f-of-x: 12), structure (x:4, f-of-x: 24))

This example produces these results:

TAB1

Table of values	
add or delete rows	
x	tab1 (x)
2	12
4	24

System-Defined Functions

G2 provides the following categories of system-defined functions:

- Arithmetic functions
- Attribute access functions (see Chapter 12, Attribute Access Facility on page 479).
- Bitwise functions

- Call function
- Character manipulation (see Chapter 49, G2 Character Support on page 1739).
- Connection functions (see Chapter 18, Connections on page 703).
- Format numeric text function
- Great circle distance function
- Quantity function
- Symbol function
- Rgb-symbol function
- Text functions (see Chapter 26, Text Parsing and Manipulation on page 1011).
- Time functions

Functions relating to a topic in its own chapter in this manual are described in that chapter, as indicated in the above list. All other functions are described in the following sections.

Arithmetic Functions

These are the system-defined arithmetic functions:

Function	Description
abs (*quantity-expression*) = (*quantity*)	Returns the absolute value of *quantity-expression*. An example is: abs(-9) = 9
arctan (*quantity-expression* [, *quantity-expression*]) = (*float*)	Using one argument: returns the arctangent of *quantity-expression* in radians. An example is: arctan (1) = 0.785 Using two arguments: returns the arctangent of *quantity-expression/quantity-expression* in radians. Use this function to handle cases where the result approaches infinity. An example is: arctan (1, 0) = 1.571 If both arguments are zero, G2 signals an error.
average (*quantity-expression*, *quantity-expression* [, *quantity-expression*...]) = (*quantity*)	Returns the average value for a list of two or more *quantity-expressions*, and no value for one *quantity-expression*. An example is: average (1,2,6) = 3

Function	Description
ceiling (*quantity-expression*) = (*integer*)	Returns the smallest integer value greater than or equal to the value of *quantity-expression*. See also the truncate and floor functions. Some examples are: ceiling (5.3) = 6 ceiling (-5.3) = -5 ceiling (5) = 5
cos (*quantity-expression*) = (*float*)	Returns the cosine of *quantity-expression* in radians. An example is: cos (1) = 0.54
exp (*quantity-expression*) = (*float*)	Returns e to the power of *quantity-expression*. An example is: exp (8) = e8
expt (*quantity-expression*, *quantity-expression*) = (*quantity*)	Returns the value of *quantity-expression* raised to the power of the second *quantity-expression*. An example is: expt(2, 4) = 16
floor (*quantity-expression*) = (*integer*)	Returns the largest integer that is less than or equal to *quantity-expression*. See also the truncate and ceiling functions. An example is: floor (5.3) = 5 floor (-5.3) = -6
ln (*quantity-expression*) = (*float*)	Returns the natural logarithm (base e) of *quantity-expression*. An example is: ln (1.5) = 0.405
log (*quantity-expression*) = (*float*)	Returns the base 10 logarithm of *quantity-expression*. An example is: log (1.5) = 0.176
max (*quantity-expression*, *quantity-expression* [,...]) = (*quantity*)	Returns the maximum value in a list of two or more *quantity-expressions*. It returns no value for just one *quantity-expression*. An example is: max (1,3,7,4) = 7

Function	Description
min (*quantity-expression*, *quantity-expression* [,...]) = (*quantity*)	Returns the minimum value in a list of two or more *quantity-expressions*. It returns no value for just one *quantity-expression*. An example is: min (3,7,2,5) = 2
quotient (*quantity-expression, quantity-expression*) = (*quantity*)	Returns the result of dividing the first *quantity-expression* by the second, truncated. The sign of the result depends on the arguments. Some examples are: quotient (5, 2) = 2 quotient (5, -2.1) = -2 quotient (-5, 2.1) = -2 quotient (5, 0) = 0 See the note at the end of this table for the results of dividing by zero using float arguments.

Function	Description
random (*quantity-expression*, [, *quantity-expression*]) = (*quantity*)	**Using one argument**: Returns a pseudo random number greater than or equal to zero and less than *quantity-expression*. If the argument is an integer, the function returns an integer (likewise for floating point numbers). The argument must be positive, and the return value is always positive. For example, random (3) can return 0, 1, or 2. **Using two integer arguments**: Returns a value greater than or equal to the first *quantity-expression*, and less than or equal to the second *quantity-expression*. If the first argument is greater than or equal to the second argument, G2 signals an error. **Using two float arguments**: Returns a value greater than or equal to the first *quantity-expression*, but less than the second. Arguments can be negative but the second *quantity-expression* must be greater than the first. For example, random (-1,1) can return either -1, 0, or 1, while random (-1.0,1) can return -1. 0, 0.1, or 0.999. G2 selects random numbers in a system-dependent manner. If the first argument is greater than or equal to the second argument, G2 signals an error.
remainder (*quantity-expression, quantity-expression*) = (*quantity*)	Returns the remainder that results from dividing the first *quantity-expression* by the second. The remainder always has the same sign as the dividend. Note that G2 provides this function in place of a mod function. Example are: remainder (5, 2) = 1 remainder (5.3, 2.0) = 1.3 remainder (5, 0) = 5 See the note at the end of this table for the results of dividing by zero using float arguments.

Function	Description
round (*quantity-expression*) = (*integer*)	Returns the nearest integer value for a floating point *quantity-expression*. Some examples are: round (2.4) = 2 round (-2.4) = -2 round (2.5) = 3
sin (*quantity-expression*) = (*float*)	Returns the sine of *quantity-expression*, in radians.
sqrt (*quantity-expression*) = (*float*)	Returns the square root of *quantity-expression*. It returns no value if *quantity-expression* has a negative value; instead, G2 signals an error. An example is: sqrt (5) = 2.236
tan (*quantity-expression*) = (*float*)	Returns the tangent of *quantity-expression*, in radians.

Function	Description
truncate (*quantity-expression*) = (*integer*)	Returns the truncated form of the decimal portion of the value of *quantity-expression*. This function always truncates toward zero. Some examples are: truncate(6.6) = 6 truncate(-5.3) = -5
truth-value (*quantity-expression*) = (*truth-value*)	Converts the value of *expression* to a fuzzy truth value. If *quantity-expression* = *1*.0, this function returns true. If *quantity-expression* = -1.0, this function returns false. If *quantity-expression* is such that -1.0 x 1.0, this function returns x true. If *quantity-expression* is a logical argument, the value returned is the same as the argument. If *quantity-expression* is a symbol or text, G2 returns no value. Instead, G2 signals an error. Some examples are: truth-value (6) = true truth-value (.6) = .600 true truth-value (-1) = false

Note When dividing by zero using the quotient and remainder functions with float arguments, the values G2 returns remain dependent on the behavior of the operating system you are using. Your operating system may return a `NaN` (not a number) or it may generate a floating point exception (`SIGFPE`).

If G2 receives a `NaN` from the operating system, it becomes the return value in G2. There are several types of `NaNs`, including positive and negative infinity (`+Inf` and `-Inf`). If a `SIGFPE` is generated within G2's evaluator, it causes a G2 stack error; otherwise the `SIGFPE` results in a G2 abort.

This floating-point behavior may be configurable on your platform. See your operating-system documentation for information.

Vector Functions

There are three system-defined vector (1-dimension array) functions, which are faster alternative for existing system procedure on arrays.

Function	Description

vector-add (*quantity-array, quantity-array, quantity-array*) = (*quantity-array*)	A faster alternative of g2-array-add, it adds the first and second *quantity-array* and put the results into the third *quantity-array*, which is also the return value (by reference). An example is:
	`array3 = vector-add(array1, array2, array3);`
	which equals to:
	`call g2-array-add(array1, array2, array3);`
	If array sizes are not equal, G2 signals an error.

vector-multiply (*quantity-array, quantity-array*) = (*quantity*)	A faster alternative of g2-array-multiply, it multiply the two *quantity-array* as parameter, and return a *quantity*. An example is:
	`result = vector-multiply(array1, array2);`
	which equals to:
	`result = call g2-array-multiply(array1, array2);`
	If array sizes are not equal, G2 signals an error.

vector-scalar-multiply (*quantity-array*, *quantity-expression, quantity-array*) = (*quantity-array*)	A faster alternative of g2-scalar-multiply, which multiplies a *quantity-array* by a scalar value *quantity* and returns the result to a second *quantity-array*, which is also the return value (by reference). An example is:
	`array2 = vector-scalar-multiply(array1, x, array2);`
	which equals to:
	`call g2-scalar-multiply(array1, x, array2);`
	If array sizes are not equal, G2 signals an error.

Attribute Access Functions

See Chapter 12, Attribute Access Facility on page 479 for information about attribute access and its associated functions.

Bitwise Functions

These are the system-defined bitwise functions:

Function	Description
bitwise-or (*value-expression*, *value-expression*) = (*integer*)	Performs an inclusive or operation on two values and returns an integer.
bitwise-and (*value-expression*, *value-expression*) = (*integer*)	Performs a logical and operation on two values and returns an integer.
bitwise-xor (*value-expression*, *value-expression*) = (*integer*)	Performs an exclusive or operation on two values and returns an integer.
bitwise-not (*value-expression*) = (*integer*)	Performs a logical not operation on a value and returns an integer.
bitwise-right-shift (*value-expression*, *value-expression*) = (*quantity*)	Returns the result of shifting the first *value-expression* n places to the right. The second *value-expression* specifies the number of bit positions to shift.
bitwise-left-shift (*value-expression*, *value-expression*) = (*integer*)	Returns the result of shifting the first *value-expression* n places to the left. The second *value-expression* specifies the number of bit positions to shift.
bitwise-test (*value-expression*, *value-expression*) = (*truth-value*)	Returns true or false after testing *value-expression* to see if the nth bit, specified by the second *value-expression*, is set.
bitwise-set (*value-expression*, *value-expression*) = (*quantity*)	Returns the result of setting the nth bit of the first *value-expression*, specified by the second *value-expression*.

Using Bitwise Operator Functions

Some examples of using the bitwise functions are:

```
new-value = bitwise-not (1918);
shift-bits = bitwise-right-shift (temp-int-var, 1)
inclusive-or = bitwise-or (quant1-param, quant2-param)
```

You can use the bitwise functions in any compiled attribute. For a definition of compiled attributes within G2.

For instance, the following example shows a readout table containing an expression with the logical-operation bitwise-or function. The two arguments of the bitwise-or function reference a quantitative parameter and an integer variable.

1584, valid indefinitely

INTEGER-VARIABLE-1

16

QUANTITATIVE-PARAMETER-1

bitwise-or (integer-variable-1, quantitative-parameter-1)	1584

a readout-table

Notes	OK
Item configuration	none
Names	none
Tracing and breakpoints	default
Expression to display	bitwise-or (integer-variable-1, quantitative-parameter-1)
Label to display	
Display format	default
Display update interval	5 seconds
Display wait interval	2 seconds
Display update priority	2
Show simulated values	no
Readout table display value	1584

Call-Function Function

The call function calls a specified function with its arguments. It lets you call a user-defined function indirectly when you know the function arguments.

The syntax is:

call-function (*function-definition-expression* [, *argument*] [,....])

An example is:

```
result = call-function (the function-definition that is the-measurement-for TANK-1, the volume of TANK-1, the level of TANK-1)
```

Note that the-measurement-for is a relation. You cannot use the call function to directly call system functions, which are G2 functions that do not have function-definition items. You could, however, define a function to call system functions, and then use call function to call that function. Within a normal function call, such as:

```
2 + combine (x, y)
```

if combine is a local name, G2 calls the function definition within that local name.

Character Manipulation Functions

See Chapter 49, G2 Character Support on page 1739 for information about character manipulation and its associated functions.

Connection Functions

See Chapter 18, Connections on page 703 for information about connections and their associated functions.

Format-Numeric-Text Function

Use this function to format numeric text:

format-numeric-text
(*quantity-text*: text, *formatting-expression*: text)
–> *formatted-text*: text

Formats numeric text, using a formatting expression such as dd.ddd. The *quantity-argument* is a text whose left-most part is a quantity. Leading spaces and any text following the quantity are both allowed. The *formatting-text* is a text containing a dd.ddd format. The result is a text that is the result of applying the formatting expression to the numeric text. For example:

```
format-numeric-text ("35.3", "dd.ddd") –> "35.300"
```

To assist with formatting variable values, which can include asterisks when the variable value has expired or when the variable has no value, the first argument can also be one of the following:

- The text value "****", which returns "****", regardless of the formatting expression, which you still need to provide.
- A text with a quantity and a trailing asterisk (*). The result is a text that has the number formatted according to the dd.ddd format expression, with a trailing asterisk. For example:

```
format-numeric-text ("35.3*", "dd.ddd") –> "35.300*"
```

This function exists to handle values of the format stored in the last-recorded-value-text hidden attribute of variables and parameters. For more information, see Obtaining the Last Recorded Value on page 619.

Great-Circle-Distance Function

great-circle-distance
(*latitude-1*: quantity, *longitude-1*: quantity,
latitude-2: quantity, *longitude-2*: quantity, *radius*: quantity)
–> *distance*: float

The great-circle-distance (gcd) is the distance between two points on a sphere when traveling on the surface of the sphere. The first point is at (*latitude-1*, *longitude-1*), and the second point is at (*latitude-2*, *longitude-2*). The latitudes and longitudes are given in degrees as quantities, usually floats. The answer is multiplied by the spherical *radius*, the fifth argument.

```
gcd = 2 * radius * arcsin (sqrt (sin ((lat1 - lat2) / 2) ^ 2 +
    cos (lat1) * cos (lat2) * sin ((long1 - long2) / 2) ^ 2))
```

arcsin(a) for -1 <= a <= 1

can be given in terms of arctan by:

arcsin(a) = arctan(a/sqrt(1-a^2))

where:

arcsin(1) = pi/2, arcsin(-1) = -pi/2

Latitudes go from 0 to 90 degrees N (positive) and from 0 to 90 degrees S (negative).

Longitudes go from 0 to 180 degrees E (positive) and from 0 to 180 degrees W (negative). 0 degrees is in Greenwich, England (prime meridian). 180 degrees is at the International Date Line. (The convention E negative and W positive is also fine as long as one is consistent.)

To convert from degrees to radians, multiply by pi/180.0, where pi = 3.141592653589793 (approx.).

So, 180 degrees = pi radians;

90 degrees = pi/2 radians = 1.5707963267948966 (approx.)

To convert from radians to degrees, multiply by 180/pi.

The units of the supplied radius determines the units of the result. For the Earth, equatorial radius = 6378.137 kilometers, polar radius = 6356.752 km, and FAI radius = 6371.0 km. radius = 1 implies that the result is in radians.

For example:

The distance in radians between LAX at (33.95, -118.4) degrees and JFK at (40.633333, -73.783333) degrees = 0.623585 radians.

great-circle-distance(33.95, -118.4, 40.633333, -73.783333, 1) = 0.623585 .

Using the FAI radius of 6371.0 km:

great-circle-distance(33.95, -118.4, 40.633333, -73.783333, 6371.0) = 3972.858 (km)

Quantity Function

The quantity function is used to obtain a quantity value from an argument that is either a logical expression or text, as:

Function	Description
quantity (*truth-value-expression*) = (*quantity*)	Returns a quantity value for a *truth-value-expression*. Some examples are: quantity (true) = 1.0 quantity (false) = -1.0 quantity (fuzzy truth value) = a real number between -1.0 and +1.0 exclusive
quantity (*text-expression*) = (*quantity*)	Extracts any numeric characters found at the beginning of *text-expression* and returns them as quantity values. Leading spaces are ignored. Numbers may begin with a plus sign, a minus sign, or may be in exponential format (contain an e preceding an exponent of ten, by which to multiply the preceding mantissa). If no numeric characters are found at the beginning of *text-expression*, G2 signals an error. This function does not evaluate an arithmetic expression that may be embedded in the text value (the string enclosed in quotation marks). If you specify initialize a quantity variable with a decimal point without trailing digits, the variable initializes to a float, not an integer. For example: x: quantity = 5. initializes x to 5.0, not 5. Some examples are: quantity ("123yes no") = 123 quantity ("-12.3abc") = -12.3 quantity ("5.") = 5.0 quantity ("exp(2)") = error quantity ("abc 123") = error

Symbol Function

The symbol function converts a text expression into a symbol. You can use the result of the symbol function to name transient items. The syntax is:

symbol (*text-expression*) = (*s*)

Caution Once G2 creates a symbol with the symbol function, the memory it uses cannot be reclaimed or reused within that execution of G2 even if you delete the item that uses the symbol. If a symbol with the name specified by the *text-expression* already exists, G2 reuses that symbol and does not create a new one.

G2 maps all alphabetic characters in a symbol to uppercase, even when such a character has been escaped. Thus, symbol ("abc") creates the same symbol as symbol ("AbC") and symbol ("@abc").

The symbol function extracts any alphanumeric characters including hyphens, periods, underscores, and single-quotes found at the beginning of *text-expression* and returns them as a symbol. These must not be numeric characters constituting a quantity. Leading and following spaces and tabs are ignored.

For example:

```
symbol("abc 123") = abc
symbol("abc-123") = abc-123
symbol("abc+123") = abc
```

Text-to-Symbol Function

The text-to-symbol function converts a text expression into a symbol:

text-to-symbol (*text-expression*) = (*s*)

It differs from the symbol function in two ways:

- It is up to an order of magnitude faster.
- It treats every character as escaped.

For example:

```
text-to-symbol ("Gensym")
```

escapes all characters, preserving their case and creating the symbol: Gensym;

while:

```
symbol ("Gensym")
```

uppercases all characters, creating the symbol: GENSYM.

Creating the symbol Gensym with the symbol function would require escaping every lowercase letter to prevent conversion to uppercase:

```
symbol ("G@@e@@n@@s@@y@@m");
```

To return a standard (all uppercase) G2 symbol with text-to-symbol, enter the argument in uppercase, as in:

```
text-to-symbol ("GENSYM");
```

Also, notice that symbol ("ITEM::TEST") would return the symbol "ITEM::TEST", and text-to-symbol ("ITEM::TEST") would return the symbol "ITEM@:@:TEST", hence they're different.

You can also use text-to-symbol to compose existing symbols into new ones while preserving the case of the symbols. Thus:

```
s1 = text-to-symbol ("aBc");
s2 = symbol ("d@@ef");
s3 = text-to-symbol ("[s1]-[s2]");
s4 = symbol ("[s1]-[s2]");
```

sets the values as follows:

```
s1: aBc
s2: DeF
s3: aBc-DeF
s4: ABC-DEF
```

Rgb-Symbol Function

You can provide a symbol of the form RGB*rrggbb* as a valid color name, where *rr*, *gg*, *bb*, are the 8-bit hex values for red, green, and blue. The full 24-bit color is used for drawing if the window is capable of it; otherwise, the closest Gensym standard color is used.

You use the rgb-symbol function to convert RGB color values to a symbol. The syntax is:

```
rgb-symbol
    (rr, gg, bb)
    –> rbg-symbol: symbol
```

Here are some examples:

NavyBlue: rgb-symbol(0,0,128) -> rgb000080

blue: rgb-symbol(0,0,255) -> rgb0000ff

brown: rgb-symbol(165,42,42) -> rgba52a2a

RoyalBlue: rgb-symbol(65,105,225) -> rgb4169e1

snow: rgb-symbol(255,250,250) -> rgbfffafa

Many examples of hexadecimal colors are available on the World Wide Web, for example, `http://www.htmlhelp.com/cgi-bin/color.cgi`. However, you do not need to restrict yourself to Web-safe colors in G2.

Text Functions

See Chapter 26, Text Parsing and Manipulation on page 1011 for information about text processing and its associated functions.

Time Functions

The G2 time functions can be used with the inference engine, the G2 Simulator, and within procedures.

In these functions, the *time-expression* argument is an integer count of the number of seconds that have elapsed since the start of the current run of G2:

the current time
the current real time

In these functions, the *time-expression* argument is a floating-point number as accurate as the operating system supplies:

the current subsecond time
the current subsecond real time

If you use the current time as an argument, the time returned is from the G2 clock. If you use the current real time as an argument, the time returned is from the real-time clock. Otherwise, the time returned is from a clock that is *time-expression* seconds or subseconds ahead of where the real-time clock was when the KB was started.

The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices on page 2169.

Computing Time with Daylight-Savings Time

Computers using the UNIX and Windows operating system support timezones and daylight-savings time, while some other operating systems do not.

To illustrate these differences, consider that the U.S. switches from daylight-savings time to standard time in November. For that month, on operating systems that recognize daylight-savings time, G2 calculates the time function as the

number of seconds in 31 days and 1 hour. For other operating systems, the time function returns the number of seconds in 31 days.

Function	Description
year (t*ime-expression*) = (*integer*)	Returns a four-digit integer representing the year for the specified *time-expression*, which must be an integer or float. An example is: year (the current time) = 2002
month (*time-expression*) = (*integer*)	Returns the month of the specified *time-expression* as an integer from the range 1 to 12 inclusive, where *time-expression* must be an integer or float. An example is: month (the current real time) = 11
day-of-the-month (*time-expression*) = (*integer*)	Returns the day of the month of the specified *time-expression* as an integer from the range 1 to 31 inclusive, where *time-expression* must be an integer or float. An example is: day-of-the-month (the current time) = 27
day-of-the-week (*time-expression*) = (*integer*)	Returns the day of the week of the specified *time-expression* as one of the symbols from this list: monday, tuesday, wednesday, thursday, friday, saturday, sunday. *time-expression* must be an integer or float. An example is: day-of-the-week (the current time) = friday
minute (*time-expression*) = (*integer*)	Returns the minute of the hour of the specified *time-expression* as an integer from the range 0 to 59 inclusive, where *time-expression* must be an integer or float. An example is: minute (10) = 29
hour (*time-expression*) = (*integer*)	Returns the hour of the specified *time-expression* as an integer from the range 0 to 23 inclusive, where *time-expression* must be an integer or float. An example is: hour (the current time) = 15

Function	Description
second (*time-expression*) = (*integer*)	Returns the second of the minute of the specified *time-expression* as an integer from the range 0 to 59 inclusive, where *time-expression* must be an integer or float. An example is: second (-2) = 10
time (*year, month, day, hour, minute, second*) = (*number*)	Returns an absolute time that can be used in other time-related functions. It's in seconds, as an integer or float (if the result exceeds the limitation of integers), which you can format using the default, interval, or timestamp display formats. You must specify the year as a four-digit integer (1995, not 95). All other arguments may be specified as one- or two-digit integers. Some examples are: time (1992, 06, 5, 13, 30, 45) = -8393 time (1992, 06, 5, 13, 30, 45) = -2 hours, 32 minutes, and 35 seconds time (1992, 06, 5, 13, 30, 45) = 5 Jun 92 1:30:45 p.m. Also notice that the returned value is only valid for current G2 session (because the internal algorithm is based on current G2 start time), it cannot be saved into KB for use when next time G2 starts.

29

Publish/Subscribe Facility

Describes how to use the publish/subscribe facility for event subcription.

Introduction

G2 provides a publish/subscribe facility, which allows application developers to implement scalable, distributed applications that can respond dynamically to changes in the application. This facility provides tools similar to what many modern development environments such as Java provide. Previously, G2 developers were required to use G2 JavaLink to access similar features, or to build their own facility within G2, which has various limitations.

The publish/subscribe facility allows applications to subscribe to these item events:

- modify — When the value of a single attribute, a list of attributes, or all attributes of an item change. This event also occurs when the value of a variable or parameter changes.
- item-color-pattern-change — When any color region of an icon changes.
- delete — When an item is deleted.
- add-item-to-workspace — When an item is added to a workspace.

- remove-item-from-workspace — When an item is removed from a workspace.
- custom-event — When a custom event is sent.

The publish/subscribe facility also provides a way of sending custom events.

When the event occurs, G2 can execute a callback procedure, which can refer to user-defined data passed to the procedure that creates the subscription. The subscription procedure returns handles, which you must use to remove subscription registrations.

Note that G2 schedules callbacks to occur in response to an event; the event does not automatically cause the callback to execute, then wait for it to finish.

When an item is deleted, all subscriptions on it are automatically deregistered.

The publish/subscribe facility supports remote application subscriptions and callbacks.

Note Subscriptions are not saved in the KB; however, they remain after resetting G2.

Application Programmer's Interface

The publish/subscribe facility API procedures appear on the g2-publish-subscribe workspace of G2 System Procedures.

For a description of these procedures, as well as the callbacks for each type of event, see the *G2 System Procedures Reference Manual*.

To display the publish/subscribe procedures:

➔ Choose Get Workspace > g2-system-procedures to display the G2 System Procedures top-level workspace, display the table of contents, and choose g2-publish-subscribe.

Registering Callbacks Remotely

To register callbacks remotely, you need to make an RPC call into G2 to a procedure that does the registration. When registering callbacks remotely, you pass a symbol as the *callback* argument, which is interpreted as the name of a procedure in the remote registration. If no such procedure exists, an error is thrown at when the callback is invoked.

Before invoking the callback remotely, any items being passed to the callback are registered for network use as if the g2-register-on-network system procedure had been called on them and their associated interface. Registering the item returns a network handle, as an integer, which is passed as the *item* argument to the callback procedure.

Note When invoking callbacks in a bridge, you must retrieve the network handle by calling `gsi_handle_of`, not `gsi_int_of`. For more information, see Chapter 9 "API Functions" in the *G2 Gateway User's Guide*.

For examples, see Example: Registering Callbacks Remotely Over a Network Interface and Example: Registering Callbacks Remotely Over a G2 Gateway Bridge.

Examples

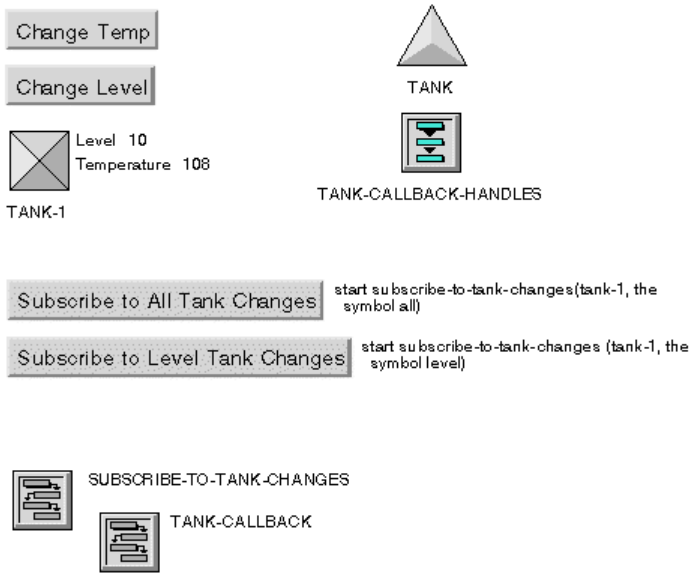
These examples are available online in `publish-subscribe-doc-ex.kb` and `publish-subscribe-remote-doc-ex.kb`, which are located in the `g2\kbs\demos` (Windows) or `/g2/kbs/demos` (UNIX), and in `pub_subscribe.c`, which is located in the `gsi` directory. To compile the `.c` file, use the `makefile` in the `gsi` directory.

Example: Subscribing to Attribute Changes

This example shows how to subscribe to attribute changes of a tank, which has two attributes, level and temperature. The subscription handles are stored in the tank-callback-handles integer list.

The subscribe-to-tank-changes procedure subscribes to changes that occur in the specified attributes of the tank. The tank-callback procedure posts a message to the Message Board when the event occurs.

The Subscribe to All Tank Changes button subscribes to all attribute changes for tank-1. The Subscribe to Level Tank Changes subscribes to attribute changes for the level attribute only for tank-1.



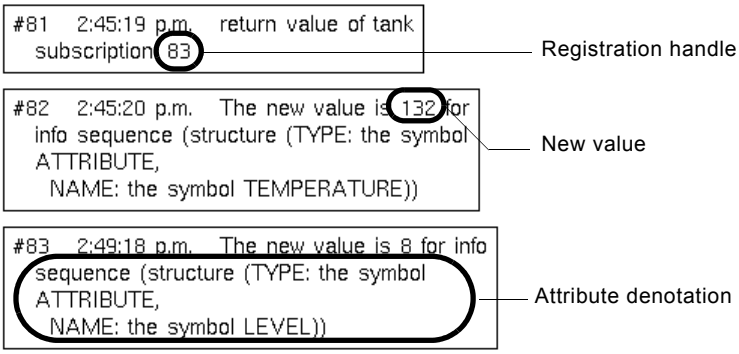
This procedure subscribes to attribute changes of a tank and executes the tank-callback when the event occurs. It posts the subscription handle to the Message Board at registration time and inserts it into the tank-callback-handles integer list.

```
subscribe-to-tank-changes(tank: class tank, attribute-spec: item-or-value)
val: value;
i: integer;
begin
    val = call g2-subscribe-to-item-attributes(tank, attribute-spec, tank-callback,
        sequence());
    post "return value of tank subscription: [val]";
    if val is an integer then
        insert val at the beginning of tank-callback-handles
    else
    begin
        for i = 0 to the number of elements in val - 1 do
            insert val[i] at the beginning of tank-callback-handles
        end
    end
end
```

Here is the callback procedure, which simply posts the new value of the changed attribute and the attribute denotation to the Message Board when the modify event occurs:

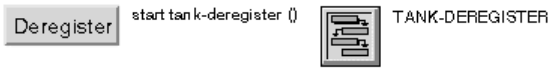
```
tank-callback(event: symbol, item: class tank, info: sequence, new-val: item-or-value,
    user-data: item-or-value, handle: integer)
begin
    if event = the symbol MODIFY then
        post "The new value is [new-val] for info [info]";
end
```

Here is the result of clicking the Subscribe to All Tank Changes button, then clicking the Change Temp button and then the Change Level button:



Example: Deregistering Subscriptions

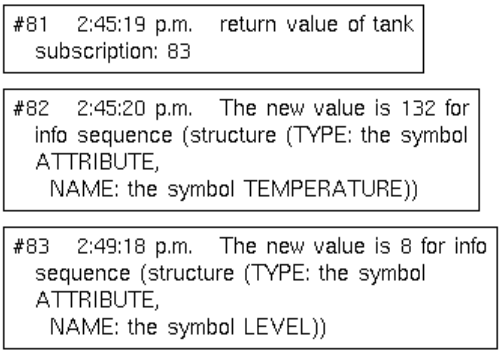
The example also shows how to deregister subscriptions:



Here is the tank-deregister procedure, which deregisters all subscription handles in the tank-callback-handles list:

```
tank-deregister()
i: integer;
begin
    for i = 0 to the number of elements in tank-callback-handles - 1 do
        start g2-deregister-subscription (tank-callback-handles[i]);
    end;
    conclude that the g2-list-sequence of tank-callback-handles = sequence();
end
```

Here is the result of clicking the Subscribe to All Tank Changes button, then clicking the Change Temp button and then the Change Level button:



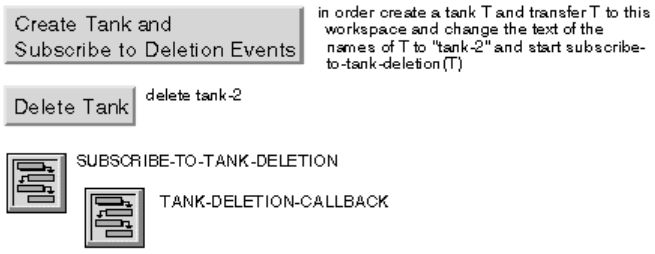
Here is the result of first clicking the Deregister button, then clicking the Subscribe to Level Changes button, and then clicking the Change Level button. Note that clicking the Change Temp button in this case would not execute the callback because the registration is only invoked for changes to the level attribute.

```
#84   2:51:43 p.m.   return value of tank
  subscription: 84
```

```
#85   2:51:47 p.m.   The new value is 7 for info
  sequence (structure (TYPE: the symbol
  ATTRIBUTE,
    NAME: the symbol LEVEL))
```

Example: Subscribing to Deletion Events

This example shows how to subscribe to deletion events for a tank. The subscribe-to-tank-deletion procedure subscribes to tank deletion events and invokes the tank-deletion-callback procedure when a tank is deleted, which posts a message to the Message Board. The button creates a tank and executes the subscription procedure.



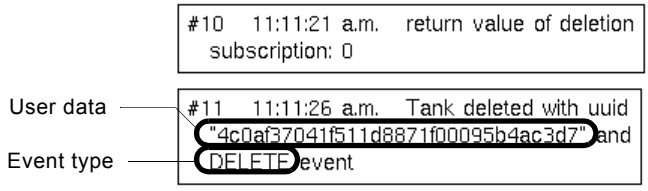
This procedure subscribes to tank deletion events and executes the tank-deletion-callback when a tank is deleted. It posts the subscription handle to the Message Board at registration time and inserts it into the tank-callback-handles integer list.

```
subscribe-to-tank-deletion(tank: class tank)
val: integer;
i: integer;
begin
    val = call g2-subscribe-to-item-deletion (tank, tank-deletion-callback, the text of
        the uuid of tank) ;
    post "return value of deletion subscription: [val]";
    insert val at the beginning of tank-callback-handles;
end
```

Here is the callback procedure, which posts the user data to the Message Board when the delete event occurs. In this case, the user data is defined as the UUID of the tank and the event type of the deleted item. It then deregisters the subscription for the deleted item.

```
tank-deletion-callback(event: symbol, item: class item, info: sequence, new-val:
    item-or-value, user-data: item-or-value, handle: integer)
begin
    if event = the symbol DELETE then
        begin
            post "Tank deleted with uuid [user-data] and [event] event";
            call g2-deregister-subscription(handle);
        end
end
```

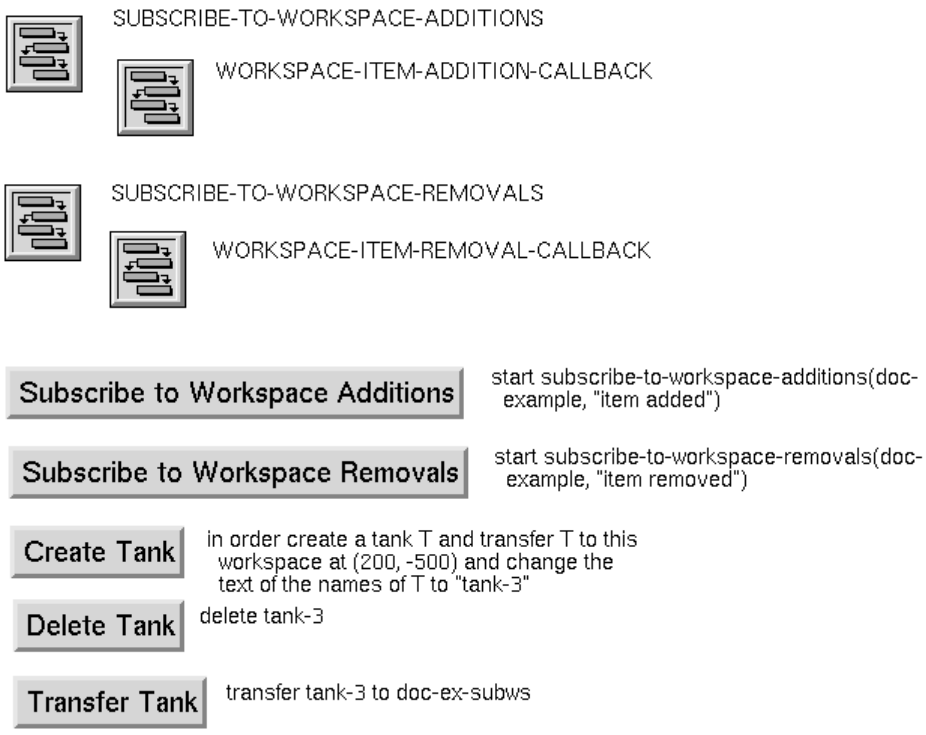
Here is the result of first clicking the Deregister button, then clicking the Create Tank and Subscribe to Deletion Events button, and then clicking the Delete Tank button:



Example: Subscribing to Workspace Events

This example shows how to subscribe to items being added to and removed from a workspace. The subscribe-to-workspace-additions procedure subscribes to workspace addition events and invokes the workspace-item-addition-callback procedure when an item is added to a workspace, which posts a message to the Message Board. Similarly, the subscribe-to-workspace-removals procedure subscribes to workspace removal events and invokes the workspace-item-removal-callback procedure when an item is removed from the workspace, which posts a message to the Message Board.

The Subscribe to Workspace Additions and Subscribe to Workspace Removals buttons execute the subscription procedures, passing doc-examples as the workspace that should listen for these events. The Create Tank button creates a tank on the current workspace, the Delete Tank buttons deletes the tank, and the Transfer Tank removes the tank from the workspace by transferring it to another workspace so it still exists.



Here is the procedure that subscribes to workspace additions and invokes the workspace-item-addition-callback when items are added to the workspace:

```
subscribe-to-workspace-additions(ws: class kb-workspace, user-data: item-or-value)
val: integer;
i: integer;
begin
    val = call g2-subscribe-to-add-item-to-workspace(ws,
        workspace-item-addition-callback, user-data);
    post "return value of subscription: [val]";
    insert val at the beginning of TANK-CALLBACK-HANDLES;
end
```

The subscription procedure for removing items from a workspace is similar except that it calls g2-subscribe-to-remove-item-from-workspace.

Here is the callback procedure for subscribing to workspace additions, which posts information to the Message Board when the add-item-to-workspace event occurs.

```
workspace-item-addition-callback(event: symbol, wksp: class kb-workspace,
    info: sequence, itm: item-or-value, user-data: item-or-value, handle: integer)
begin
    if event = the symbol ADD-ITEM-TO-WORKSPACE then
    post "add-item-to-ws-callback: event: [event], workspace: [(if the name of wksp
        exists then the name of wksp else the text of the uuid of wksp)], info: [info],
        itm: [(if the name of itm exists then the name of itm else the text
        of the uuid of itm)], user-data: [user-data], handle: [handle]";
end
```

The callback procedure for subscribing to workspace removals is similar except that it tests for the remove-item-from-workspace event. In addition, the item removal callback tests for the existence of the item that was removed from the workspace. If the item exists, it refers to the name of the item; otherwise, it indicates that the item has been deleted.

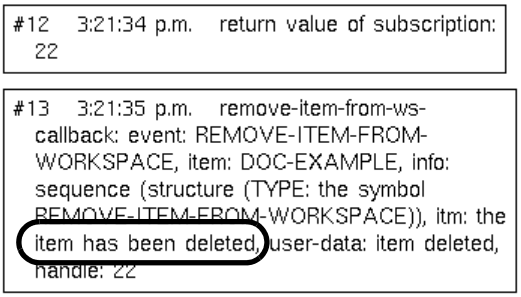
```
workspace-item-removal-callback(event: symbol, wksp: class item, info: sequence,
    itm: item-or-value, user-data: item-or-value, handle: integer)
begin
    if event = the symbol REMOVE-ITEM-FROM-WORKSPACE then
        post "remove-item-from-ws-callback: event: [event], item: [(if the name of
        wksp exists then the name of wksp else the text of the uuid of wksp)],
        info: [info], itm: [(if itm exists then the name of itm else "the item has been
        deleted")], user-data: [user-data], handle: [handle]";
    end
end
```

Here is the result of clicking the Subscribe to Workspace Additions button, then clicking the Create Tank button:

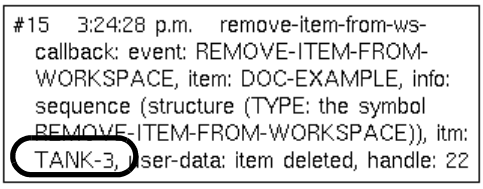
```
#10   3:20:29 p.m.   return value of subscription:
  21
```

```
#11   3:20:32 p.m.   add-item-to-ws-callback:
  event: ADD-ITEM-TO-WORKSPACE, wksp:
  DOC-EXAMPLE, info: sequence (structure
  (TYPE: the symbol ADD-ITEM-TO-
  WORKSPACE)), itm: TANK-3, user-data: item
  added, handle: 21
```

Here is the result of clicking the Subscribe to Workspace Removals button, then clicking the Delete Tank button. Because the item no longer exists, the message indicates that the item has been deleted.



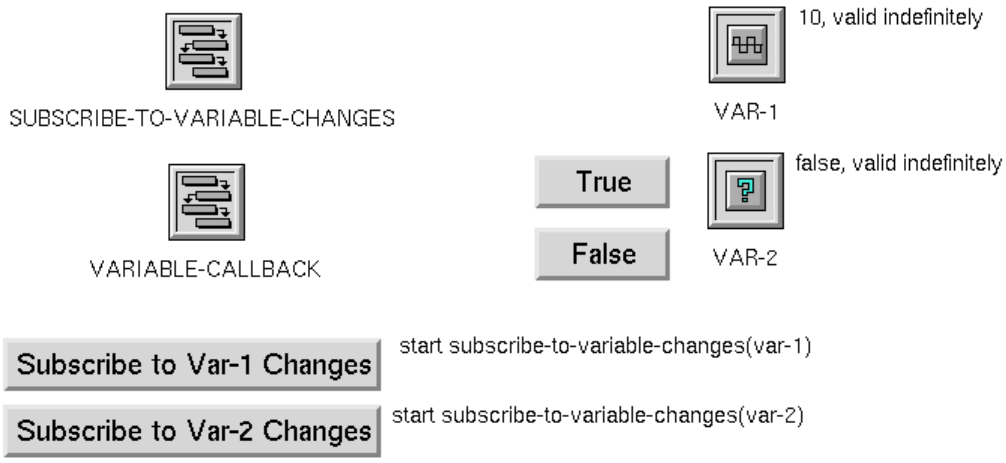
Here is the result of clicking the Subscribe to Workspace Removals button, then clicking the Transfer Tank button. This time, the message refers to the name of the item because it still exists.



Example: Subscribing to Variable Events

This example shows how to subscribe to variable changes. The subscribe-to-variable-changes procedure subscribes to variable events and invokes the variable-callback procedure when the value of the variable changes, which posts a message to the Message Board. Note that the callback is only invoked when the value of the variable changes to a new value; it does not get invoked when the value of the variable changes to the same value.

The Subscribe to Var-1 Changes and Subscribe to Var-2 Changes buttons execute the subscription procedures, passing var-1 and var-2 as arguments. The var-1 integer variable updates its value based on a formula, and the var-2 logical variable updates its value by clicking the True and False buttons.



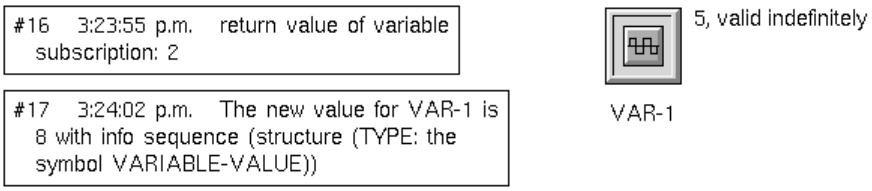
Here is the procedure that subscribes to variable changes and invokes the variable-callback when the value of the variable changes to a different value:

```
subscribe-to-variable-changes(var: class variable)
val: integer;
i: integer;
begin
    val = call g2-subscribe-to-variable-or-parameter-value (var, variable-callback,
        sequence());
    post "return value of variable subscription: [val]";
    insert val at the beginning of TANK-CALLBACK-HANDLES;
end
```

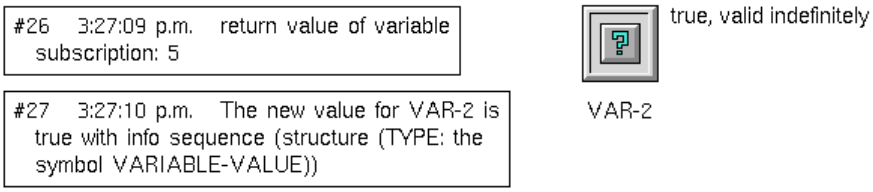
Here is the callback procedure for subscribing to variable changes, which posts information to the Message Board when the modify event occurs.

```
variable-callback(event: symbol, item: class variable, info: sequence, new-val:
    item-or-value, user-data: item-or-value, handle: integer)
begin
    if event = the symbol MODIFY then
        post "The new value for [the name of item] is [new-val] for info [info]";
    end
end
```

Here is the result of clicking the Subscribe to Var-1 Changes button, then waiting for the variable value to update:



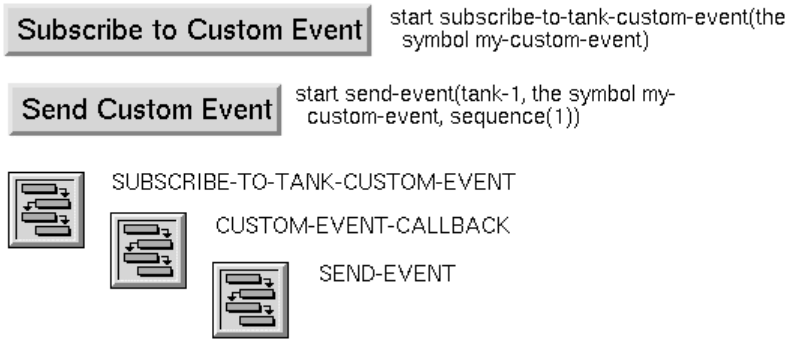
Here is the result of clicking the Subscribe to Var-2 Changes button, then clicking the True button twice. Notice that clicking the value twice does not invoke the callback because the value did not change.



Example: Subscribing to Custom Events

This example shows how to subscribe to custom events for a tank. The example is similar to the previous two examples except that for custom events, you provide the custom event as an argument to the subscription system procedure. Also, you must explicitly send the custom event.

The subscribe-to-custom-event procedure subscribes to custom events and invokes the custom-event-callback procedure when a custom event is sent, which posts a message to the Message Board that indicates the event name and its value. The send-event procedure sends a custom event to a tank. The Subscribe to Custom Event button subscribes to the my-custom-event event, and the Send Custom Event button sends the custom event to tank-1.



Here is the subscribe-to-custom-event procedure, which passes the custom event as an argument to the g2-subscribe-to-custom-event system procedure.

```
subscribe-to-tank-custom-event(event: symbol)
val: integer;
i: integer;
begin
   val = call g2-subscribe-to-custom-event(tank-1, event, custom-event-callback,
         sequence());
     post "return value of tank subscription: [val]";
     insert val at the beginning of TANK-CALLBACK-HANDLES;
end
```

Here is the custom-event-callback procedure, which posts various information to the Message Board:

```
custom-event-callback(event: symbol, i: class item, info: sequence,
    new-val: item-or-value, user-data: item-or-value, handle: integer)
begin
    if event = the symbol custom-event then
    post "Custom-event-callback: event: [event], item: [the name of i], info: [info],
    new-val: [new-val], user-data: [user-data], handle: [handle]";
end
```

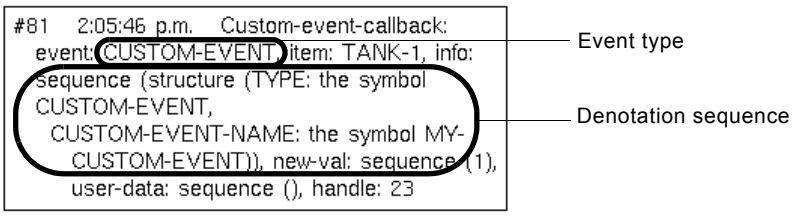
Here is the send-event procedure, which sends a custom event to an item:

```
send-event(i: class item, event: symbol, info: item-or-value)
begin
    call g2-send-notification-to-item(i, event, info);
end
```

Clicking the Subscribe to Custom Event button posts the subscription handle to the Message Board:

```
#102   4:12:23 p.m.   return value of tank
  subscription: 28
```

Clicking the Send Custom Event button posts the following information to the Message Board, including the event type, which is custom-event, and the custom event name, which is my-custom-event:



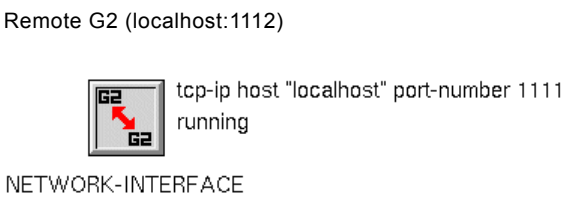
Example: Registering Callbacks Remotely Over a Network Interface

This example shows how to register callbacks in a remote G2 over a G2-to-G2 interface. You can follow a similar procedure to deregister callbacks remotely over an interface.

To register callbacks remotely over a G2-to-G2 interface:

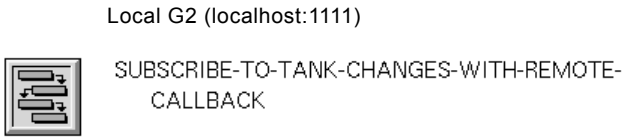
3 In the remote G2, create a g2-to-g2-interface and configure it with the network information of the local G2 that will define the subscription procedure:

For example, the g2-to-g2-interface named network-interface connects to the G2 running on the localhost at port 1111:



4 On the local machine, create a procedure that performs the subscription, which will be called remotely across the G2-to-G2 interface.

Here is a procedure in the local G2 running on port 1111, which subscribes to tank changes and uses a remote callback:



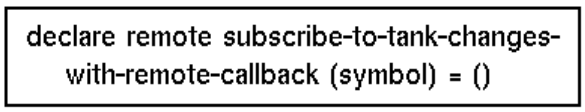
Your procedure passes in the remote callback as a symbol to the G2 system procedure that does the subscription. Compare this to local subscriptions in which you pass in the callback procedure itself.

This procedure subscribes to level attribute changes in tank-1 and executes the remote callback when the event occurs. The remote callback name is passed in as a symbol argument to your procedure. When the remote G2 calls this procedure, it will provide the name of the remote callback as a symbol argument.

```
subscribe-to-tank-changes-with-remote-callback(remote-callback-name: symbol)
val: value;
i: integer;
begin
    val = call g2-subscribe-to-item-attributes(tank-1, the symbol level,
        remote-callback-name, sequence());
    post "return value of tank subscription: [val]";
    if val is an integer then
        insert val at the beginning of TANK-CALLBACK-HANDLES
    else
    begin
        for i = 0 to the number of elements in val - 1 do
        insert val[i] at the beginning of tank-callback-handles
        end
    end
end
```

5 In the remote G2, create a remote procedure declaration for the procedure in the local G2 that performs the subscription.

This remote procedure declaration is for the subscribe-to-tank-changes-with-remote-callback procedure you saw earlier:



6 In the remote G2, create a callback procedure that is invoked when the subscription event occurs in the local G2.

This procedure typically makes calls into the local G2 for information about the item on which the event occurs.

This procedure calls another local procedure named get-item-name-from-handle across the interface when the event occurs and posts a message to the Message Board that includes the item name:

Remote G2 (localhost:1112)

REMOTE-CALLBACK

```
remote-callback (event: symbol, host-handle: integer, info: sequence,
     new-val: item-or-value, user-data: item-or-value, handle: integer)
item-name: symbol;
begin
     item-name = call get-item-name-from-handle (host-handle) across
          network-interface;
     if event = the symbol MODIFY then
     post "The new value is [new-val] for [info] for [item-name]";
end
```

7 In the local G2, create any other necessary procedures that the remote callback references, then in the remote G2, create remote procedure declarations for the procedures.

Here is the local procedure referenced in the remote callback and its remote procedure declaration in the remote G2:

Local G2 (localhost:1111)

GET-ITEM-NAME-FROM-HANDLE

Remote G2 (localhost:1112)

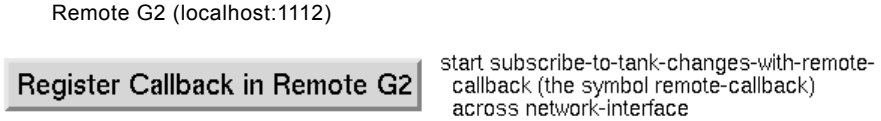
```
declare remote get-item-name-from-handle
   (integer) = (symbol)
```

This procedure gets the item name from the network handle, using G2 system procedures:

```
get-item-name-from-handle (handle: integer) = (symbol)
interface: class network-interface;
item: class item;
begin
    interface = call g2-current-remote-interface();
    item = call g2-get-item-from-network-handle (handle, interface);
    return the name of item;
end
```

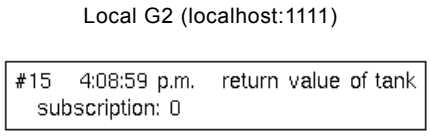
8 In the remote G2, create some way to call the local procedure that performs the subscription across the network interface.

Here is an action button that calls subscribe-to-tank-changes-with-remote-callback across the network-interface, passing in the name of the remote callback as a symbol as the argument to the procedure:



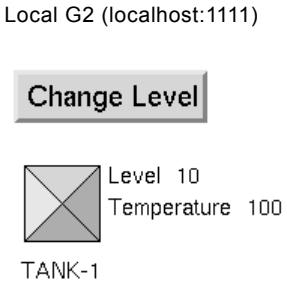
9 In the remote G2, call the local procedure.

Clicking the Register Callback in Remote G2 action button causes this message to appear in the local G2 indicating that the subscription has been created:

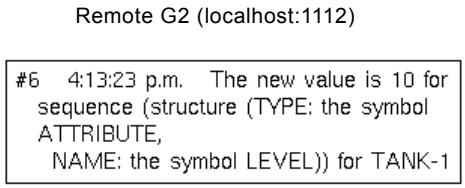


10 In the local G2, execute the subscription event to invoke the remote callback.

This action button changes the value of the level attribute of tank-1 to 10:



Changing the level in the local G2 invokes the remote callback, which causes this message to appear in the remote G2:



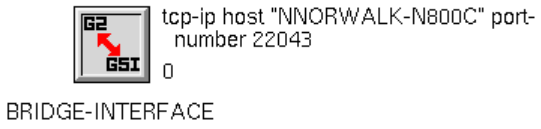
Example: Registering Callbacks Remotely Over a G2 Gateway Bridge

This example shows how to register callbacks defined in a G2 Gateway bridge over a GSI interface.

To register callbacks remotely over a G2 Gateway Bridge:

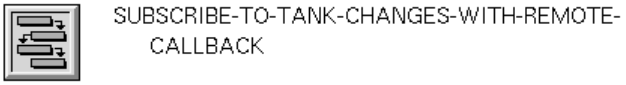
1 In your G2 application, create a gsi-interface and configure it with the network information of the local G2 that defines the subscription procedure:

For example, the gsi-interface named bridge-interface specifies the host and port of the G2 Gateway bridge process and the protocol to use:



2 In your G2 application, create a procedure that performs the subscription, which will be called remotely from your G2 Gateway bridge.

Here is the same procedure as in the previous example, which subscribes to tank changes remotely:



Recall that this procedure takes as an argument a symbol, which is the remote callback name.

3 Create a G2 Gateway bridge that uses gsi_rpc_declare_local to declare a local procedure as the remote callback in the gsi_set_up.

```
void gsi_set_up ()
{
   gsi_rpc_declare_local(rpc_remote_callback,
   "RPC-REMOTE-CALLBACK");
}
```

4 Use declare_gsi_rpc_receiver_fn to allow the local procedure that is the remote callback to receive values from G2.

```
/* Allow the local procedure that is the callback to receive
values from G2 */

extern declare_gsi_rpc_local_fn(rpc_remote_callback);
```

5 Define the local procedure that is the remote callback.

```
/* Remote callback that G2 invokes when an attribute of the tank
changes */

void rpc_remote_callback(gsi_item item_array[], gsi_int count,
          call_identifier_type call_index)
{
   int i;
   /* event: symbol, handle: integer, info: sequence,
   new-val: item-or-value, user-data: item-or-value,
   callback-handle: integer */
   printf("Event: %s, Item-handle: %d, Level: %d\n",

   gsi_sym_of(item_array[0]),
   gsi_handle_of(item_array[1]),
   gsi_int_of(item_array[3]));

   for (i=0;i<count;i++)
      gsirtl_display_item_or_value(item_array[i], 0, 0);
}
```

6 Use gsi_set_sym to tell G2 the name of the remote callback, as a symbol.

```
/* Tell G2 the name of the remote callback */

  gsi_set_sym(args[0], "RPC-REMOTE-CALLBACK");
```

7 Use gsi_function_handle_type to create a handle for the remote procedure in G2 that performs the subscription.

```
/* Create a handle for the remote procedure in G2 */

gsi_function_handle_type subscribe_to_tank_changes_remotely;
```

8 Use `gsi_rpc_declare_remote` to declare a remote procedure for the procedure in G2 that performs the subscription in the `gsi_int gsi_initialize_context`.

```
/* Declare a remote procedure for the subscription
procedure in G2 */

  gsi_rpc_declare_remote(&subscribe_to_tank_changes_remotely,|
"SUBSCRIBE-TO-TANK-CHANGES-REMOTELY", NULL_PTR, 1, 0,
context);
```

9 Use `gsi_rpc_start` to start the subscription procedure remotely from the bridge in the `gsi_int gsi_initialize_context`.

```
/* Start the subscription procedure in G2 */

  gsi_rpc_start(subscribe_to_tank_changes_remotely, args, context);
```

10 Create an executable for the G2 Gateway bridge.

To run the G2 Gateway bridge that registers callbacks remotely:

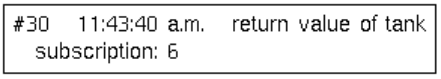
1 Invoke the executable for the G2 Gateway bridge you created.

The bridge is ready to accept connections from a G2 application.

2 Establish the connection to the bridge by restarting G2 or by disabling and enabling the interface object.

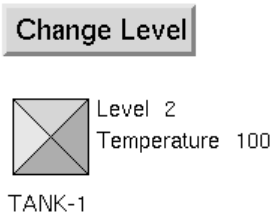
The bridge should now be connected to your G2 application, and the gsi-interface status should be 2.

Recall that the subscribe-to-tank-changes procedure in G2 posts a message to the message board. Thus, you should see the following message in the G2 Message Board, indicating that the bridge process has invoked the subscription procedure in G2 remotely.



3 In G2, send an event that will trigger the remote callback.

In our example, this action button changes the level attribute of tank-1, which is the item whose attribute is registered.



Changing the level attribute of the tank results in the following output in the command window that started the bridge. Notice that the callback reports values that it receives from G2 for the event name, handle, and attribute value.

```
Event: MODIFY, Item-handle: 1, Level: 2
symbol value MODIFY
Item handle 1
a sequence with 1 elements
a structure with attributes...
TYPE: symbol value ATTRIBUTE
NAME: symbol value LEVEL
integer value 5
a sequence with 0 elements
integer value 0
```

30

G2 Graphical Language (G2GL)

Describes G2GL, a graphical language for describing processes.

Introduction

The G2 Graphical Language (G2GL) provides a self-contained graphical programming environment for the specification of any type of process. It allows the execution of processes, including business, industrial, and general reasoning processes, directly within G2.

The process activities are generally based on the Business Process Execution Language for Web Services (BPEL4WS or BPEL for short) language. BPEL is an industry initiative, now managed by OASIS, to establish an effective standard framework for describing and defining high-level business processes that are offered as Web services. The following companies are members of the BPEL technical committee and provide BPEL-inspired products: IBM, Oracle Corporation, Microsoft Corporation, SAP, BEA Systems Inc., Gensym, and others.

G2GL provides these features:

- Implements most of BPEL and provides expressive power and process invocation options beyond BPEL, as well as high execution performance, to support lower-level as well as higher-level process. This means that, unlike most other BPEL products, developers can do everything at the BPEL level, rather than having to descend into Java, for example.
- Integrates process modeling, authoring, compilation, execution, animation, and debugging together in one software environment that can run online as well as offline, providing large productivity and manageability advantages over other major implementations of BPEL.
- Integrates with G2 and, thus, can use G2's powerful domain modeling, reasoning, real-time data handling, and system integration capabilities.

You can model process flows graphically in G2GL, or you can import them from a BPEL document. You can also export G2GL process specifications to a BPEL document. G2GL uses a namespace prefix when exporting G2GL extensions to BPEL.

G2GL provides many activities that you can use to compose an execution flow. Typical activities include receiving messages, setting variables on message parts to the value of expressions, deciding on next steps, invoking external services, waiting, starting parallel executions, and calling G2 procedures. For example, a process might receive messages from an external system, evaluate expressions and metrics, decide next steps, and call external services embedded in COM objects.

You compile and execute G2GL processes within G2. Multiple G2GL processes can execute in parallel. A G2GL process integrates fully with G2 procedures, which means you can invoke G2GL processes from G2 procedures, and you can call G2 procedures from within a G2GL process. G2GL provides graphical debugging capabilities for process execution.

G2GL processes are inherently multithreaded, that is, they have built-in logic to share processing by advancing their execution in fraction-of-a-second time slices. Contrast this with G2 procedures, which can lock up G2 for up to the time allowed by the uninterrupted procedure execution limit, which defaults to 30 seconds.

G2GL relies on a variant of the G2 expression language and on G2 procedures to implement integration mechanisms with G2, COM, JMS, custom SOAP bridges, and so on.

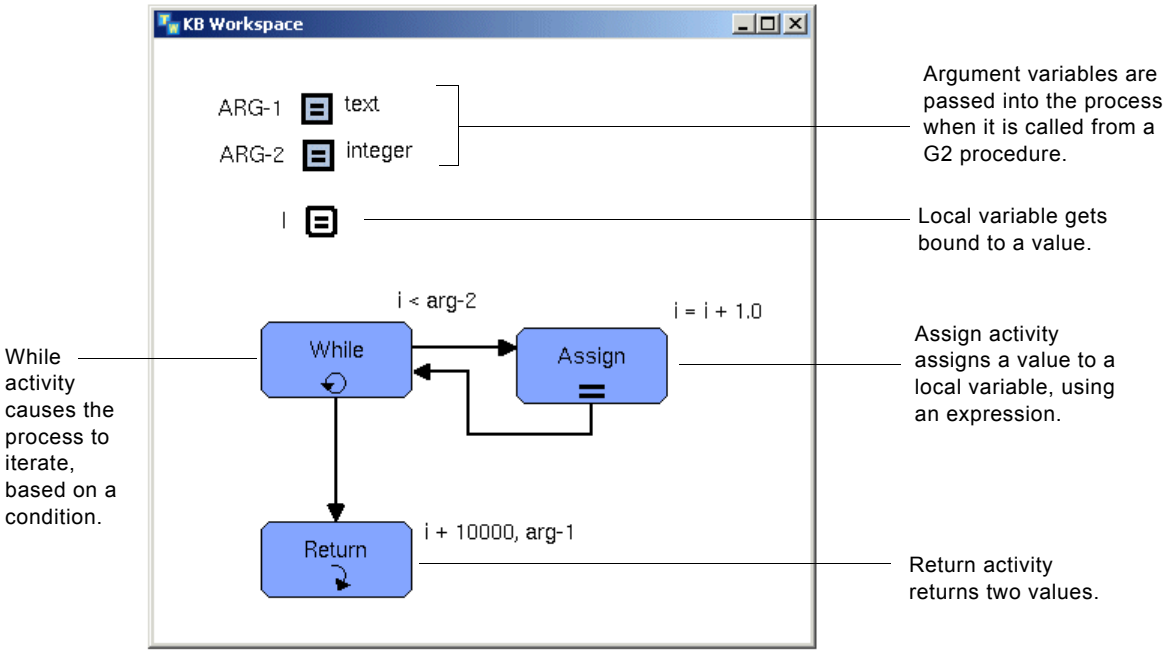
Terms and Concepts

- BPEL: Business Process Execution Language.
- BPEL4WS: Business Process Execution Language for Web Services Version 1.1 (`http://www.oasis-open.org/cover/bpel4ws.html`).
- COM: Component Object Model (`http://www.microsoft.com/com/`)
- WS-BPEL: Business Process Execution Language for Web Services Version 2.0
- WSDL: Web Services Description Language (`http://www.w3.org/TR/wsdl`).
- XML: Extensible Markup Language.

Creating G2GL Processes

A G2GL process is a graphical representation of a flow chart of activities. You create the process on the subworkspace of a g2gl-process object as the **process body**. The process body consists of activities, local variables, argument variables, and various types of handlers, as needed to specify the process.

Here is a sample process that shows several features:



You can create G2GL processes that execute a flow chart of activities to perform calculations, assign values to local variables, create messages, and return values.

G2GL processes provide a graphical procedural programming environment, an alternative to the G2 programming language.

This section describes the G2GL activities that you can use to create arbitrary process flows. For information on activities that provide communication in a process, see Communicating Between G2GL Processes.

G2GL processes can declare local variables of various types to which the process can assign values by using the Assign activity. You use the Assign activity to make general assignment statements, using the G2GL expression language. The G2GL expression language provides many of the features of the G2 expression language, including arithmetic, relational, and logical operators, text expressions, and symbols. To provide communication, processes can use the Assign activity to assign variables to message parts.

G2GL processes can define argument variables, which you pass into the process when it executes by calling the g2-call-g2gl-process-as-procedure system procedure. G2GL processes can also return values by executing a Return activity. For more information, see Calling a G2GL Process as a Procedure.

Note You cannot subclass G2GL activity classes.

Using G2GL within the Business Process Management System Module

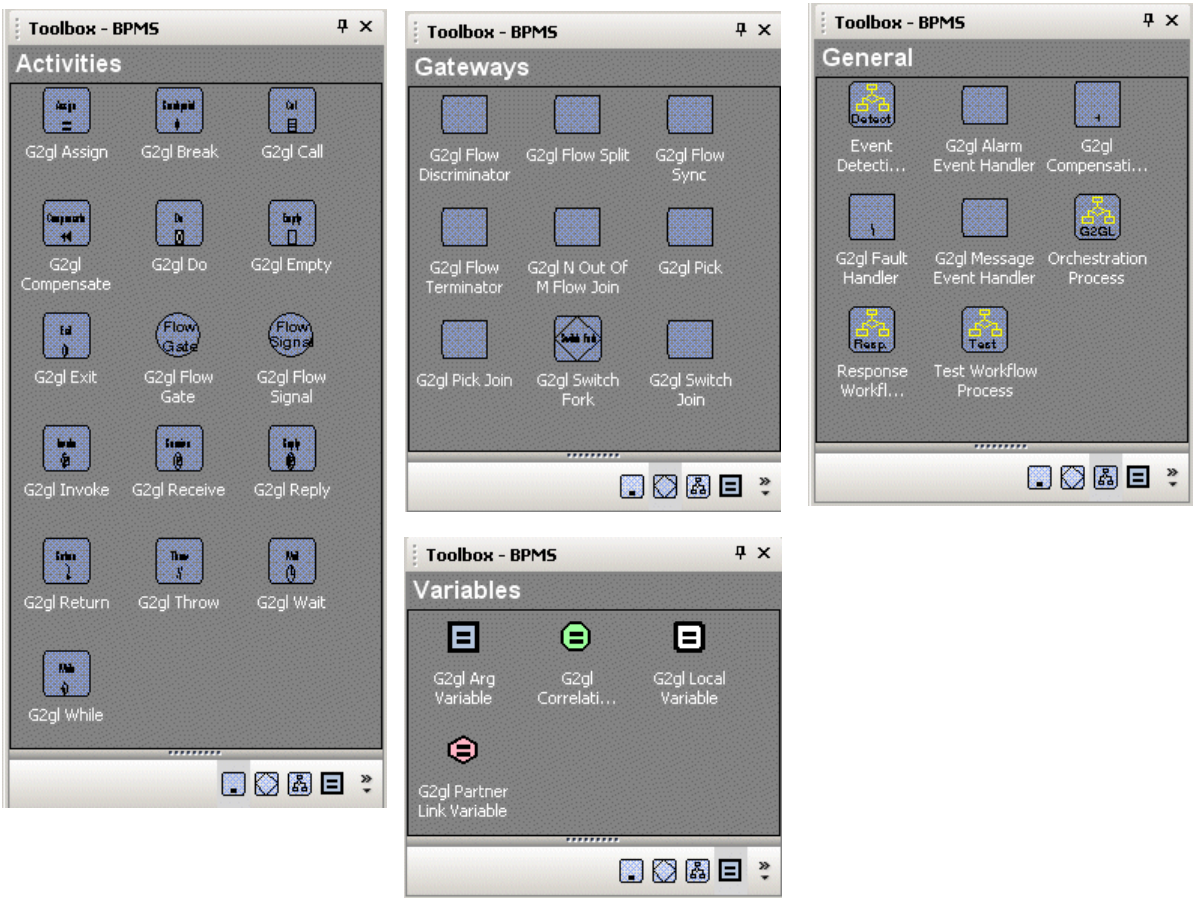
The Business Process Management System (BPMS) module, which is part of the G2 Developer's Utilities, provides the following extensions to G2GL:

- Windows dialogs and palettes for all G2GL blocks.
- A class hierarchy of G2GL process subclasses to organizes the processes as detection flows, tests, responses, or orchestration processes. These processes are automatically organized in the navigator tree view and the Project menu.
- Standard menus, message browser integration, and APIs to invoke the detection, test, and response processes for a domain object.
- Predefined G2 services that can be called from G2GL processes including services for interacting with OS processes, performing ping and trace route operations on a computer, sending email, interacting with databases and files, creating or querying operator messages and event states, generating SymCure events, and invoking BRMS rules.

To use G2GL within the BPMS module:

1. Load `bpms.kb` from the `kbs` subdirectory of the `g2i` directory of your G2 bundle installation.
2. To access a toolbox of G2GL blocks, choose View > Toolbox - BPMS.

Here are the palettes in the BPMS toolbox:



For more information, see the *Business Process Management System Users' Guide*.

Summary of G2GL Activities

Here is a summary of the various types of G2GL activities:

- Activities that execute statements and return values, and that provide integration with G2 and external systems:
  - Assign — Assigns values to one or more local variables within the process.
  - Return — Returns one or more values from a G2GL procedure, and can have one or more G2GL expressions that determine the values.
  - Do — Concludes values into attributes of G2 items, can also assign values to local variables, and can perform any general statements, using the G2GL expression language.
  - Call — Calls a G2 procedure or G2GL process, passing arguments and receiving return values, which can be assigned to local variables.

- Activities that control the flow of execution:
  - Switch Fork — Provides a two-way (yes/no) decision point with an expression representing the switch condition.
  - Switch Join — Brings together, through its top-edge input connections, the branches of a Switch Fork activity or a group of Switch Fork activities.
  - While — Provides iteration in a process, based on an expression representing the iteration (loop) condition.
  - Wait — Has an expression representing a duration or a deadline and waits until the determined time.
  - Flow Split — Splits the process flow into two or more branches, which execute concurrently.
  - Flow Sync — Brings together, through its top-edge in connections, and synchronizes the branches of a Flow Split activity.
  - Flow Discriminator — Brings together the branches of a Flow Split activity, and goes to the next activity when one of the input process threads executes.
  - N-Out-Of-M Flow Join — Brings together the branches of a Flow Split activity, and goes to the next activity when n out of m input process threads execute.
  - Flow Terminator — Brings together the branches of a Flow Split activity, without synchronizing, and goes to the next activity when one of the input process threads executes, immediately terminating all other incoming process threads.
  - Flow Signal — Provides a cross-branch synchronization signal within a flow, where the top input connection is from the source activity, the left or right output connection goes to a Flow Gate activity, and the bottom output connection, if any, goes to the next activity.
  - Flow Gate — Receives a cross-branch synchronization signal within a flow, where the top input connection, if any, is from the previous activity, the left or right input connections (one or more) comes from a Flow Signal activity, and the bottom output connection is to the next activity.
  - Empty — Performs no action.
  - Exit — Provides an optional abrupt termination activity for any G2GL process or indicates the normal end of a process.
- Activities that provide fault handling and debugging
  - Breakpoint — Creates and displays an individual execution display for a process, if one does not already exist, and creates a breakpoint, waiting for the user to click the breakpoint icon before proceeding. This activity does not prevent other concurrent threads from continuing.

  - Throw — Throws a fault to the nearest applicable named Fault Handler.
  - Compensate — Appears only on the body of a Fault Handler or Compensation Handler; explicitly invokes the Compensation Handler for a named Scope.
- Activities with bodies that define scopes and handlers
  - Scope — Provides a subordinate process that executes within a process, where activities within the scope body can refer to and set higher-level variables.
  - Fault Handler — Provides a subordinate process like a Scope, which handles system-defined faults and user-defined faults that a Throw activity generates and may specify a fault data argument variable. The Fault Handler can have a left input connection from an Invoke activity, in which case it serves a local handler for that activity's invocation action.
  - Alarm Event Handler — Provides a subordinate process like a Scope, which executes based on a duration or as an alarm clock, similar to a Wait activity that is always active.
  - Message Event Handler — Provides a subordinate process like a Scope, which asynchronously handles messages with a given operation name as if it were a Receive activity that is always active.
  - Compensation Handler — Provides a subordinate process like a Scope, which specifies how to compensate for (for example, undo) the work done by a Scope in case a fault is signalled in a parent Scope after the Scope has completed successfully.
- Activities that provide communication
  - Invoke — Invokes named operations in linked partners; can provide one-way or two-way communication by sending an operation invocation and optionally receiving a response message transmission; and can connect to local fault handlers specific to this activity, using optional right output connection.
  - Receive — Receives messages that an Invoke activity sends and can instantiate a process by receiving a message.
  - Reply — Replies to messages that an Invoke activity sends by sending a response or fault message to complete a two-way communication.
  - Pick — Receives one of several messages sent by Invoke activities; branches to one of several Receive activities, based on a received message; can also branch to a Wait activity as a time-out; and can instantiate process by receiving a message.
  - Pick Join — Brings together the branches of a Pick activity.

Creating a G2GL Process

To create a G2GL process, you create an instance of a g2gl-process object or a subclass, create a subworkspace as the process body, and place the desired types of objects on the body.

You can create local variables, which the process uses within the process flow. You can also create argument variables, whose values are passed to the process when you execute it as procedure. These are analogous to local variables and procedure arguments in a G2 procedure.

You create and connect various types of activities to describe the process flow and perform operations within the process body. The process body can also contain various types of handlers, such as fault and alarm handlers.

For a description of how to configure the attributes of a G2GL process for debugging, see Debugging G2GL Processes.

For a description of the other attributes, see BPEL Compliance.

To create a G2GL process:

1 Do one of the following:

➔ Choose KB Workspace > New Object > G2GL-object > G2GL-process.

or

➔ With `bpms.kb` loaded, choose View > Toolbox - BPMS, display the General palette, and choose one of the four types of Workflow Processes: Event Detection, Test, Response, or Orchestration.

By creating one of these types of processes, you can manage and execute different categories of processes together.

2 Configure the names attribute of the G2GL process.

Tip To configure the attributes through a properties dialog, with `bpms.kb` loaded, switch to any user mode except administrator.

Note The names attribute is required for processes that communicate via partner links.

---

3 Choose create subworkspace on the G2GL process object or Show Detail on a BPMS workflow process.

The subworkspace of the g2gl-process object is the process body.

4 Create and configure g2gl-local-variable and g2gl-arg-variable objects to specify local variables and argument variables, respectively, for the process.

Create variables from the New Object menu or from the Variables palette of the BPMS toolbox.

Place the argument variables above the local variables at the top of the process body. The order of the argument variables in the process body determines their order when passing arguments to the process, where the top-most variable is the first argument, the next variable down is the second argument, and so on.

For details on configuring variables, see Creating Local and Argument Variables.

For information on passing arguments to G2GL processes, see Executing G2GL Processes.

5 Create, configure, and connect g2gl-activity objects to describe the process flow.

For details, see:

- Creating G2GL Processes.
- Communicating Between G2GL Processes.

6 Create and configure g2gl-activity-with-body objects to define scopes and handlers.

For details, see Defining Scopes and Handlers.

7 Choose Redo Layout on the body of a G2GL process to aesthetically reconfigure the variables and activities, as needed.

The Redo Layout menu choice places argument variables at the top of the process body, local variables below the argument variables, handlers below the variables, and flow chart activities below the handlers.

To redo the layout programmatically, use the g2-system-command system procedure, which is described in User Interface Operations in the *G2 System Procedures Reference Manual*.

The first argument is the symbol redo-layout. The *item* argument is a G2GL body, that is, the subworkspace of a g2gl-process or g2gl-activity-with-body.
For example:

```
start g2-system-command
    (the symbol redo-layout, my-g2-window, bpel-flow-with2-loops-body,
    the symbol none)
```

Creating Local and Argument Variables

You declare local variables in G2GL process bodies to hold data. You can also declare argument variables to a G2GL process that is used as a procedure. You pass in these arguments by calling the g2-call-g2gl-process-as-procedure system procedure.

Place the argument variables above the local variables at the top of the process body. The order of the argument variables in the process body determines their order when passing arguments to the process, where the top-most variable is the first argument, the next variable down is the second argument, and so on.

The local and argument variables types are:

- general — Accepts any item or value.
- float — Floating point values.
- integer — Integer values.
- truth-value — True or false.
- text — Text strings.
- symbol — G2 symbols.
- sequence — G2 sequences.
- structure — G2 structures.
- class *class-name* — Class names, where *class-name* is any G2 class.

For information on passing arguments to a process, see Executing G2GL Processes.

For information on partner link variables and correlation variables, see Communicating Between G2GL Processes.

To create a G2GL variable:

1 Do one of the following:

➔ Choose KB Workspace > New Object > G2GL-object > G2GL-local-variable or G2GL-arg-variable.

or

➔ With `bpms.kb` loaded, choose View > Toolbox - BPMS, display the Variables palette, and choose a variable.

Place the variable on a G2GL process body above the flow chart.

The argument variables must be at the top in the order in which they are passed into the process.

2 Configure the names attribute to name the local variable.

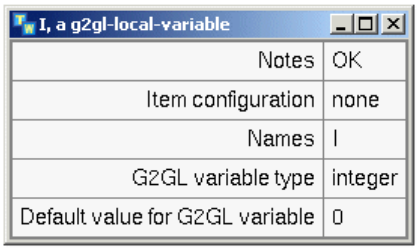
3 Configure the g2gl-variable-type to be any of the above variable types.

4 Optionally, configure default-value-for-g2gl-variable to be the initial value for the variable, which must match its type.

The default value type must correspond with the variable type.

For example, here is an integer variable with an initial value of 0:

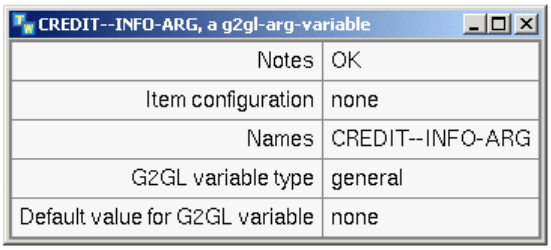
I



I, a g2gl-local-variable	
Notes	OK
Item configuration	none
Names	I
G2GL variable type	integer
Default value for G2GL variable	0

Here is an argument variable of type general:

CREDIT--INFO-ARG



CREDIT--INFO-ARG, a g2gl-arg-variable	
Notes	OK
Item configuration	none
Names	CREDIT--INFO-ARG
G2GL variable type	general
Default value for G2GL variable	none

G2GL Expressions

Numerous activities use G2GL expressions. A G2GL expression is similar to a G2 expression in that it can be a local variable, arithmetic expression, time expression, truth-value expression, text expression, or symbolic expression. However, note that the G2GL expression language is separate from the G2 expression language.

For example, you use G2GL expressions to determine:

- Variable values in an Assign activity.
- Return values in a Return activity.
- Concluded attribute values of items in a Do activity.
- Iteration condition of a While activity.
- Duration or deadline expression of a Wait activity.
- Switch Fork condition of a Switch Fork activity.

A *g2gl-expression* is any of the following expressions, which can be used in these contexts:

Expression	Example	Description
Any G2GL expression	Any of the following expressions	Used in any context that accepts a *g2gl-expression*
arithmetic-expression	*a* * *b* - *c* + *d* / *e* (*a* * (*b* - *c*) + *d*) / *d* *a* ^ *b* where *a*, *b*, *c*, *d*, and *e* are variable names, integers, floats, or a *time-expression*	Used when assigning values to integer or float variable types. For information about the precedence order of arithmetic expressions, see Using Operators in Expressions.

Expression	Example	Description
time-expression	the current subsecond time the current subsecond real time the current time the current real time the current system time the current system real time the current time + 10 seconds 10 seconds 2 hours and 10 seconds 5 days, 2 hours, and 10 seconds	Used when assigning values to float or integer variable types or anywhere that the syntax accepts a *time-expression*. The G2GL time expressions are analogous to the G2 time expressions with the same syntax. For more information, see Referring to the Current Time.
truth-expression	true false (a > b) and (c <= d)	Used when assigning values to truth-value variable types.
"*text*"	"here is some text" "Random = [random(100)]"	Used when assigning values to text variable types.
the symbol *symbol-name*	the symbol 123-abc	Used when assigning values to symbol variable types.
sequence-expression	sequence(1,2,3)	Used when assigning sequence variable types.
sequence-element	my-sequence[0] my-sequence[1][2]	Used for accessing elements of a sequence.
structure-expression	structure (name: tasha birthday: 112760)	Used when assigning structure variable types.
if *truth-expression* then *g2gl-expression* else *g2gl-expression*	(if (a > b) then (c = 10) [else (c = 20)])	Used in any context that accepts a *g2gl-expression*.

Expression	Example	Description
the *message-part* of *message-variable-name*	credit-rating = the credit-rating of credit-report the credit-rating of credit-report = the symbol good	Used when getting or setting the value of a part of a message bound to a variable. For more information, see Message Part Assignment Statements.
the *attribute-name* of *item*	the my-attr of item-1	Used for referencing G2 item attributes. You pass G2 items as arguments to the process or provide them as return values from G2 procedure calls.
this process	this process	Returns the current g2gl-process.
function (*arguments*)	average (11, 27, 60) my-func (11, 27, 60)	Calls any of the G2 system-defined functions or any user-defined function.

G2GL Statements

The following sections describe the various types of G2GL statements that you can use.

General Variable Assignment Statements

To assign a value to one or more local variables, use this syntax:

variable-name = *g2gl-expression*[; *variable-name* = *g2gl-expression*] . . .

For example:

```
i = i + 1
i = i + 1; j = j + 1
i = true
i = “hello world”
i = the symbol red
i = sequence (the symbol julian, the symbol simon)
i = structure(person: the symbol julian, birthday: 090987)
```

You use the Assign activity to make general variable assignment statements.

Message Part Assignment Statements

G2GL processes communicate with each other and with Web services by using message structures containing XML data. A Web service message has a set of message parts, represented as the attributes of a structure.

To assign a value to a message part, use this syntax:

the *message-part* of *message-variable-name* = *g2gl-expression*

where:

- *message-part* is an attribute name.
- *message-variable-name* is a variable that is either uninitialized or is bound to a Web service message structure.
- *g2gl-expression* is an expression whose value is a text, an XML element value, or a sequence of texts and/or XML element values.

An XML element value is a structure representing an XML element with this syntax:

structure
(tag-name: *text*,
attributes: *structure*,
children: *sequence*)

where:

- tag-name is the element tag name. This attribute is required.
- attributes is a structure containing named attribute values, which are texts. This attribute is optional.
- children is a sequence of XML elements and/or texts. This attribute is optional.

Attribute names with hyphens correspond to XML names with mixed case. For example, a structure attribute named my-attribute corresponds to an XML attribute named `myAttribute`.

For example, this XML text:

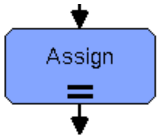
```
<elt attrName="attrValue">
    <child>text1</child>
    text2
</elt>
```

corresponds to this XML element value:

```
structure (tag-name: "elt",
        attributes: structure (attr-name: "attrValue"),
        children: sequence
        (structure (tag-name: "child",
                children: sequence ("text1")),
         "text2"))
```

You can use the Assign activity or the Do activity to make message part assignment statements.

Assigning Values



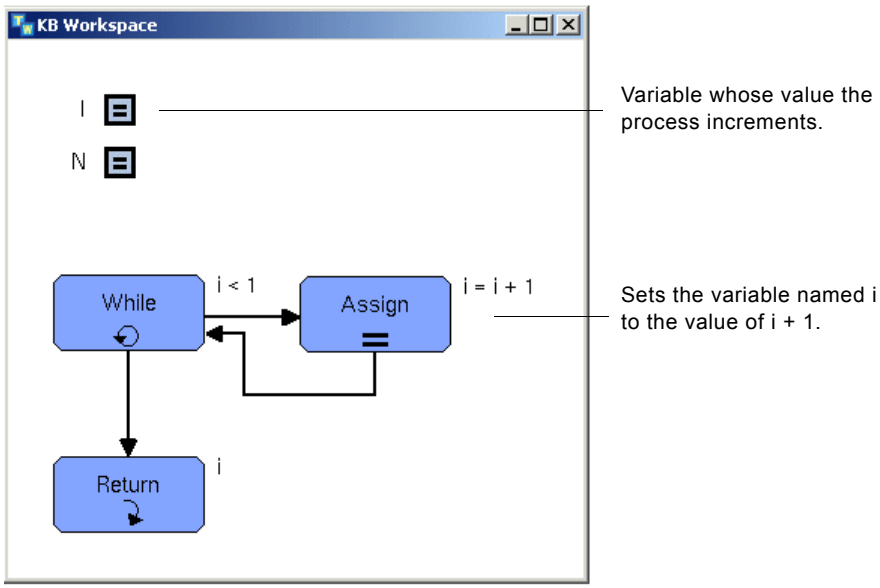
You use the Assign activity to set local variables to values.

Assigning Variables to General Value Types

To assign values to any of the standard types of variables (general, float, integer, truth-value, text, or symbol), you configure the g2gl-assignments attribute of the Assign activity to be a general variable assignment statement.

For details, see General Variable Assignment Statements.

For example, this Assign activity sets the variable i to the value of the expression i + 1:



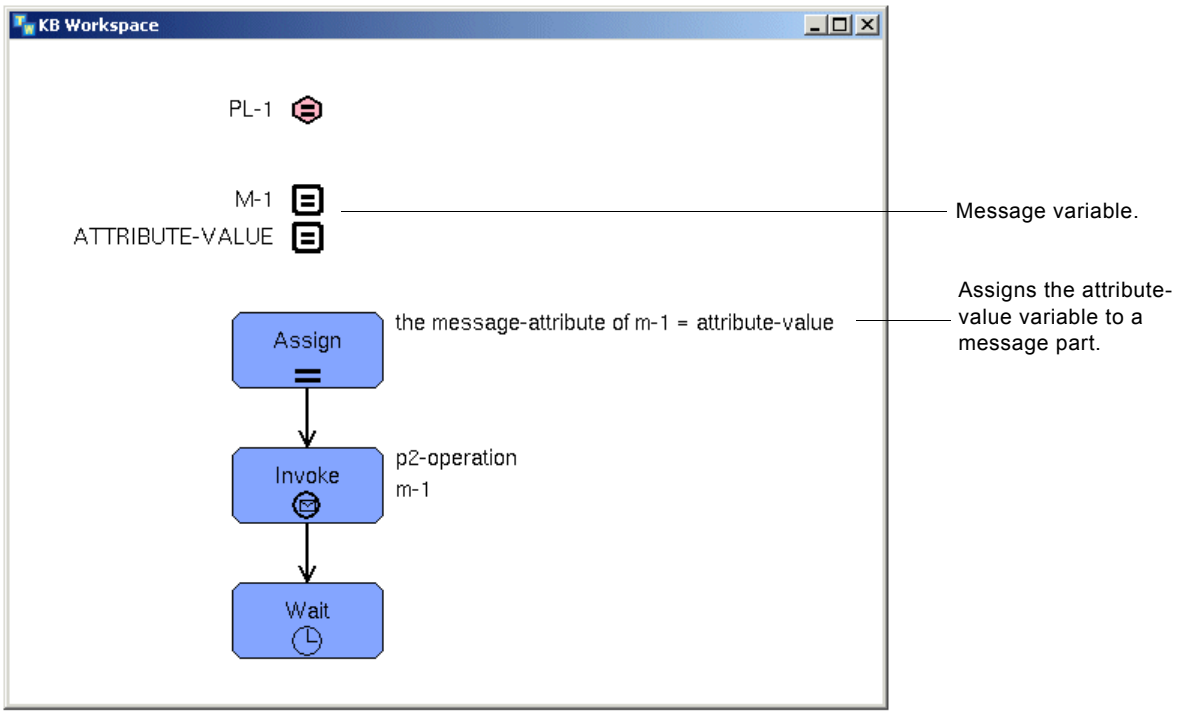
Working with Message Parts

A process flow might require setting a variable to the value of a message part or assigning values to message parts. You do this by configuring the g2gl-assignments attribute of the Assign activity to be a message part assignment statement.

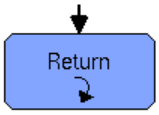
For details, see Message Part Assignment Statements.

For example, a credit rating process might set the credit-rating variable to the value of the credit-rating attribute of the credit-report message variable. The process might also assign a value to the credit-rating attribute of the credit-report message variable.

This example shows how to assign the value of a variable to a message part. The Assign activity assigns the value of the attribute-value variable to the message-attribute of m-1.



Returning Values



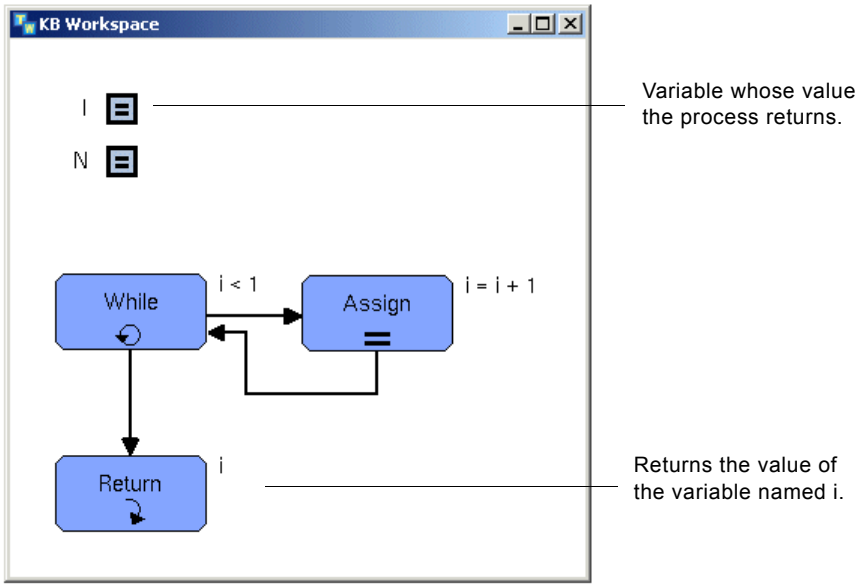
A process can return one or more values when it is finished executing by including a Return activity. To specify the values to return, you configure the g2gl-values-expression to be one or more G2GL expressions, separated by commas:

g2gl-expression[, *g2gl-expression*]

The Return activity has one input connection and can have one output connection, for example, to connect to a Switch-Join or Flow-Sync.

To obtain the return values, you call the g2-call-g2gl-process-as-procedure system procedure. The number of return values that you get when you call the system procedure with return values will match the number of return values specified in the Return activity.

This example shows a simple process that increments the value of a local variable named i and returns the value of i:



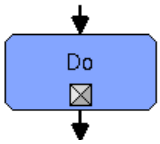
For information on calling a process as a procedure with return values, see Executing G2GL Processes.

Interacting with G2 Items

You use these activities to execute statements that interact with G2 items:

- Do activity — Concludes values for G2 items.
- Call activity — Calls a G2 procedure.

Concluding Values for G2 Items



You use the Do activity to conclude attribute values for G2 objects, just as you would by using the conclude action in a G2 procedure. To conclude values for G2 objects, you configure the g2gl-statements attribute of the Do activity include one or more statements of the form:

conclude that the *attribute-name* of *item* = *g2gl-expression*

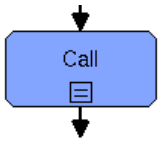
For example:

```
conclude that the my-attr-1 of my-object-1 = 27
```

Using the Do activity to conclude attributes of G2 items triggers whenever rules and forward chaining, and updates table attributes, just like using the G2 conclude action. You can use the conclude expression in a Do activity to conclude user-defined and system-defined attributes.

You can also use the Do activity to execute any type of assignment statement that you can with the Assign activity. For more information, see Assigning Values.

Calling G2 Procedures



You use the Call activity to call a G2 procedure or method from within a G2GL process, with or without arguments. If the procedure has return values, you can set the value of local variables within the G2GL process to the return values of the G2 procedure. To call a G2 procedure, you configure the g2gl-procedure-call-statement to be a statement of this format:

return-values-list = call *g2-procedure-name* (*arguments-list*)

where:

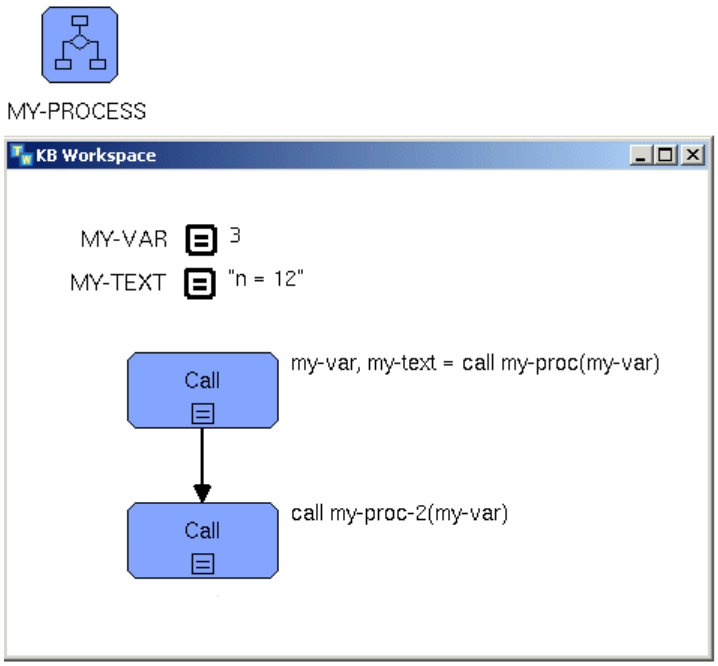
- *return-values-list* is an optional list of one or more return values for the procedure, separated by commas, where each return value is:
  - *variable-name*

    or
  - the *message-part* of *message-variable-name*
- *g2-procedure-name* is the name of a G2 procedure.
- *arguments-list* is an optional list of one or more G2GL expressions, which are arguments to the procedure, separated by commas. You can use an if-then-else statement in the argument list.

Here are some examples:

```
i, j = call my-proc (x)

the customer-name of customer-info, the id of customer-info = call my-proc (x)

a, b = call my-proc (if (i > j) then 12 else 13)
```

This example shows a process with two G2 procedure calls, using the Call activity. The first Call activity takes one argument and returns two values, and the second Call activity takes two arguments.



The g2gl-procedure-call-statement attributes for each Call activity are:

my-var, my-text = call my-proc (my-var)

call my-proc2 (my-var, my-text)

Here are the two procedures that the Call activities call. The my-proc procedure takes an integer as argument and returns an integer and a text. The my-proc2 procedure takes an integer and a text as arguments and returns no values.

MY-PROC

```
my-proc (N: integer) = (integer, text)
begin
   post "N is [n].";
   post "waiting for [n] seconds";
   wait for n;
   post "OK.";
   n = n + 1;
   return n, "NUMBER-[n]";
end
```

MY-PROC2

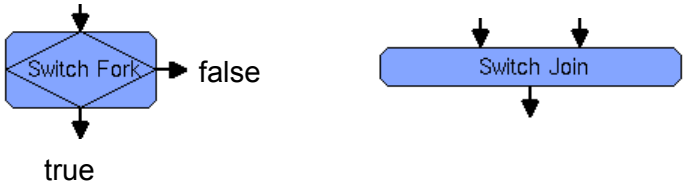
```
my-proc2 (N: integer, T: text)
begin
   post "N is [n]; T is [T]";
   post "Waiting for [N] seconds . . .";
   wait for N;
   post "OK";
end
```

Using Flow-Related Activities

A process flow can have various types of flow-related activities, which control the flow of execution within the process. These activities typically have multiple input and/or output connections. The flow-related activities are:

- Switch Fork and Switch Join activities support alternative branching.
- While activity provides iteration.
- Flow Split, Flow Sync, Flow Discriminator, Flow Terminator, and N-Out-Of-M Flow Join activities support concurrency branching with and without synchronization.
- Flow Signal and Flow Gate activities support concurrency branching with synchronization.
- Exit activity abruptly stops the flow of execution.

Switch Fork and Switch Join

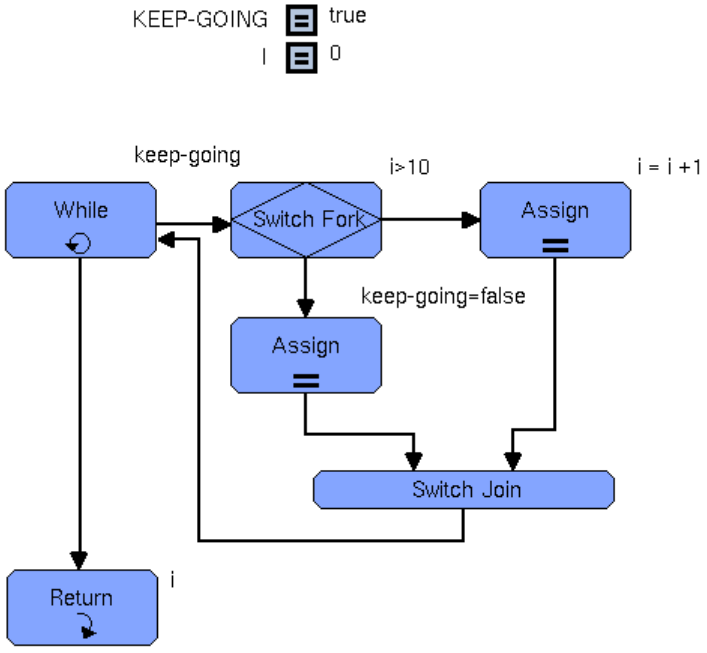


You use the Switch Fork activity to provide a two-way decision point in a flow. You configure the switch-fork-condition to be a G2GL expression that returns a truth value to determine which branch to take.

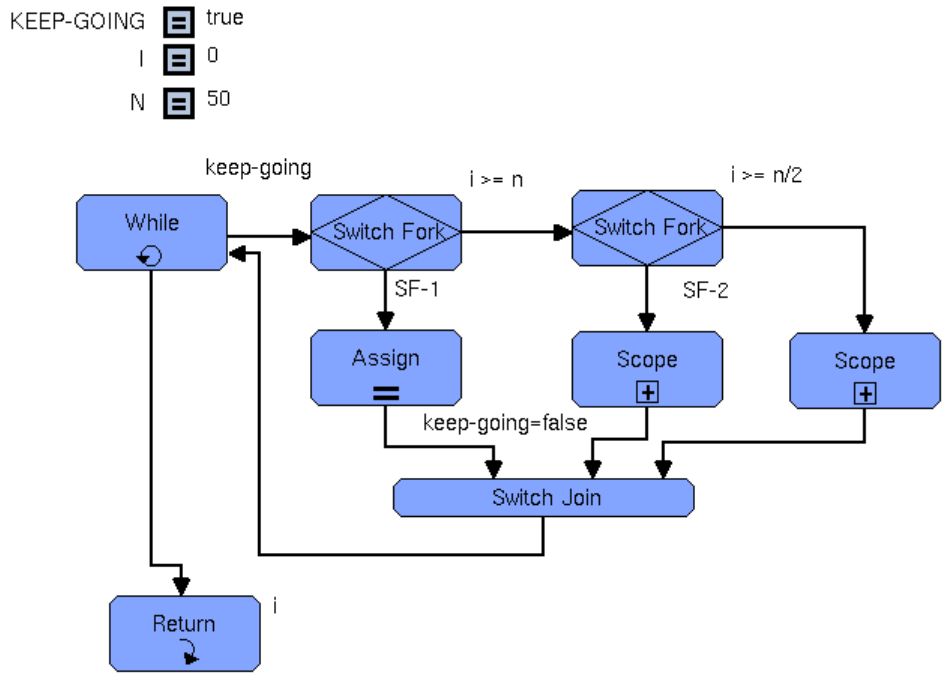
The activity has one input connection and two output connections. The bottom output connection is the true branch, and the right output connection is the false branch.

The Switch Join activity brings together the true and false branches of a Switch Fork activity. The activity has input connections from all the branches of a group of Switch Fork activities, but it has only a single output connection. The activity has no attributes to configure.

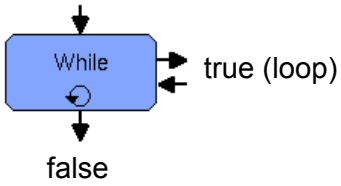
This example shows a simple process with a single Switch Fork and Switch Join activity. The Switch Fork tests the value of the variable i and increments its value by one on the false branch. The process loops until the variable exceeds 10, then it assigns the variable keep-going to false as the true branch.



This example has two Switch Fork activities and a single Switch Join activity, where the right (false) output connection of SF-1 goes to another Switch Fork activity. The left and middle input connections to the Switch Join activity are the true branches from each of the two Switch Fork activities, and the right input connection is the false branch of SF-2.



While



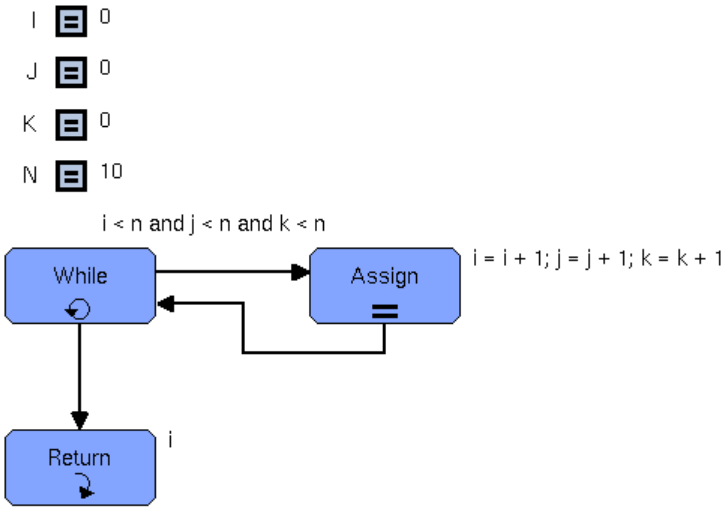
You use the While activity to perform iteration in a process. You configure the while-iteration-condition to be a G2GL expression that returns a truth value to determine whether to continue iterating. The expression might contain logical operators and if-then-else expressions, for example:

```
i < 10 and j < 100
```

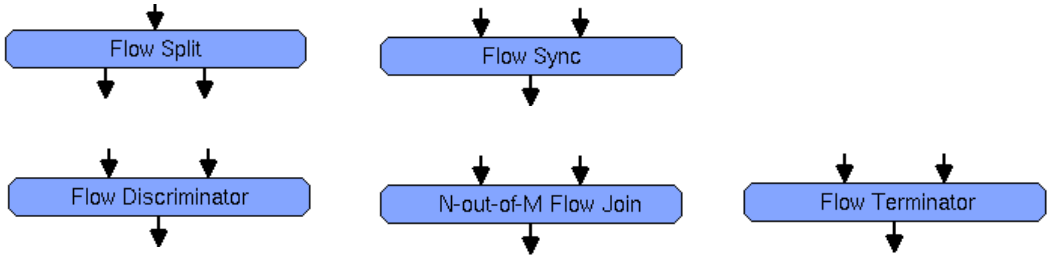
```
if (i < 10) then (j < 100) else (k < 100)
```

The activity can have one input connection on the top that comes from a preceding activity. The right output connection is the true branch, which must loop back into the activity on the right input connection. The bottom output connection, if any, goes to the next activity. As long as the iteration condition remains true, the process continues the loop. When the iteration condition becomes false, iteration is finished.

For example:



Flow Split, Flow Sync, Flow Discriminator, N-Out-Of-M Flow Join, and Flow Terminator



A process might require that the flow of execution splits so that separate threads can run concurrently before rejoining back into a single thread. You use the Flow Split activity to create multiple execution threads that run in parallel. A Flow Split activity can have any number of output connections.

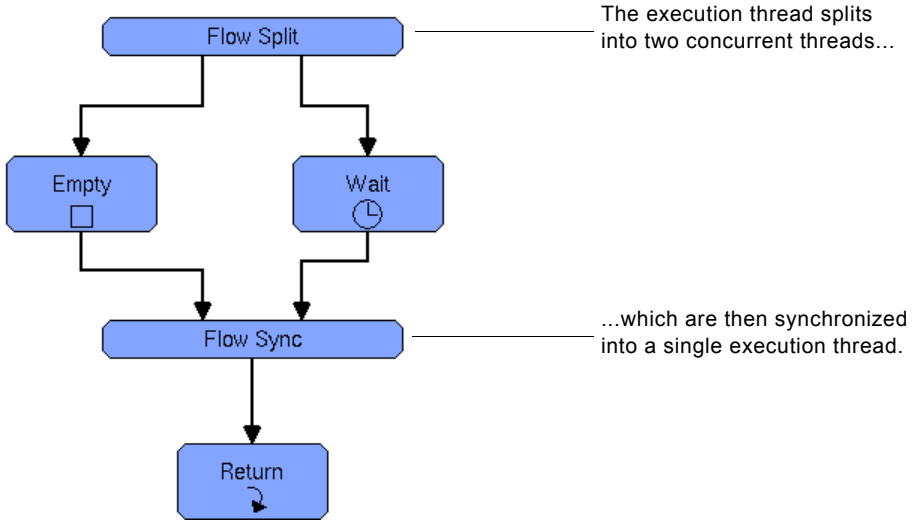
You must use one of these activities to rejoin concurrent threads that have been split, depending on when the following activity should execute:

- The Flow Sync activity synchronizes the execution of all concurrent threads. Process execution waits until all concurrent threads that had been split are rejoined before continuing with a single thread of execution.
- The Flow Discriminator activity merges the execution of multiple concurrent threads, without synchronizing. Process execution continues when one of the concurrent threads executes.
- The N-Out-Of-M Flow Join activity merges the execution of multiple concurrent threads and performs partial synchronization. Process execution continues when *n* out of *m* execution threads execute, where *n* is specified by the number-of-branches-to-synchronize attribute of the activity, and m is the number of branches coming into this activity.
- The Flow Terminator activity merges the execution of multiple concurrent threads and terminates all active, incoming execution threads when any one of the threads executes.

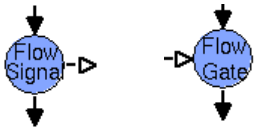
Note The Flow Discriminator and N-Out-Of-M Flow Join activities are not permitted within a While activity on the same body; however, they can occur within a scope body within a While activity.

Note If a process body, either at the top level or in a scope, has threads that are still executing when the overall final activity is reached on the main thread of execution, the execution of the body waits and does not end until all those threads have finished executing, that is, until they have reached their terminal flow join.

This example shows a Flow Split activity that splits a single execution thread into two separate threads, which rejoin at a Flow Sync activity:



Flow Signal and Flow Gate



A Flow Signal activity is the start of a synchronization link, which synchronizes two activities in a process. The Flow Signal connects to a Flow Gate on some other branch of the process flow, which is the end of the synchronization link.

Assuming no transition or join conditions are specified, when the Flow Signal executes, it sends a signal to the Flow Gate. A Flow Gate waits until it receives a signal from all connected Flow Signals before it lets the thread of execution proceed to the next activity.

The Flow Signal activity has an input connection from a source activity, and two output connections. The left or right output connection goes to a left or right input connection of a Flow Gate, and the bottom output connection goes to the next activity in the process.

The Flow Gate activity can have any number of input connections and has one output connection. The top input connection is from a source activity. The left or right input connections come from the left or right output connection of Flow Signal activities. The bottom output connection goes to the next activity.

When a Flow Signal activity executes, it determines its status before sending the signal. The status is either positive or negative. You can configure the flow-signal-transition-condition attribute to be a G2GL expression that returns a truth-value to

determine the signal status. If the expression evaluates to true or if the attribute is unspecified, the status of the Flow Signal is positive; otherwise, its status is negative.

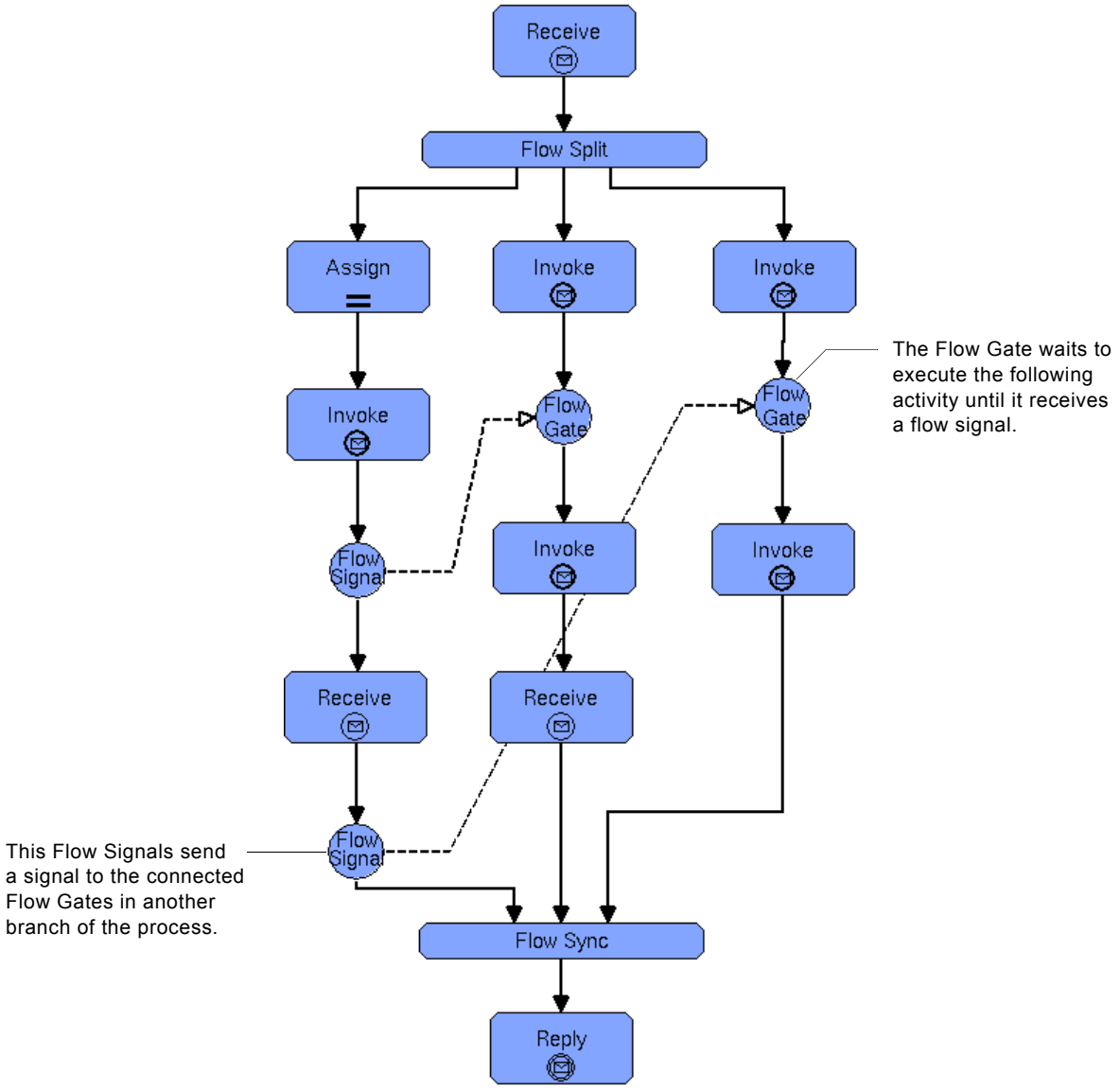
When a Flow Gate has received a signal from all connected Flow Signal activities, it checks the g2gl-join-condition, which is a G2GL expression that returns a truth-value. If the expression evaluates to true, then the next activity is executed; otherwise, the next activity is not executed. If the g2gl-join-condition is unspecified, the Flow Gate checks the status values of the signals. If at least one status value is positive, then the next activity is executed; otherwise, the next activity is not executed.

By default, if the g2gl-join-condition expression evaluates to false, or if the g2gl-join-condition is not specified and all the status values of the connected Flow Signal activities are negative, a join-failure fault is thrown. You can also set the suppress-join-failure attribute of the Flow Gate to yes, in which case the join-failure fault is not thrown and instead execution continues with the activity that follows the activity connected to the Flow Gate.

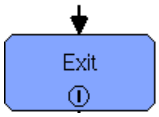
For information on catching faults, see Handling Faults.

The g2gl-link-name attribute of a Flow Signal is not currently supported.

This figure shows two pairs of Flow Signal and Flow Gate activities. The left branch handles shipping requests, the middle branch handles invoicing, and the right branch handles scheduling. The Flow Signal activities in the shipping thread send signals to the connected Flow Gates, which wait to execute the following activities until they receive the signals.



Exit



If the process needs to terminate abruptly at any point, you can use the Exit activity. You can place the Exit activity anywhere within the process, including within a Scope activity of arbitrary depth to exit at that point, for example, on one of several branches.

Although you can place the Exit activity at the bottom of the flow of execution in a process body to exit without returning any values, this technique is not recommended or necessary.

The Exit activity has one input connection. It has no attributes to configure.

Defining Scopes and Handlers

In G2GL, you can have **scope** activities, which have bodies that specify subprocesses. You can also have scope-like fault, alarm event, message event, or compensation handlers.

G2GL provides several types of scopes, which are all subclasses of g2gl-activity-with-body:

- Scope — An activity with a body that defines a subprocess within a higher-level process.
- Fault Handler — A handler for named faults, which catches system-defined faults or user-defined faults that a Throw activity throws.
- Alarm Event Handler — A handler for alarms, which executes based on a duration or deadline expression.
- Message Event Handler — A handler for message events, which an Invoke activity sends and a Receive activity receives.
- Compensation Handler — Specifies how to compensate for (for example, undo) the work done by a Scope in case a fault is signaled in a parent Scope after the Scope has completed successfully.

You can reference local variables that are defined within the scope or that are defined in the higher-level process that defines the scope. Local variable values defined within the scope override local variable values defined in the higher-level process.

For more information about Message Event Handlers, see Handling Message Events.

Defining Scope Activities

You use a Scope activity to define a subprocess within the overall process. Activities within the Scope activity can refer to local variables within the subprocess or in the process that defines the Scope activity. Local variables within the Scope body override local variables in the high-level process.

To define a scope within a process:

1 Do one of the following:

   ➔ Choose KB Workspace > New Object > G2GL-object > G2GL-activity-with-body > G2GL-scope.

   or

   ➔ With `bpms.kb` loaded, choose View > Toolbox - BPMS, display the General palette, and create a Scope activity.

   Place it within the body of a G2GL process.

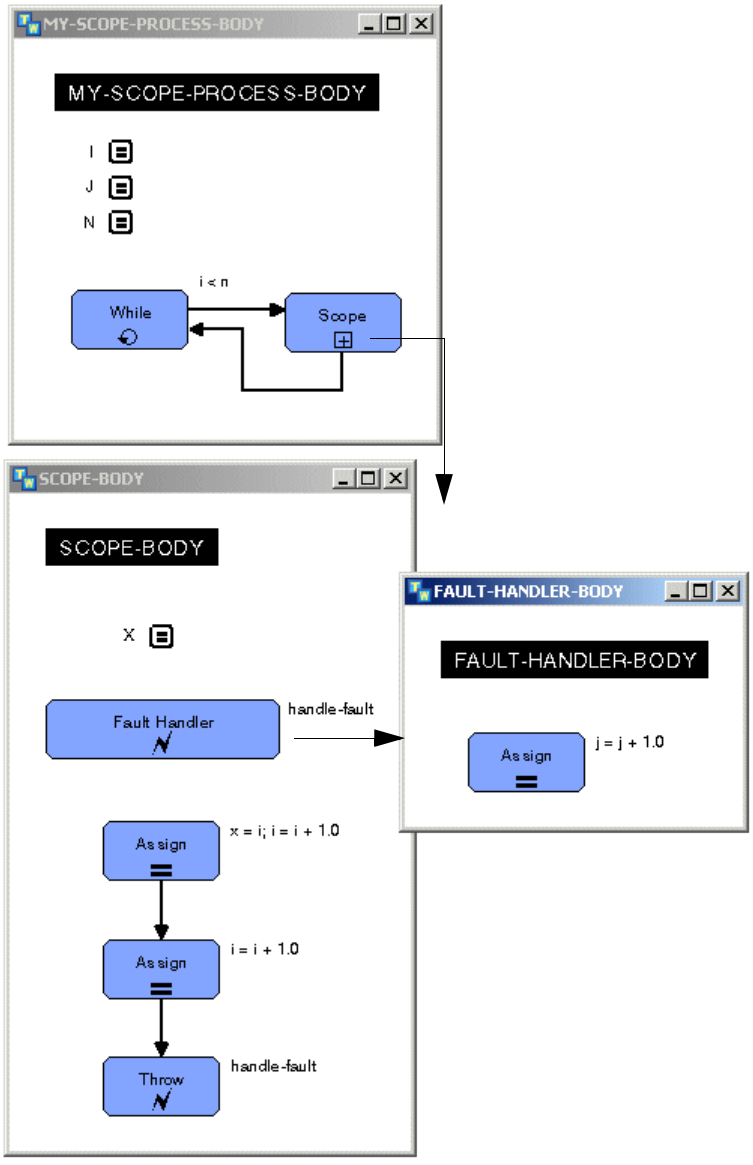
2 Connect the Scope activity to other activities in the process.

3 Create a subworkspace for the Scope activity and configure the body as needed to define the local scope.

The activities in the body of the Scope can refer to local variables in the Scope body or in the superior process.

Note The variable-access-serializable attribute is not implemented in this release.

Here is a process that defines a Scope. The high-level process defines three local variables, i, j, and k. The Scope body assigns local variables defined in the body, as well as in the superior process. The Scope defines a Fault Handler, which assigns values to variables defined in the superior process.



For information on the Fault Handler, see Handling Faults.

Handling Faults

You use a Fault Handler to handle certain kinds of faults that might arise during execution of a process, typically by undoing the actions of an incomplete and unsuccessful execution of a process. When a fault is signaled during execution, the execution thread is terminated, and if a matching fault handler is specified, the specified fault handler executes to handle the fault.

A Fault Handler can handle user-defined faults, which a process throws by using the Throw activity. The Throw activity specifies a fault name and fault data, which are used to match against a Fault Handler that should catch the fault. The Fault Handler can have a g2gl-arg-variable on its body. The Fault Handler only handles faults that have fault data whose type matches the type of the g2gl-arg-variable. The value of this g2gl-arg-variable is copied from the g2gl-variable specified in the fault-data-g2gl-variable-name attribute of the Throw activity.

A Fault Handler can also handle system-defined faults. An example of a system-defined fault is join-failure. System-defined faults never have fault data. To catch all faults, use catch all.

Note The system-defined faults are formatted with hyphens in G2GL, whereas in BPEL they are formatted similar to this: `joinFailure`.

For processes that provide communication, you can connect an Invoke activity to a Fault Handler, in which case the Fault Handler is local to the Invoke activity. The activities on the body of the Fault Handler are executed when a fault occurs while invoking the named operation.

Activities within the Fault Handler can refer to local variables within the subprocess or in the process that defines the Fault Handler.

To handle faults in a process:

1 Do one of the following:

   ➔ Choose KB Workspace > New Object > G2GL-object > G2GL-activity-with-body > G2GL-handler > G2GL-fault-handler.

   or

   ➔ With `bpms.kb` loaded, choose View > Toolbox - BPMS, display the General palette, and create a Fault Handler activity.

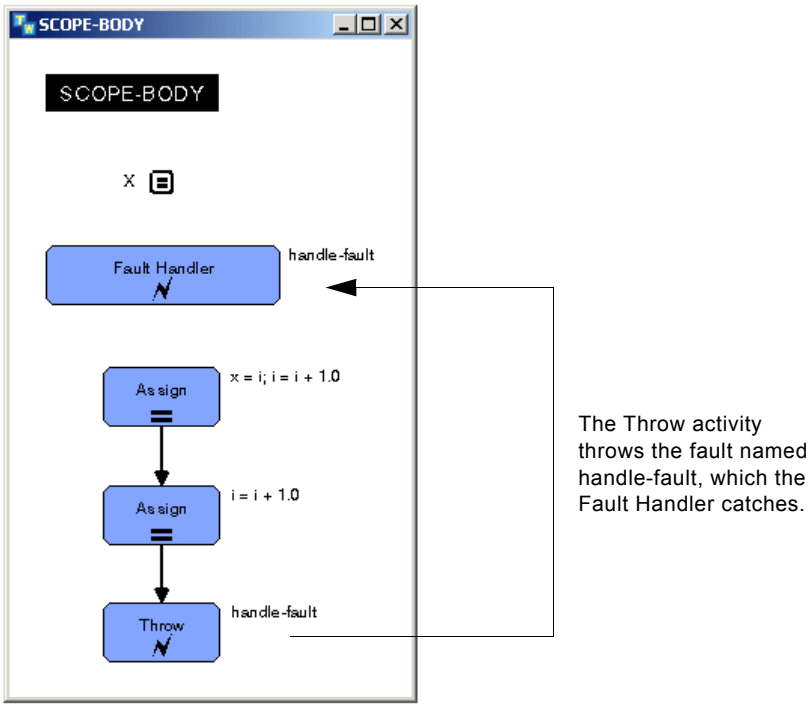
   Place it at the top of the body of a G2GL process.

2 To provide a local Fault Handler for an Invoke activity, choose add stub for local handler on the Fault Handler to create a left-side input stub and connect it to the right side of an Invoke activity.

3 Configure the g2gl-fault-name in the Fault Handler to be a system-defined fault or any user-defined fault, for example, handle-fault.

4 Create a subworkspace for the Fault Handler and configure the body of the handler, as needed to handle the fault.

Here is a Fault Handler that catches a fault named handle-fault, which the Throw activity throws.



Handling Alarm Events

An Alarm Event Handler is similar to a Message Event Handler except that it executes based on a duration or deadline expression, similar to a Wait activity.

For more information on duration and deadline expressions, see Wait.

For more information on Message Event Handlers, see Handling Message Events.

To handle alarm events in a process:

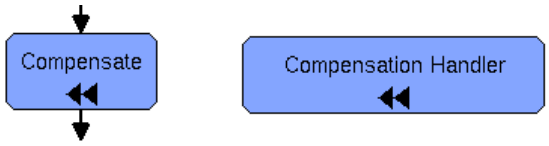
1 Choose KB Workspace > New Object > G2GL-object > G2GL-activity-with-body > G2GL-handler > G2GL-event-handler > G2GL-alarm-event-handler and place it at the top of the body of a G2GL process.

2 Configure the type-of-g2gl-alarm-time-expression in the Alarm Event Handler to be either duration-expression or deadline-expression.

3 Configure the duration-or-deadline-expression to be a *time-expression* that represents the specified type of expression, either a duration or a deadline.

4 Create a subworkspace for the Alarm Event Handler and configure the body of the handler, as needed to handle the alarm.

Compensating for Faults



The compensation mechanism in G2GL supports transaction-based computing. A Compensation Handler on a Scope body specifies how to compensate for (for example, undo) the work done by that Scope in case a fault is signaled in a parent Scope after the Scope has completed successfully. A Scope completes successfully if it finishes executing without signaling a fault. Even if a signaled fault is handled by a Fault Handler in that Scope, the Scope is not considered to have completed successfully.

When a Compensation Handler is invoked, the activities on its body are executed. All variables in the handler's Scope and its ancestor Scopes are temporarily restored to their values at the time when the Scope completed successfully.

A Compensation Handler can be invoked in one of two ways:

- Explicitly by a Compensate activity in a Fault Handler or Compensation Handler in the parent Scope.
- Implicitly if there is no applicable Fault Handler or no Compensation Handler in the parent Scope.

By default, a Compensate activity in a Fault Handler or Compensation Handler body invokes the Compensation Handlers for all Scopes that completed successfully in the same body as the handler, in reverse order of their completion. Alternatively, you can configure the scope-name-for-compensate-activity to be the name of a specific Scope in the same body as the handler; the Compensate activity will then only invoke the Compensation Handler on the body of that Scope.

If the named Scope did not complete successfully or if its Compensation Handler was already invoked, then its Compensation Handler is not invoked, and the Compensate activity does nothing.

If the named Scope completed successfully multiple times because it was inside a While loop, its Compensation Handler is invoked once for each completion, in reverse order.

If there is no applicable Fault Handler in a Scope when a fault is signaled, all Compensation Handlers in Scopes that have completed successfully in that Scope are invoked, in reverse order of their completion, before the fault is propagated to the parent Scope. If there is no Compensation Handler in a Scope that is being

compensated, all Compensation Handlers in Scopes that have completed successfully in that Scope are invoked, in reverse order of completion. If a Fault Handler or Compensation Handler has no Compensate activities, then the Scopes on the same body as the handler will not be compensated. In other words, implicit compensation only happens if there is no applicable Fault Handler when a fault is signaled or no Compensation Handler when a Scope is compensated.

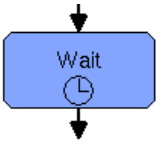
An Invoke activity may be connected to a Compensation Handler with a right-side output connection. This is equivalent to the Invoke activity being inside its own Scope with the Compensation Handler being on that Scope body.

Handling Message Events

You use Message Event Handlers with processes that provide communication. For more information, see Handling Message Events.

Miscellaneous Activities

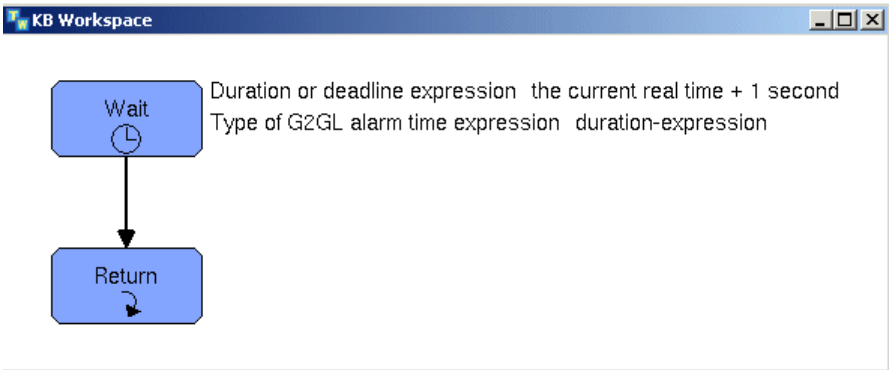
Wait



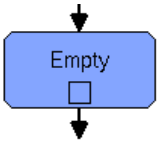
You use the Wait activity to provide a duration or a deadline expression that causes the process to wait for a period of time. You configure the type-of-g2gl-expression-in-wait-activity to be either duration-expression or deadline-expression, and the duration-or-deadline-expression to be a *time-expression*.

A duration expression is the amount of time to wait before proceeding to the next activity. A deadline expression is an absolute time at which to proceed to the next activity, which you can express in terms of the current real time.

For example, this example waits for 1 second:



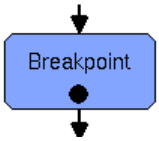
Empty



The Empty activity performs no action.

Debugging

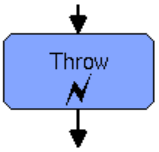
Breakpoint



You use the Breakpoint activity to add a permanent breakpoint to a process for debugging. The flow stops at the Breakpoint activity and shows an individual execution display with a breakpoint at the activity. To continue, click the breakpoint.

For more information about working with individual execution displays and continuing from breakpoints, see Debugging G2GL Processes.

Throw



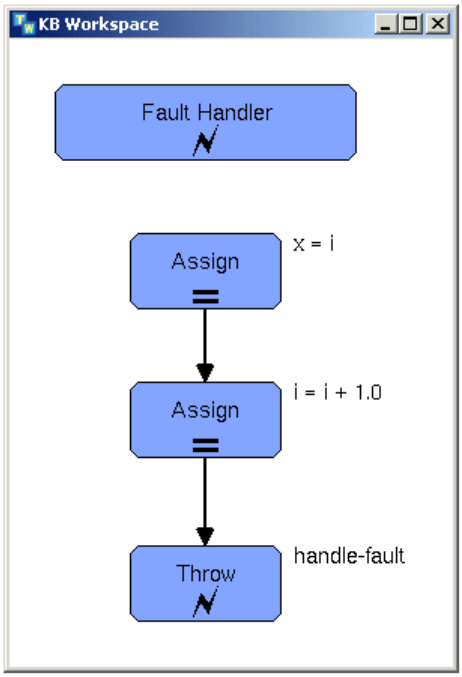
You use the Throw activity to signal a named fault. You configure the fault-name-for-throw to be the named fault to throw. Optionally, you configure the fault-data-g2gl-variable-name to be the name of a g2gl-variable whose value is to be used as fault data.

If break-on-execution-fault is true for the individual process or if break-on-all-execution-faults is true in the G2GL Parameters system table, an individual execution display appears with a breakpoint at the Throw activity.

You can configure a Fault Handler to catch the named fault, as needed. The Fault Handler can have a g2gl-arg-variable on its body. The value of this g2gl-arg-variable is copied from the g2gl-variable specified in the fault-data-g2gl-variable-name activity of the Throw activity.

For more information about fault handlers, see Handling Faults.

This example throws a fault named handle-fault:



Summary of Differences Between G2GL and BPEL Activities

This table summarizes which G2GL activities are standard BPEL, which are G2GL extensions to BPEL, and which have somewhat different behavior than standard BPEL activities.

G2GL Activity	Description
Assign	Standard BPEL
Return	G2GL extension
Do	G2GL extension
Call	G2GL extension
Switch Fork/ Switch Join	To implement a BPEL switch activity might require several Switch Fork activities.
While	Standard BPEL
Wait	Standard BPEL

G2GL Activity	Description
Flow Split/ Flow Sync	G2GL provides separate activities for splitting and synchronizing the flow of execution, whereas BPEL specifies a single flow activity, which contains the concurrent activities as subactivities.
Flow Discriminator/ N-out-of-M Flow Join/ Flow Terminator	G2GL extension.
Flow Signal/ Flow Gate	G2GL breaks out the Flow Signal and Flow Gate activities as separate activities, whereas BPEL specifies these as properties of arbitrary other activities.
Empty	Standard BPEL
Exit	Standard BPEL Terminate activity.
Breakpoint	G2GL extension.
Throw	Standard BPEL
Compensate	Standard BPEL

Communicating Between G2GL Processes

G2GL provides communication between two linked partner processes via a **partner link**, which is a connection between two partner processes. A partner is a series of connected elements that mediate communication between two linked partners.

Partners communicate at the most basic level through **message transmissions**, which are combinations of named operations and messages. In general, a message transmission involves one partner that invokes the operation, and another partner that receives the message transmission and possibly replies to the partner that invoked the operation. You can think of a message transmission as an RPC call, where the message that one partner sends and the other partner receives is the argument to the RPC call.

The partner that invokes the operation creates a message and assigns it to a local variable. The partner that receives the message transmission assigns a local variable to the received message.

You represent partner links in a process as **partner link variables**, which are specialized process variables that can get bound to partner links. When a partner link variable is bound, its value is one of the two end elements of the partner link,

which not only represents the partner link as a whole, but also identifies which end of it belongs to a particular process.

For example, in a purchase order fulfillment process, two partners might be linked via a *purchasing* partner link. To establish communication, you instantiate the partner process that invokes the *process-purchase-order* operation with the *purchase-order* message transmission. The partner link is established when the partner process receives the purchase order message for the invoked operation, which instantiates the linked partner process.

Invocation

At the heart of all communication is the Invoke activity, which invokes a named operation with a message across an established or newly created partner link. Typically, you use the Invoke activity to provide two-way synchronous communication by invoking an operation with a message and waiting for a response as part of the same activity. The Invoke activity specifies the partner link variable, the named operation, the message to send, and the message variable in which to receive the response message.

You use the Receive activity to receive messages that the Invoke activity sends across a partner link. The Receive activity identifies the message to receive by referring to the named operation sent by the Invoke activity and the partner link variable. It also specifies a message variable in which to receive the sent message.

You use the Reply activity to send a response by specifying the partner link variable, the named operation sent by the Invoke activity, and the message variable whose value is set to the message to send in response.

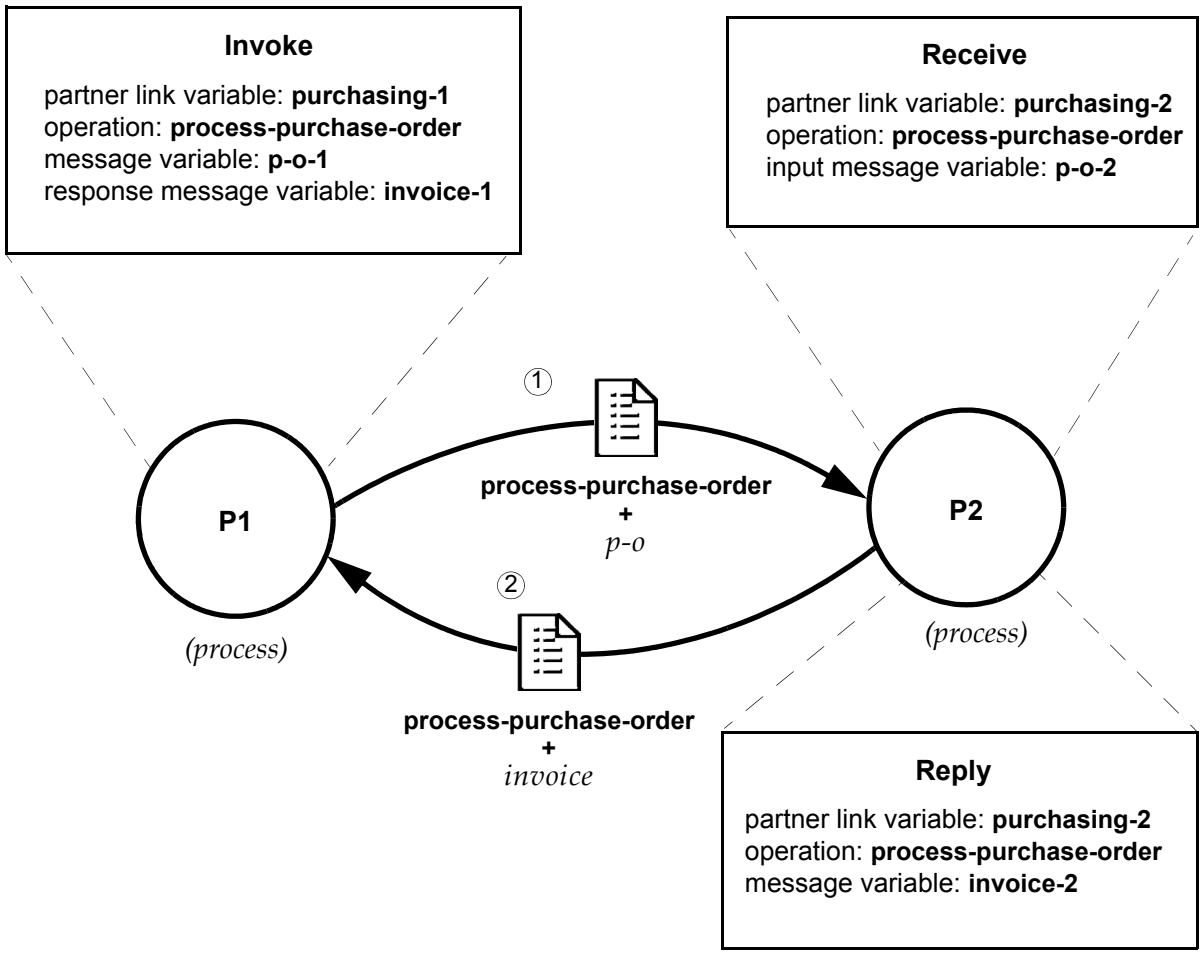
You can also use the Pick activity and the Message Handler activity to receive messages that the Invoke activity sends. These activities use the received message in different ways.

The Receive activity and the Pick activity are both **instantiation triggers**, which means if they are the first activity in the flow chart, they can trigger the instantiation of the process. Invoking the named operation for an instantiation trigger establishes a partner link between two partner processes.

This table describes the required activities and specifications for two-way synchronous communication:

Activity	Specification	Description
Invoke	partner link variable name operation name message variable name response message variable	A partner invokes a named operation with a given message and partner link to send the message, then waits for a response as part of the same operation. It sets the value of the response message variable to the message received in response.
Receive	partner link variable name operation name message variable name	A partner receives a message that the Invoke activity sends. The Receive activity identifies the message to receive by the named operation that sent the message and the partner link. It sets the value of the specified message variable to the received message.
Reply	partner link variable name operation name message variable name	A partner sends a response message to a message that the Invoke activity sends and the Receive activity receives. The Reply activity identifies the message to send in response by the named operation that sent the message and the partner link. The specified message variable must be set to the message to send in response.

The following figure shows two-way communication between two partners that participate in the *purchasing* partner link. P1 invokes the *process-purchase-order* operation with the *p-o* message, which sends the message to the linked partner. P2 receives the *p-o* message that P1 sends. P2 then sends an *invoice* message in response as part of the same operation. You specify the partner link variable, the named operation, and the message variable for the Invoke, Receive, and Reply activities, and you specify the response message variable for the Invoke activity.



BPEL Compliance

To simplify the specification of processes that communicate, G2GL only requires that you specify the partner link variable, without declaring its type, the named operation to invoke, and the message to send or receive. G2GL does not require partner link type definitions to invoke named operations with message types across a partner link.

In a future release, G2GL will support the BPEL-compliant specification for communication between two G2 processes, including partner link type definitions, port type definitions, roles, correlation variables, and correlation sets.

To configure a process to provide communication by using the simplified G2GL specification, configure the following attributes:

Attribute	Object	Description
suppress-unspecified-partner-link-variable-type-faults	G2GL Parameters system table	Set this value to yes to suppress faults when partner link variables do not specify a g2gl-variable-type.
names-of-g2gl-service-switches-for-instantiation	G2GL process	Use the default value, which is none, to use the simplified G2GL method of communicating between processes.
name-of-g2gl-service-switch-for-connection	G2GL process	Use the default value, which is g2gl-standard-service-switch, to use the simplified G2GL method of communicating between processes.
g2gl-variable-type	Partner link variable	Set this value to unspecified to provide communication without requiring a partner link variable type definition.
g2gl-port-type	Invoke, Receive, and Reply activities	Set this value to unspecified to provide communication without requiring a port type definition.
g2gl-correlations	Invoke, Receive, Reply, and Message Event activities	Set this value to none to provide communication without requiring correlation variables.

Creating Processes that Communicate

To create two-way communication, you use these types of objects:

- **Partner link variables** — Get bound to a representation of the partner link that links the two partners.
- **Message variables** — Local variables that are set to messages, which represent the messages to send and/or messages received.
- **Invoke activity** — Invokes a named operation with the value of a message variable, response message variable, and partner link variable.
- **Receive activity** — Receives messages sent by an Invoke activity.
- **Reply activity** —Sends responses to an Invoke activity with a given named operation and message, along a partner link.

Corresponding activities in each side of a partner link must use the same operation name. To provide communication, one process invokes operations that send messages, and the other process receives and optionally responds to received messages. Thus, one process specifies the Invoke activity, and the other process specifies the Receive activity, and optionally, the Reply activity.

The following sections describe how to create a process that provides two-way communication. One-way communication is similar but only requires the Invoke and Receive activities.

Creating Partner Link Variables

A partner link variable gets bound to one end of a partner link. Activities within each linked process refer to the partner link variable in their specification.

In G2GL, partner link variables do not require a type declaration to establish a partner link. For more information, see BPEL Compliance.

To create a partner link variable:

1 Create a g2gl-process and show its subworkspace.

For more information, see Creating a G2GL Process.

2 Do one of the following:

➔ Choose KB Workspace > New Object > G2GL-object > G2GL-partner-link-variable.

or

➔ With `bpms.kb` loaded, choose View > Toolbox - BPMS, display the Variables palette, and create a partner link variable.

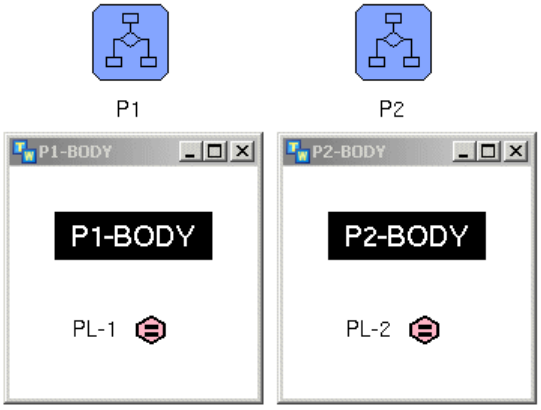
Place the partner link variable at the top of the subworkspace of the G2GL process.

3 Configure the names attribute to describe the partner link, for example, pl-1.

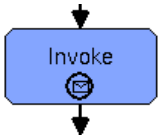
4 Create another g2gl-process with which the first process should communicate.

5 Create and configure another partner link variable for the linked process, for example, pl-2.

For example, here are two processes named p1 and p2 with partner link variables named pl-1 and pl-2:



Invoking an Operation that Sends a Message



To invoke an operation, you use the Invoke activity. For one-way communication, you specify the partner link variable, the operation name, and the message variable that is the message to send. You can think of the message variable as the argument to the named operation.

The Invoke activity does not require a port type or correlations to invoke operations. For more information, see BPEL Compliance.

To invoke an operation that sends a message:

1 Create a g2gl-invoke activity, place it on the body of a G2GL process.

2 Configure the g2gl-partner-link-variable-name of the Invoke activity to be a partner link variable, for example, pl-1.

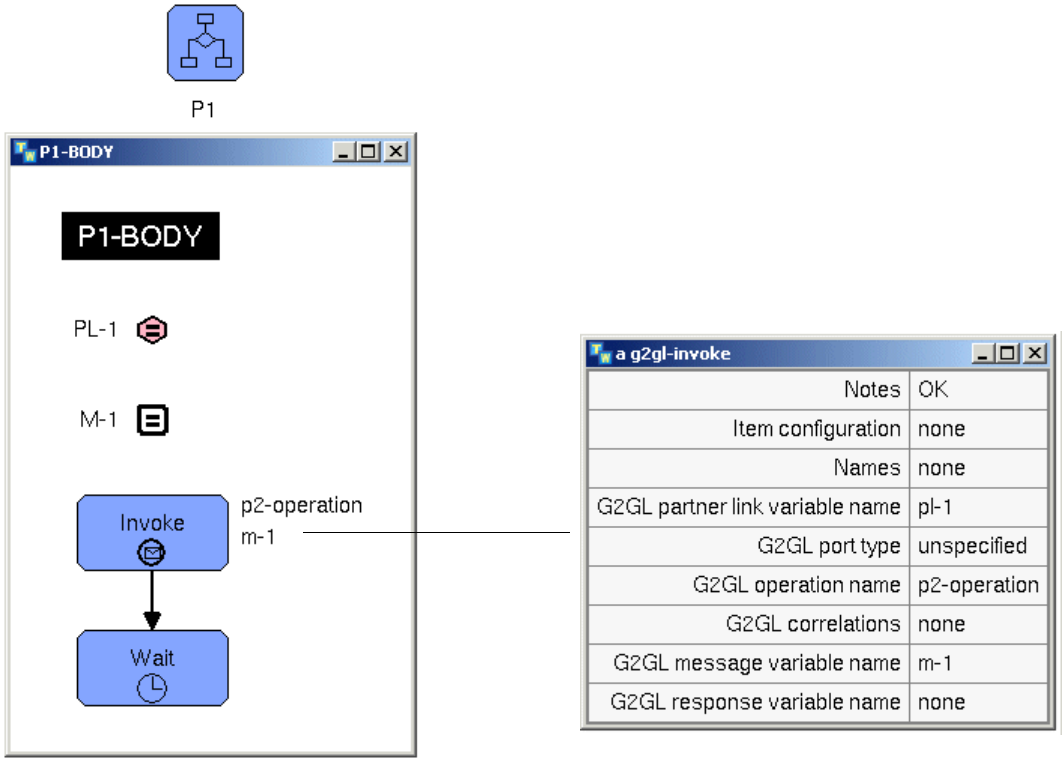
3 Configure the g2gl-operation-name to be any named operation to invoke, for example, p2-operation.

4 Configure the g2gl-message-variable-name to refer to a message variable whose value is set to the message to send when the operation is invoked, for example, m-1.

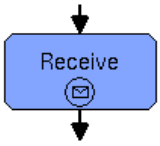
5 Configure the g2gl-port-type as unspecified.

For more information, see BPEL Compliance.

This example shows the body of the p1 process and the table for the Invoke activity. The Invoke activity invokes the operation named p2-operation with the m-1 message variable across the pl-1 partner link. The process then waits.



Receiving a Message that an Invoke Activity Sends



To receive a message that an Invoke activity sends, you use the Receive activity. You specify the partner link variable, the operation name, and the message variable that should be set to the message to receive.

The Receive activity can be an instantiation trigger. If the Receive activity is the first activity in the process, it must be an instantiation trigger.

The Receive activity does not require a port type or correlations to receive messages. For more information, see BPEL Compliance.

To receive a message that an Invoke activity sends:

1 Create and configure a g2gl-invoke activity to send a message.

For details, see Invoking an Operation that Sends a Message.

2 Create a g2gl-receive activity and place it on the body of a G2GL process that should receive the message.

The Receive activity can be in a different process from the process that sends the message.

3 Connect the input and output connections to other activities in the process.

The Receive activity can initiate the process, in which case it has no input connections.

4 Configure the g2gl-partner-link-variable-name of the Receive activity to be a partner link variable, for example, pl-2.

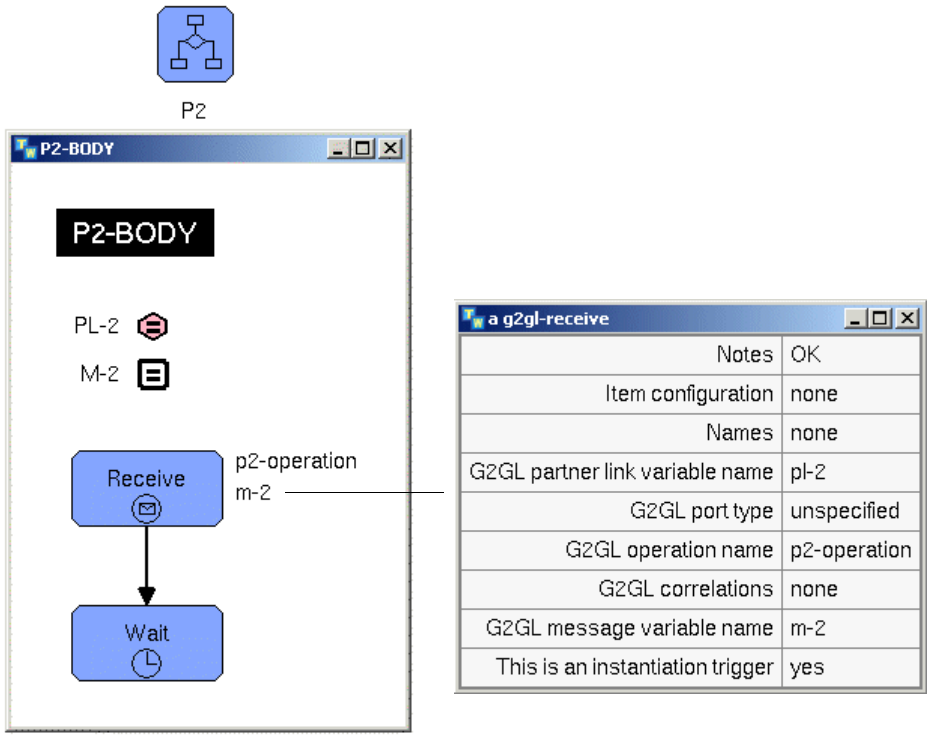
5 Configure the g2gl-operation-name to be the named operation that the Invoke activity sends, for example, p2-operation.

6 Configure the g2gl-message-variable-name to refer to a message variable whose value should be set to the message to receive.

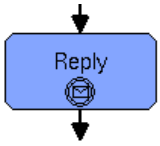
7 Configure the this-is-an-instantiation-trigger attribute of the Receive activity to be yes.

This attribute causes the process to be instantiated when the Receive activity receives a message via a named operation.

This example shows the body of a process named p2 and the table for the Receive activity. The Receive activity receives a message that the Invoke activity sends by invoking the operation named p2-operation across the pl-2 partner link. The value of the local variable named m-2 gets set to the message received. The process then waits.



Replying to a Message that a Receive Activity Receives



To create two-way synchronous communication that sends and receives a message as part of the same activity, you use the Invoke activity. You specify the partner link variable, the operation name, the message variable that is the message to send, and the message variable that gets set to the response message. The Invoke activity waits until it receives a response before continuing execution.

The partner process that receives the message also includes a Reply activity, which specifies the message variable to send in response. It also specifies the same partner link variable and operation name as the Receive activity.

You can use the same message variable for the Receive activity and the Reply activity, in which case the message that is received is also the response message.

You can also create a new response message by using the Assign activity to assign a different message variable to a new message, and specify that message variable in the Reply activity.

The Reply activity does not require a port type or correlations to send response messages. For more information, see BPEL Compliance.

To reply to a message that a Receive activity receives :

1. Create and configure a g2gl-invoke and g2gl-receive activity to send a message and receive the sent message.

   For details, see:

   - Invoking an Operation that Sends a Message.
   - Receiving a Message that an Invoke Activity Sends.

2. Create a g2gl-reply activity, place it on the body of the G2GL process that receives a message that the Invoke activity sends, and connect it *after* the Receive activity that receives the message.

   In the example, you would place the Reply activity in the p2 process.

   The Reply activity can be the last activity in a process, in which case it has no downstream activities.

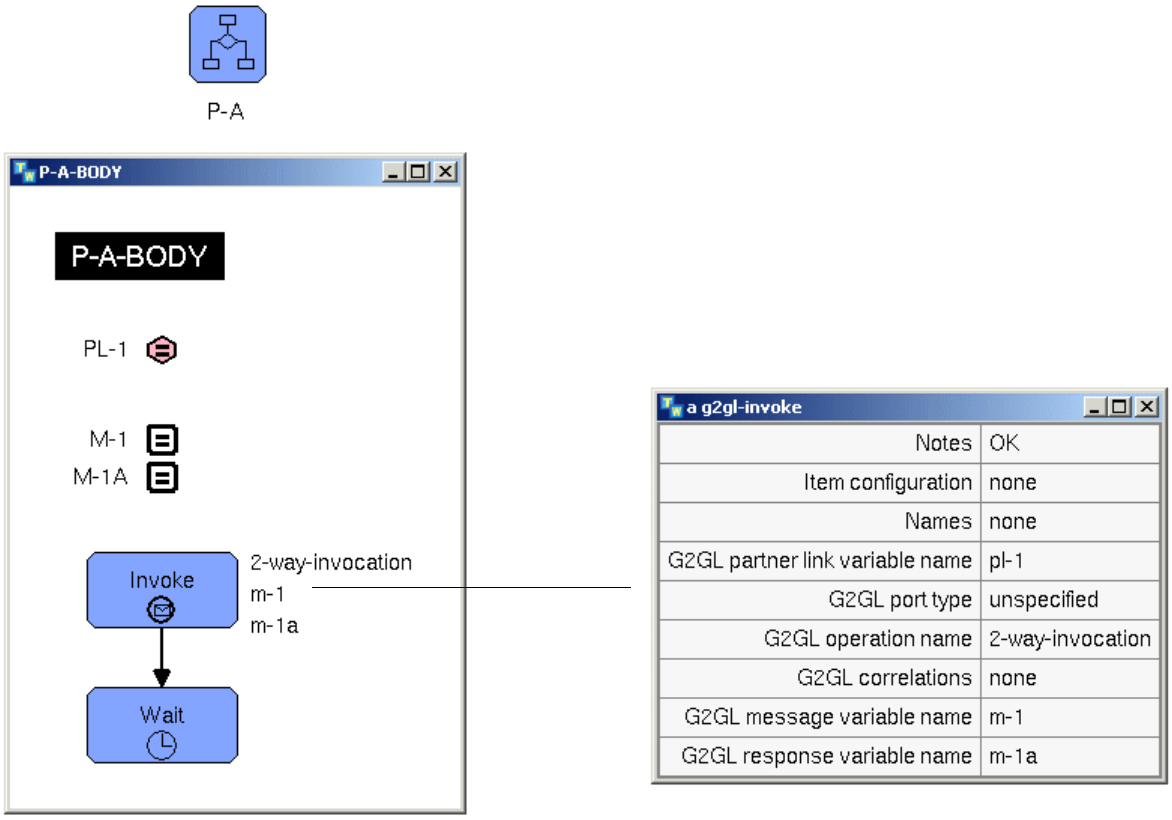
3. Configure the g2gl-partner-link-variable-name of the Reply activity to be the same partner link variable that the Receive activity uses to receive a message, for example, pl-2.

4. Configure the g2gl-operation-name of the Reply activity to be the named operation that the Invoke activity uses to send a message, which the Receive activity receives, for example, p2-operation.

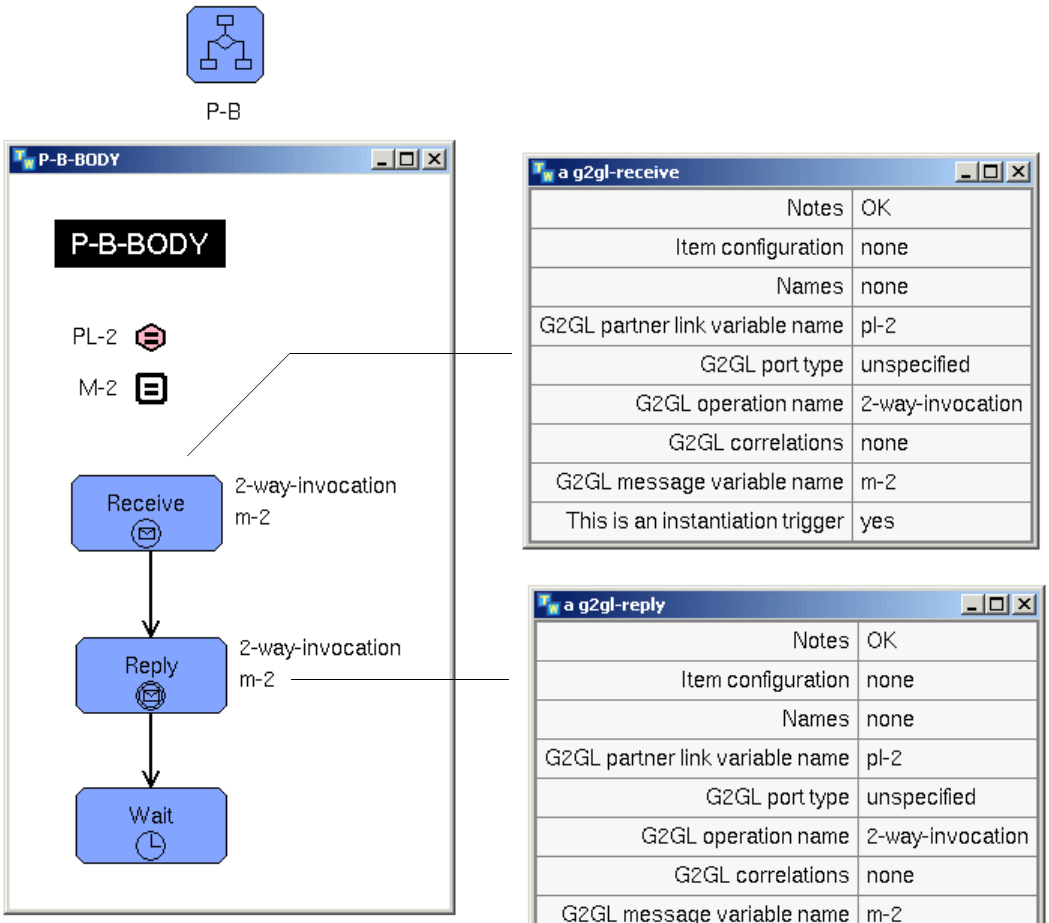
5. Configure the g2gl-message-variable-name of the Reply activity to refer to the message variable to set to the response message.

You can use the same message variable for the Receive activity and the Reply activity, in which case the Reply activity sends the same message that was received as the response message. You can also send a different message.

This example shows the body of a process named p-a and the table for the Invoke activity. The Invoke activity invokes the operation named 2-way-invocation across the pl-1 partner link. It specifies m-1 as the message variable to send and m-1a as the message variable to send in response. The process then waits.



Here is the body of a process named p-b and the tables for the Receive and Reply activities. The Receive activity receives a message that the Invoke activity sends by invoking the operation named 2-way-invocation across the pl-2 partner link. The Receive activity specifies m-2 as the message to receive. The Reply activity replies to the message that the Invoke activity sends by invoking the operation named p2-operation across the pl-2 partner link. The Reply activity specifies the same message variable, m-2, as the response message. The process then waits.



a g2gl-receive

Notes	OK
Item configuration	none
Names	none
G2GL partner link variable name	pl-2
G2GL port type	unspecified
G2GL operation name	2-way-invocation
G2GL correlations	none
G2GL message variable name	m-2
This is an instantiation trigger	yes

a g2gl-reply

Notes	OK
Item configuration	none
Names	none
G2GL partner link variable name	pl-2
G2GL port type	unspecified
G2GL operation name	2-way-invocation
G2GL correlations	none
G2GL message variable name	m-2
Fault name for reply	none

Receiving Multiple Messages

You use the Pick activity to accept the first of several distinct operation invocations, based on the receipt of one of many message transmissions or a timeout. The branches of a Pick activity must join together as the input connections of a Pick Join activity.

You connect a Pick activity to one or more Receive activities, each of which specifies a different named operation. The Pick activity accepts the message transmission whose named operation matches the message transmission received from a partner process's Invoke activity. The Pick activity can also connect to one or more Wait activities to provide a time-out condition.

Often, you configure the Pick activity to be an instantiation trigger. If the Pick activity is the first activity in the process, it must be an instantiation trigger. The Receive activities that connect to the output connections of a Pick activity cannot be instantiation triggers.

To choose between multiple messages:

1. Create and configure multiple g2gl-invoke activities to send multiple messages, using different values for the g2gl-operation-name.

   For details, see Invoking an Operation that Sends a Message.

2. Create a g2gl-pick activity and place it on the body of a G2GL process that should receive the multiple messages.

   The Pick activity can be in a different process from the process that sends the message. Typically, the Pick activity can initiate the process, in which case it has no input connections.

3. Create multiple g2gl-receive activities and connect them to the output paths of the Pick activity.

   If connecting more than two Receive activities to a Pick activity, drag the input connection to the Receive activity directly into the Pick activity to create a new connection.

4. Configure the g2gl-partner-link-variable-name of each Receive activity to be the same partner link variable, for example, pl-2.

5. Configure the g2gl-operation-name of each Receive activity be the named operation for each message that each Invoke activity sends, for example, operation-1 and operation-2.

6 Configure the g2gl-message-variable-name of each Receive activity to refer to a message variable whose value should be set to the message to receive.

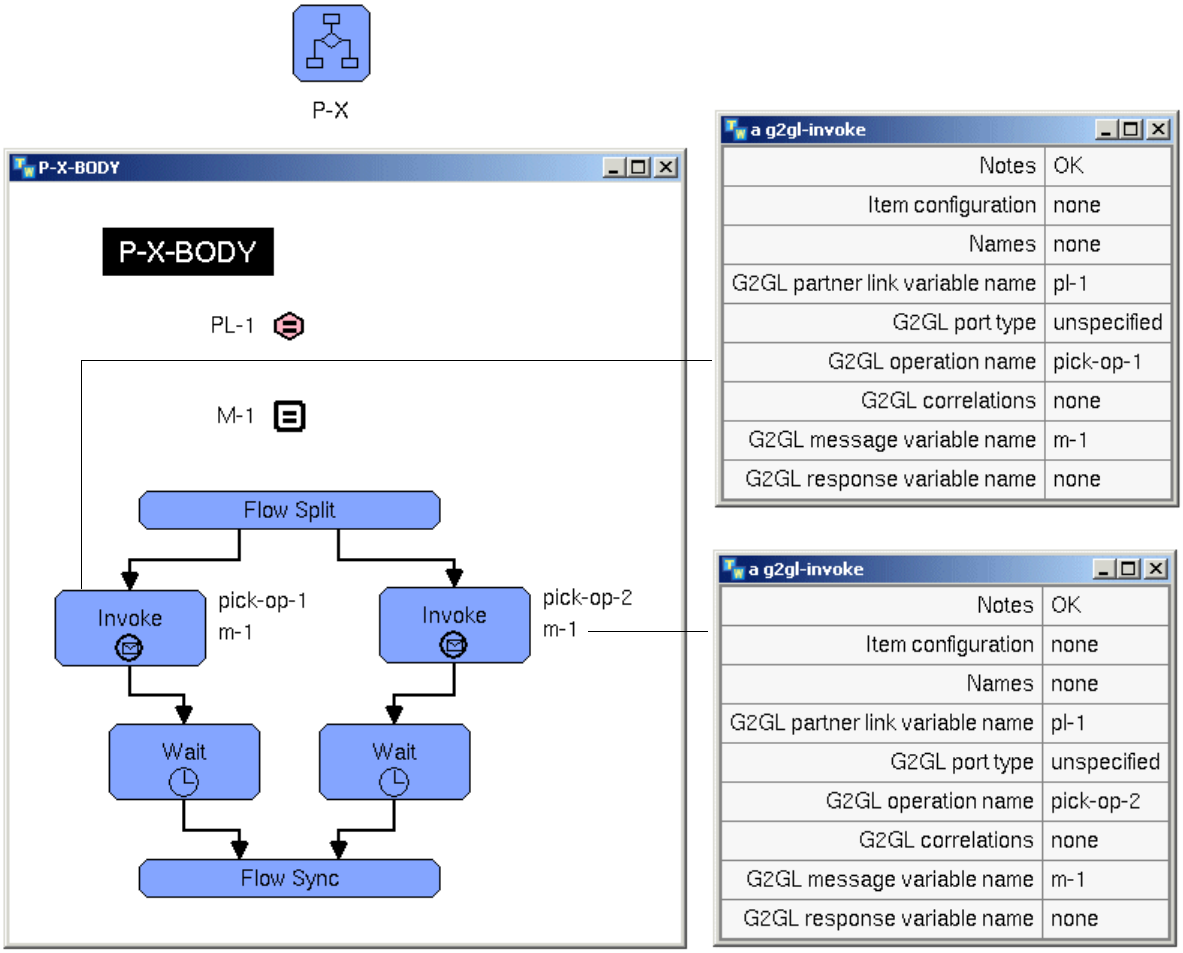
You can use the same message variable for each Receive activity.

7 Optionally, create and configure a g2gl-wait activity and connect it to the output of the Pick activity to provide a time-out condition.

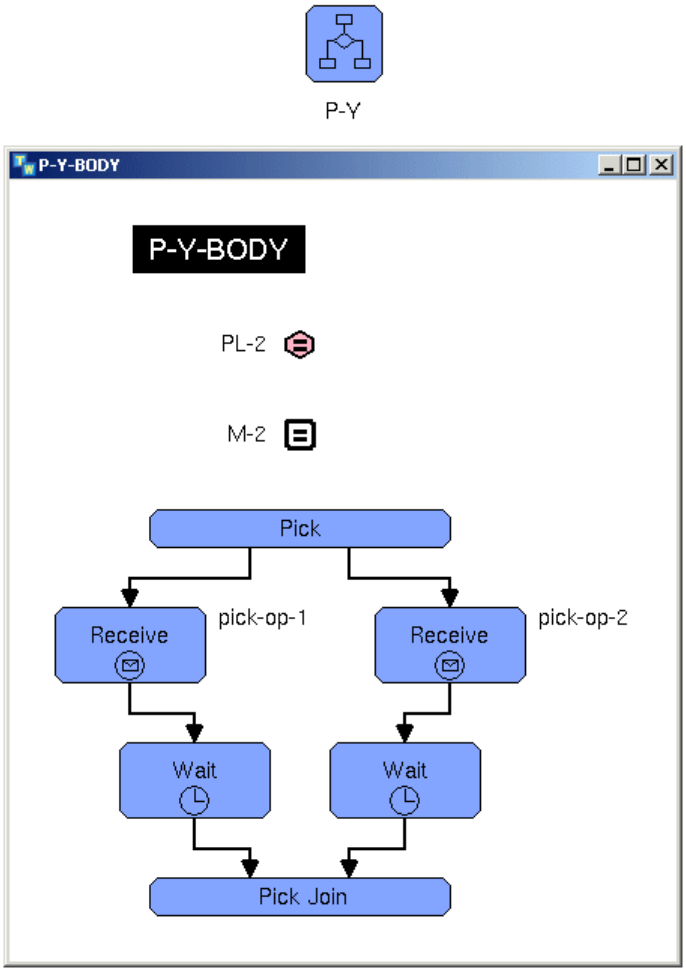
8 Configure the this-is-an-instantiation-trigger attribute of the Pick activity to be yes.

This attribute causes the process to be instantiated when the Pick activity receives a message via the named operations of one of its connected Receive activities.

Here is the body of a process named p-x and the tables for each of two Invoke activities, which invoke the operations named pick-op-1 and pick-op-2 with the m-1 message across the pl-1 partner link. The process then waits.



Here is the body of the process named p-y, which includes a Pick activity and two Receive activities. The Receive activities receive messages that the Invoke activity sends by invoking the operations named pick-op-1 and pick-op-2 with the m-2 message across the pl-2 partner link. The process then waits.



Handling Message Events

Any process that provides communication can have a Message Handler, which handles messages of a given type when they are received from outside of the process. The Message Handler is activated when the process is instantiated, and it waits for messages of the specified type to be received. When a message of the specified type is received, the body of the Message Handler executes concurrently with the execution of the process that receives the message. For example, you might use a Message Handler to handle modifications to an order.

To handle message events:

1. Choose KB Workspace > New Object > G2GL-object > G2GL-activity-with-body > G2GL-handler > G2GL-event-handler > G2GL-message-event-handler and place it at the top of the subworkspace of a G2GL process that receives messages that an Invoke activity sends.
2. Configure the g2gl-partner-link-variable-name of the Message Handler to be the same partner link variable that the Receive activity uses to receive a message, for example, pl-2.
3. Configure the g2gl-message-variable-name to be the same message variable that the Receive activity uses to receive a message, for example, m-2.
4. Configure the g2gl-operation-name to be the named operation that the Invoke activity uses to send a message, which the Receive activity receives, for example, p3-operation.
5. Create a subworkspace for the Message Handler and configure the body of the handler, as needed to handle the message.

Handling Faults

The Invoke activity can have an additional right-side output connection that can connect to the left-side input connection of a Fault Handler to handle faults that are triggered from within the process. For more information, see Handling Faults.

Also, the Reply activity can trigger a Fault Handler in the linked partner process by specifying a fault name in the fault-name-for-reply attribute. The Fault Handler in the linked partner process whose g2gl-fault-name matches the fault-name-for-reply is triggered when the fault occurs. The fault data for the fault handler is the message to which the Reply message is responding.

If an Invoke activity is connected to a Fault Handler that matches the fault-name-for-reply of a Reply activity, that Fault Handler executes before any others in the process or in any enclosing Scope activity.

Invoking Web Service Operations

Partner link variables have a default-value-for-g2gl-variable attribute, similar to local and argument variables. Its value can either be local (the default) to represent a link to another G2GL process in the same KB, or an endpoint reference specification, to represent a link to a remote Web service.

For more information, see Invoking Web Service Operations in Interfacing with Web Services.

Example: Credit Rating Partner Processes

This example shows how to implement two linked partner processes that communicate by making a credit request and providing a credit report. The credit rating requester sends a credit request and receives the credit report in response as part of the same synchronous operation. The credit information is provided as a local variable in the credit request provider and is used to determine whether the credit rating is good, ok, or bad. The credit rating requester process is called from a G2 procedure, which passes the credit information as an argument and returns the credit rating.

You can load this example from this location:

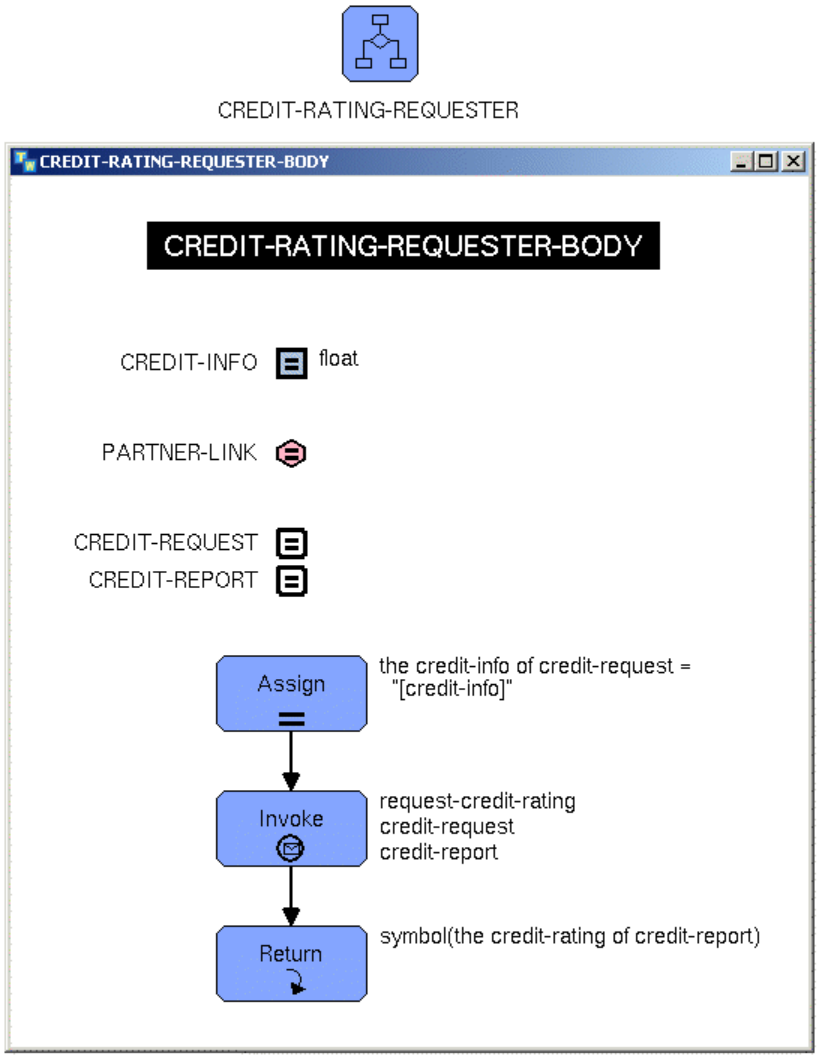
Windows	Start > Programs > Gensym G2 2011 > Examples > G2 > G2GL Credit Rating Example
UNIX	`/g2/kbs/demos/g2gl-credit-rating-example.kb`

The credit-rating-requester process assigns the credit-info attribute of the credit-request message variable to the value of the credit-info argument variable, which is passed in as an argument to the G2 procedure that invokes the process.

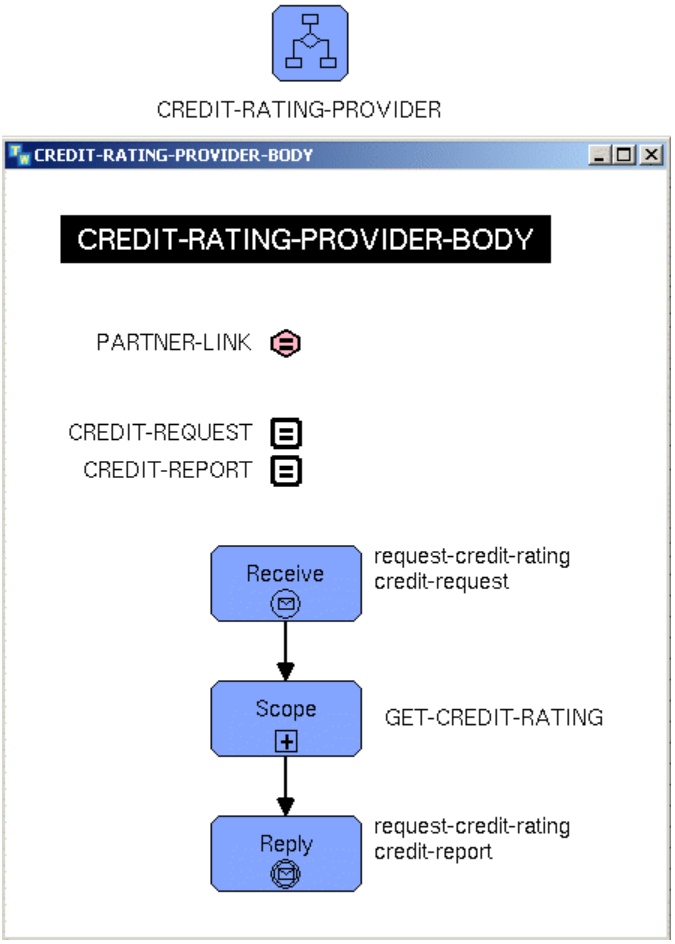
The process uses the Invoke activity to send the credit request and receive the credit report in response as part of the same synchronous operation. The Invoke activity invokes the request-credit-rating operation with the credit-request message variable and the partner-link partner link variable. The Invoke activity waits until it receives the credit report before proceeding and assigns it to the credit-report message variable.

The process returns the value of the credit-rating attribute of the credit-report message variable, which is the return value of the G2 procedure that executes the G2GL process.

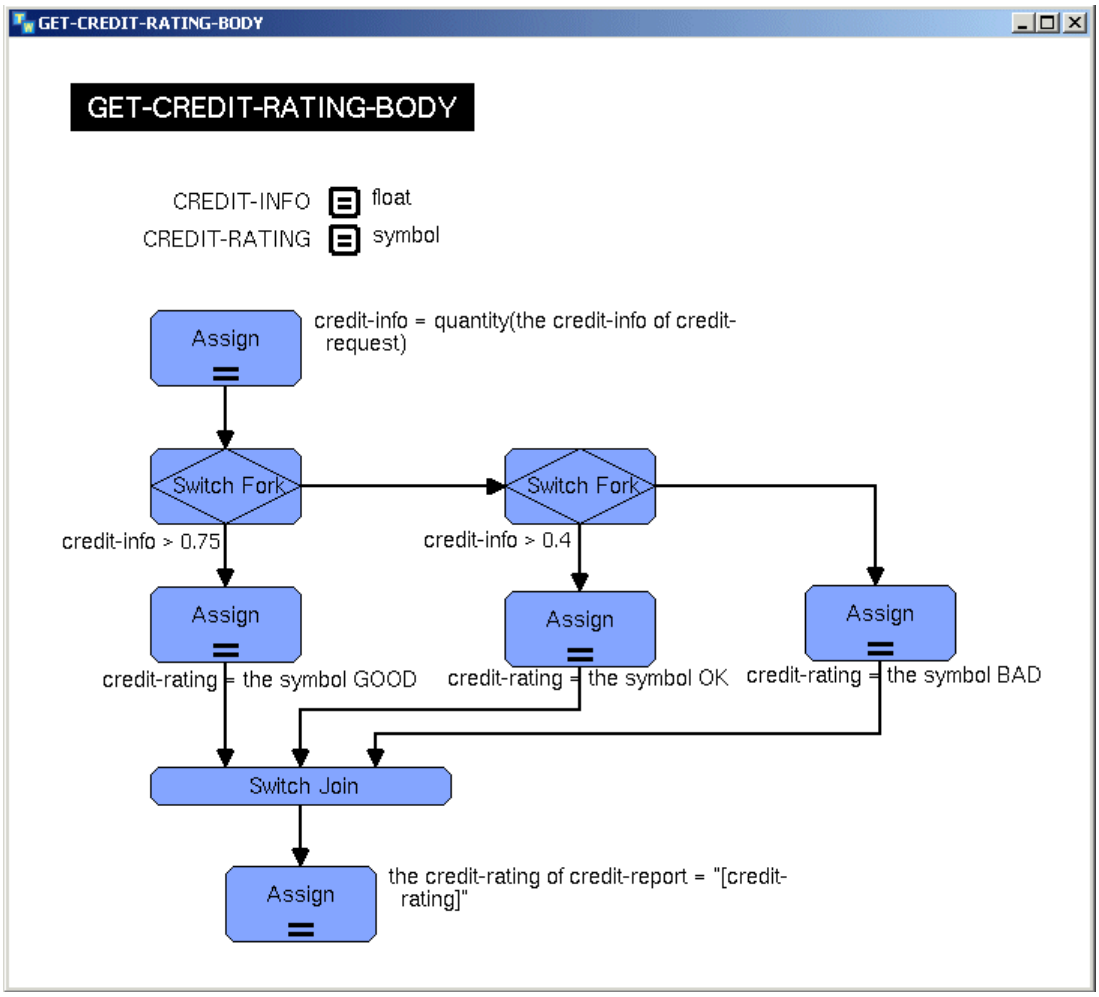
Here is the body of the credit-rating-requester process:



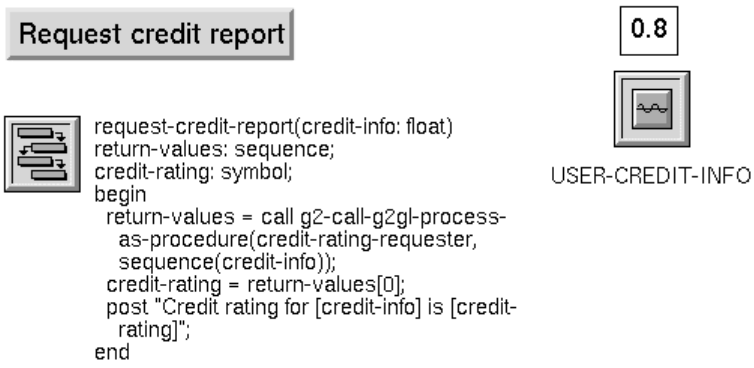
The credit-rating-provider process receives the credit request from the credit rating requester sent by invoking the request-credit-rating operation, which it stores in the credit-request message variable. The process then executes the activities on the body of the get-credit-rating scope, which creates a credit report and determines the credit rating of the credit request. It uses the Reply activity to send the credit report to the Invoke activity in the partner process by invoking the request-credit-rating operation with the credit-report message variable.



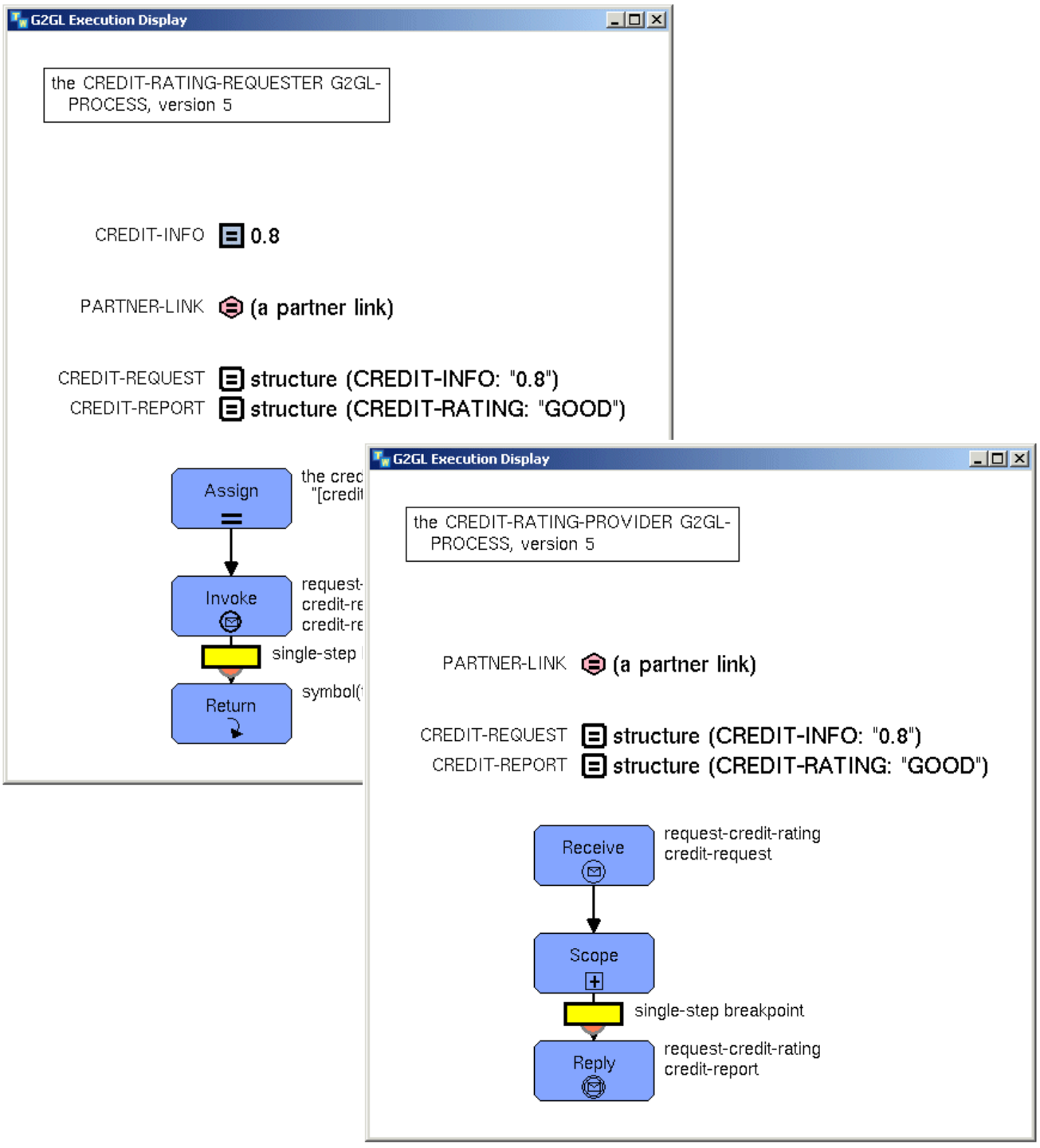
The body of the get-credit-rating scope assigns the credit-info of credit-request to the credit-info local variable, then tests the value to determine if the credit is good. It uses two Switch Fork activities to branch to three possible states, good, ok, and bad, which are assigned to the credit-rating local variable. The Switch Join joins the three inputs and the value is assigned to the credit-rating attribute of the credit-report local variable.



The request-credit-report procedure executes the credit-rating-requester process, given a value for the credit-info. It calls the g2-call-g2gl-process-as-procedure system procedure, passing in as arguments the name of the G2GL process and a sequence of argument variables defined in process. The procedure returns the value of the credit-rating variable. The button executes the procedure, using the value of the user-credit-info variable as the argument.



This figure shows the individual execution displays that result from clicking the button and single-stepping through the process. The credit-rating-requester process executes, passing in 0.8 as the value of the credit-info-arg, and returning the value of the credit-rating, which is good.



Clicking the breakpoint at the end of the credit-rating-requester execution display causes this message to appear in the Message Board.

#9 3:02:04 p.m. Credit rating for 0.8 is GOOD

Interacting with G2GL Processes

A G2GL process is must be compiled before it can be instantiated. If you change the process, you must recompile it to instantiate the latest compilation version. Once the process has been compiled, you can execute **process instances**, which are analogous to procedure invocations in G2. A process instance executes the activities on the process body flow chart, following the flow chart and using the locally specified variables.

You can set various attributes of a G2GL process to debug the process by setting breakpoints, tracing, and single-stepping through the process. You can also configure global parameters in the G2GL Parameters system table that control various aspects of execution, debugging, animation, and tracing.

You can create a G2GL process by importing it from an XML document that conforms with the BPEL specification. You can also export a G2GL process to an XML document.

Compiling G2GL Processes

Once you have defined a G2GL process, you must compile it before it can be executed.

G2GL automatically compiles a G2GL process when it is loaded from a KB and the first time it is executed by using a menu choice or a system procedure. For more information, see Executing G2GL Processes.

If you change the process, you must recompile it in order to use the latest version for the execution. If you change the process and do not recompile it, the process uses the existing compilation version for its execution.

When you compile, if an old compilation version is still being used to execute a process instance, the old version continues to exist until all process instances that are based on that version terminate. This means that, in theory, there could be any number of distinct versions of the process executing concurrently. However, all new instances of the process use the latest version that has been successfully compiled.

G2GL detects compilation errors and warnings, and displays them in the process body. The process also keeps track of the number of errors and warnings.

The G2GL process updates these attributes when the process is compiled:

- latest-attempted-compilation-version-number — The version number of the most recent successful compilation, which increments each time the G2GL process is compiled.
- latest-attempted-compilation-version-time — The time of the most recent attempt to compile a process. This attribute is saved in the KB.
- g2gl-process-compilation-version-number — The version number of the latest compilation version, which is a positive integer, or none if no compilation version exists.
- g2gl-process-compilation-version-time — The time of the latest compilation version or none if no compilation version exists.
- number-of-errors-in-latest-attempted-compilation and number-of-warnings-in-latest-attempted-compilation — The number of errors and warnings in the latest compilation attempt.
- g2gl-process-procedure-signature — A parenthesized list of argument descriptions for the process to be called as a procedure or none if no compilation version exists. An empty parenthesized list appears in the case of a process that has no argument variables (instances of g2gl-arg-variable). Each argument description is of one of the following forms:
  - For an argument variable that has no default value:

    arg-name : g2gl-variable-type
  - For an argument variable that has a default value:

    arg-name : g2gl-variable-type (default: default-value)
  - For an argument variable that is optional:

    arg-name : g2gl-variable-type (optional, default: default-value)

Note that an argument variable is optional if it has a default value and is either the last argument or is followed only by argument variables that are optional.

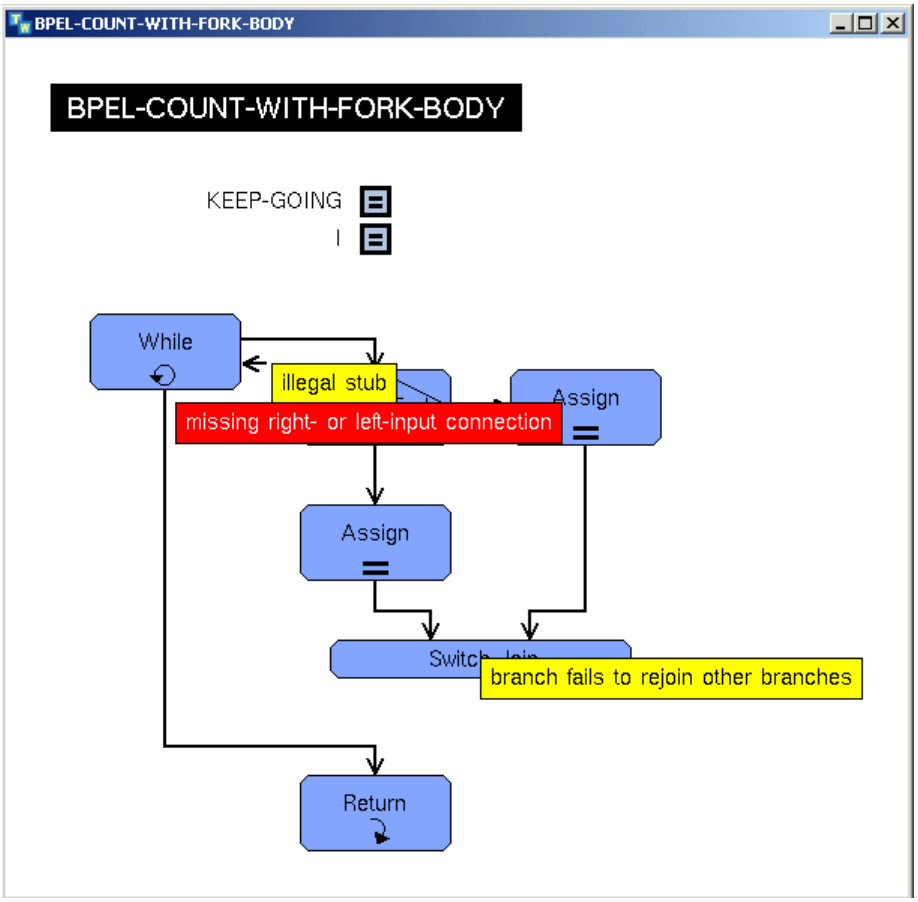
To compile a process manually:

➔ Choose Compile Process on a G2GL process or on the process body workspace.

or

➔ g2-compile-g2gl-process
(*process*: class g2gl-process)
-> *success*: truth-value

Here is a process that has compilation warnings because the While activity is missing one of its input connections:



To clear compilation postings:

➔ Choose Clear Compilation Postings on the process body workspace.

Recompiling also automatically clears old compilation postings.

Executing G2GL Processes

Once you have compiled a G2GL process, you can execute it in one of the following four ways:

- Manually, by using a menu choice on the G2GL process or process body.
- Programmatically, by calling a G2 system procedure that executes the G2GL process.

- If the process has arguments and/or return values, by calling a G2 system procedure that calls the G2GL process as if it were a procedure.
- By invoking an instantiation trigger operation on the process either from an Invoke activity in an existing process instance or programmatically by calling a G2 system procedure, which establishes a partner link. Instantiation triggers include the Receive activity and the Pick activity.

For information on using the Invoke activity to invoke instantiation triggers, see Communicating Between G2GL Processes.

You can execute a G2GL process only when G2 is running. If the process has never been compiled, executing the process also compiles it. Thereafter, executing a process uses the last good compilation.

Executing a G2GL Process Manually

To execute a G2GL process manually:

➔ Choose Execute Process on a G2GL process or on the process body workspace.

The process executes the flow chart on the body. If tracing and/or breakpoints are enabled, or if the process hits a Breakpoint activity, the individual execution display for the process instance appears. Otherwise, the process executes without any visual indication.

Executing a G2GL Process Programmatically

You can execute a simple G2GL process programmatically from a G2 procedure, as long as it does not have arguments or return values.

To execute a G2GL process programmatically:

➔ g2-execute-g2gl-process
(*process*: class g2gl-process)
-> *process-instance*: g2gl-process-instance

This procedure compiles the G2GL process first if it has not already been compiled, then executes it.

Similar to executing a process manually, the process executes the activities on the body without any visual indication unless the process specifies tracing and/or breakpoints.

The procedure returns a process instance, which you can pause, resume, and kill programmatically. For details, see Managing G2GL Process Instances.

Calling a G2GL Process as a Procedure

If you define a G2GL process with arguments and/or return values, you must call it as if it were a procedure, using a different system procedure, which allows you to pass arguments and return values.

To call a G2GL process as a procedure:

1 Create a G2GL process with arguments and/or return values.

For details, see:

- Creating a G2GL Process.
- Returning Values.

2 Specify the callable-as-procedure attribute of the G2GL process object as yes, the default.

3 Call this system procedure from a G2 procedure, passing the arguments to the G2GL process as an argument to the system procedure and returning values, as needed:

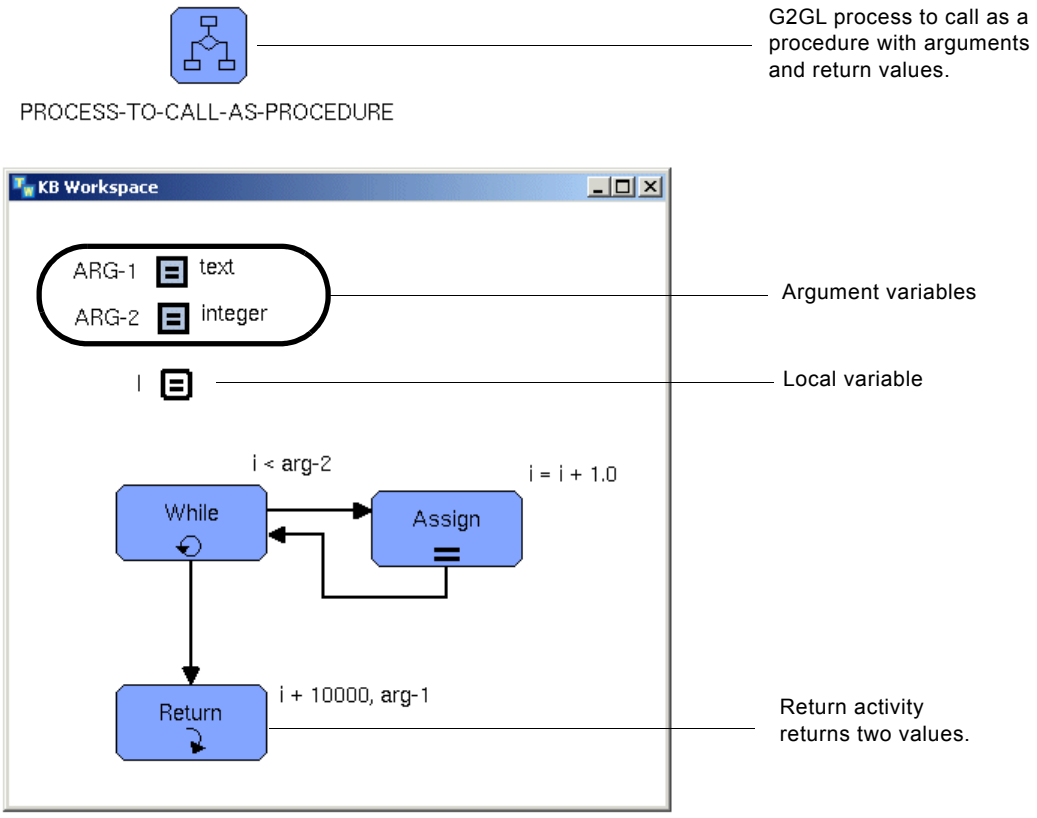
```
g2-call-g2gl-process-as-procedure
    (process: class g2gl-process, argument-list: sequence
    -> return-values: sequence)
```

This procedure compiles the G2GL process first if it has not already been compiled, then calls it.

The procedure returns a single value, the sequence of values returned by the g2gl-process, or signals an error if a fault is not handled at the top-level of the g2gl-process. The error is an instance of the system error class g2gl-fault, which is a subclass of g2-error. The error-description has a text describing the fault, which is the same as the message shown on the breakpoint execution display. The error item has two additional attributes, fault-name, a symbol, and fault-data, which is "none" for all non-user faults and for user faults that don't include fault data.

The following example shows how to call a G2GL process as a procedure with arguments and return values.

Here is the process-to-call-as-procedure process and its body, which declares two argument variables and returns two values. The process increments the local variable named i while it is less than the value of arg-2, which is passed in as an argument to the process. It returns i + 10000 and the first argument to the process.



Here is a G2 procedure that calls the process-to-call-as-procedure G2GL process as a procedure. The procedure passes in the argument list to the G2GL process as an argument to the procedure, and it posts the return values to the Message Board. The procedure uses the g2-call-g2gl-process-as-procedure API to call the G2GL process as a procedure.

```
call-to-process-to-call-as-procedure (arglist: sequence)
values-sequence: sequence;
begin
    values-sequence = call g2-call-g2gl-process-as-procedure
        (process-to-call-as-procedure, arglist);
    post "result of calling process-to-call-as-procedure with arglist [arglist]:
        [values-sequence]";
end
```

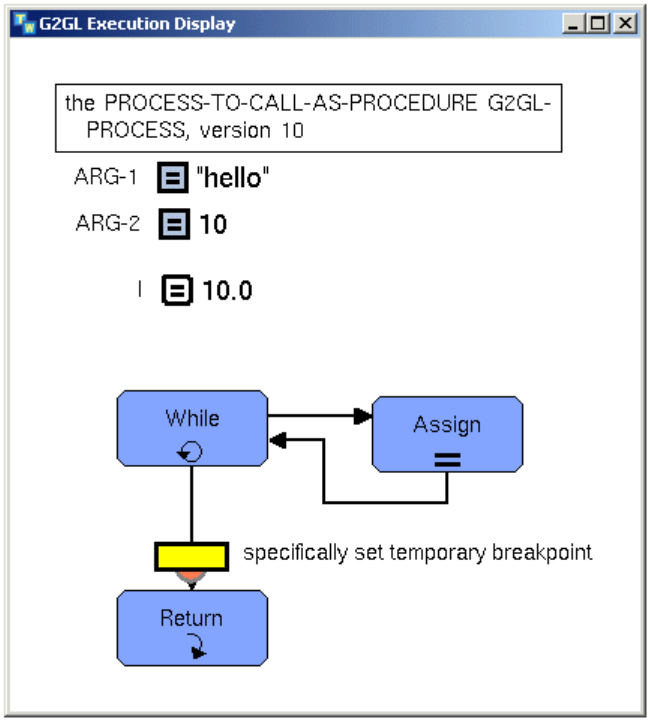
This button executes the call-to-process-to-call-as-procedure procedure:

start procedure call

```
start call-to-process-to-call-as-procedure
  (sequence("hello", 10))
```

```
start call-to-process-to-call-as-procedure (sequence("hello", 10))
```

Here is the individual execution display that results when setting a temporary breakpoint on the Return activity. Notice that the values of arg-1 and arg-2 are set to the arguments passed to the procedure that calls the process as a procedure. Just before the process returns its values, i is equal to the value of the second argument.



When you click the temporary breakpoint to finish the execution of the process, the Message Board posts this information about the return values:

```
#39  9:32:06 a.m.  result of calling process-to-
  call-as-procedure with arglist sequence ("hello",
  10): sequence (10010.0,
  "hello")
```

Invoking Instantiation Trigger Operations Programmatically

You can use the following system procedure to invoke an instantiation trigger operation in a G2GL process programmatically:

g2-invoke-g2gl-operation
(*service-switch*: class g2gl-service-switch, *operation-name*: symbol, *input-message*: item-or-value)
-> *output-message*: item-or-value, *reply-or-fault-name*: symbol

In order for a G2GL process to be available for instantiation this way, the name of the *service-switch* item must be in its names-of-g2gl-service-switches-for-instantiation list, or, if that is "none", the name of the *service-switch* item must be its name-of-g2gl-service-switch-for-connection. By default, every KB has a g2gl-service-switch item named g2gl-standard-service-switch, which is also the default name-of-g2gl-service-switch-for-connection for G2GL processes.

The *operation-name* is the name of the instantiation trigger operation. The *input-message* and *output-message* are Web service message structures. For a description of web service message structures, see Web Services in Web Operations in the *G2 System Procedures Reference Manual*.

The *reply-or-fault-name* is either the symbol reply if the reply was not a fault, or else the name of the fault.

Note Currently, this system procedure can only invoke two-way synchronous operations. The system procedure always waits for a reply from the G2GL process. If the process exits without responding, a g2gl-fault error is signaled, whose fault-name is partner-has-terminated.

Managing G2GL Process Instances

G2GL provides the g2gl-process-instance class, which is the G2GL analog of the G2 procedure-invocation class. G2GL creates an instance of this class is when you:

- Start a G2GL process via the g2-execute-g2gl-process system procedure.
- Evaluate the following G2GL expression:

  this process instance

- Call the following system procedure, which returns all process instances, g2-collect-all-g2gl-process-instances.

The g2gl-process-instance class defines the g2gl-process-instance-is-paused attribute, which is true when the process instance is paused and false otherwise.

Pausing and Resuming G2GL Process Instances

You can pause and resume process instances from an individual execution display. The current state of the process instance appears in the title bar. You can also pause and resume process instances programmatically.

To pause and resume a G2GL process instance interactively:

➔ Choose pause process instance and resume process instance on an individual execution display workspace, depending on the current state.

To pause a G2GL process instance programmatically:

➔ g2-pause-g2gl-process-instance
(*process-instance*: class g2gl-process-instance)

or

➔ Conclude that the g2gl-process-instance-is-paused attribute of the process instance is true.

To resume a paused G2GL process instance programmatically:

➔ g2-resume-g2gl-process-instance
(*process-instance*: class g2gl-process-instance)

or

➔ Conclude that the g2gl-process-instance-is-paused attribute of the process instance is false.

Deleting G2GL Process Instances

You can delete individual G2GL process instances associated with a process interactively or programmatically. Deleting an execution instance stops the execution of that instance and deletes its individual execution display.

To delete a process instance interactively:

➔ Choose Delete Process Instance on an individual execution display.

To delete an individual G2GL process instance programmatically:

➔ g2-kill-g2gl-process-instance
(*process-instance*: class g2gl-process-instance)

To delete all executing G2GL process instances for a process:

➔ g2-kill-all-g2gl-process-instances
(*process*: class g2gl-process)

Getting All G2GL Process Instances

You can get a list of all G2GL process instances associated with a process.

To get all G2GL process instances:

➔ g2-collect-all-g2gl-process-instances
(*process*: g2gl-process)
-> *process-instances*: sequence

Debugging G2GL Processes

To debug a G2GL process, you can show **individual execution displays** when executing the processes. Individual execution displays represent execution instances of a G2GL process and allow you to trace the execution of the process, including its variable assignments and subprocesses.

Individual execution displays use **thread tokens** to show execution flows. These thread tokens move around the execution display as the process executes.

You can show individual execution displays by configuring the process to:

- Trace the top-level process or all levels.
- Use breakpoints on the top-level process or all levels.
- Use temporary breakpoints, which you can set in an individual execution display.
- Use single-stepping, which sets breakpoints at each activity.
- Automatically break on execution faults.

Breakpoints cause an execution thread to pause at that point. To continue processing, you click on the breakpoint icon.

A G2GL process can have multiple execution threads executing concurrently, which you can view in a single execution display that superimposes all executing process threads.

Tracing G2GL Processes

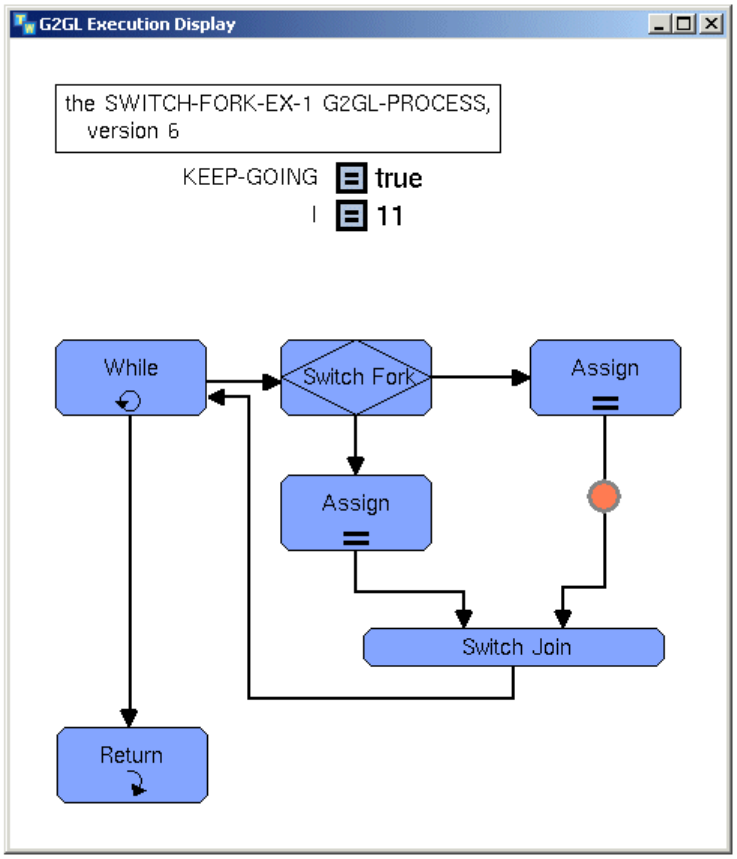
You can trace the top-level process or the top-level process and all subprocesses. When tracing a process, G2GL shows an individual execution display for each execution instance, depending on the tracing level.

To trace a process:

➔ Set the individual-execution-display-mode of a G2GL process to one of these values:

- trace-top-level to show an individual execution display for the top-level process.
- trace-all-levels to show an individual execution display for the top-level process and any subprocesses.

Here is the individual execution display that appears when tracing a process. Notice the thread token, compilation version information, and variable values.



Setting Breakpoints

You can set a breakpoint at the beginning of the top-level process or at the beginning of the top-level process and all subprocesses. When setting a breakpoint, G2GL shows an individual execution display for each execution instance, depending on the breakpoint level.

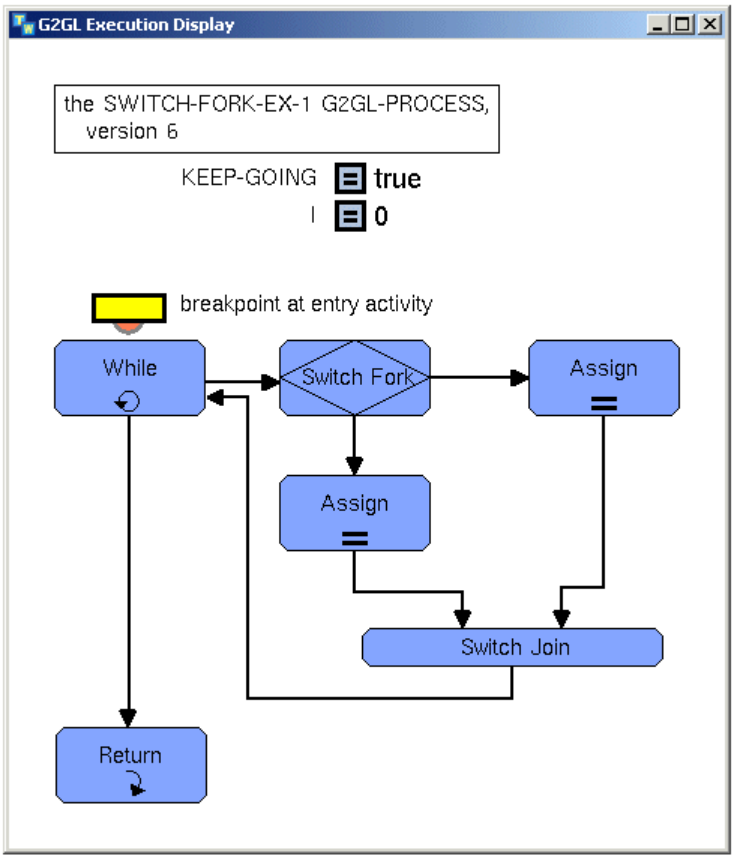
You can also use the Breakpoint activity to set a permanent breakpoint in the process. For details, see Breakpoint.

To set a breakpoint:

➔ Set the individual-execution-display-mode of a G2GL process to one of these values:

- break-on-entry to show an individual execution display for the top-level process with a breakpoint at the beginning of the process.
- break-on-entry-at-all-levels to show an individual execution display for the top-level process and any subprocesses, with a breakpoint at the beginning of the top-level process and each subprocess.

Here is an individual execution display with a breakpoint. To continue processing, click the breakpoint.



Setting Temporary Breakpoints

You can set a temporary breakpoint on an activity instance within an individual execution display. The breakpoint remains until you explicitly remove it or the display disappears. The temporary breakpoint only affects the current individual execution display.

To set a temporary breakpoint:

1 Execute the process to show an individual execution display.

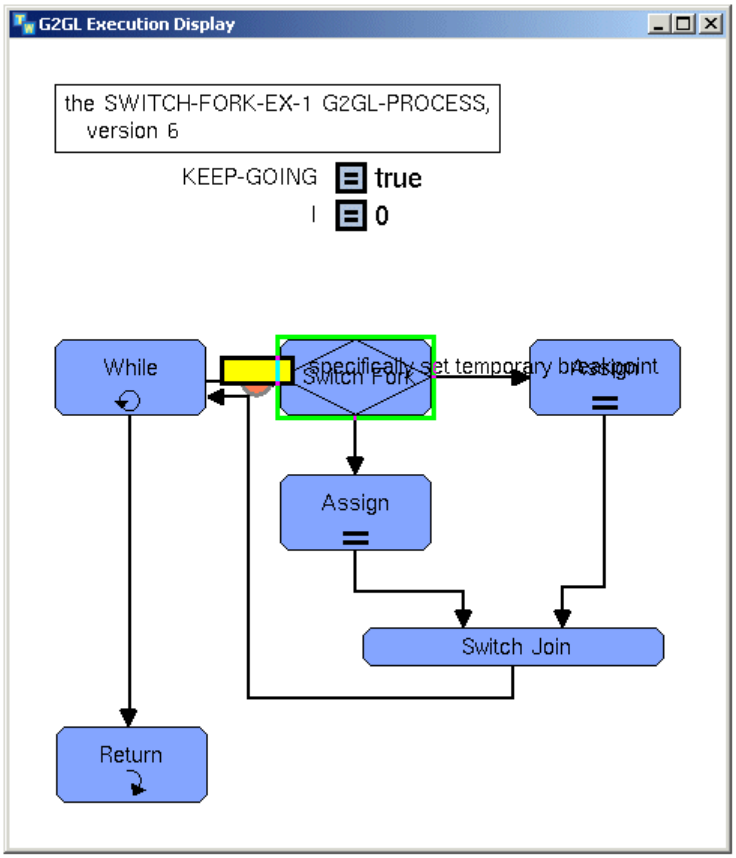
2 Choose set temporary breakpoint on an activity instance within the execution display.

When the thread token reaches the temporary breakpoint, it pauses and a breakpoint appears. Click the breakpoint to continue.

To remove a temporary breakpoint:

➔ Choose remove temporary breakpoint on the activity instance that has a breakpoint set.

Here is an individual execution display with a temporary breakpoint set on the Switch Fork activity:



Single-Stepping through the Execution

When you are tracing or are at a breakpoint, you can single-step through the process, which sets temporary breakpoints at each activity instance in the individual execution display. Single-stepping only affects the current individual execution display.

To single-step through the execution:

1. Enable tracing or breakpoints for the process.
2. Execute the process to show an individual execution display.
3. Choose Single-Step on the individual execution display workspace.

Note Single-stepping is not available for superimposed tracing execution displays.

The execution stops at each activity instance and a temporary breakpoint appears. Click each breakpoint to continue.

To remove single-stepping:

➔ Choose Do Not Single-Step on the individual execution display workspace.

Showing Superimposed Tracings Execution Displays

When setting tracing or breakpoints, you can show a **superimposed tracing execution display** to show simultaneous executions of instances of a process in a single execution display. You can set superimposed tracing for the top-level process or for the top-level process and all subprocesses.

To show superimposed execution displays:

➔ Set the superimposed-tracings-execution-display-mode of a G2GL process to one of these values:

- trace-top-level to show a superimposed execution display for the top-level process.
- trace-all-levels to show a superimposed execution display for the top-level process and any activities with subprocesses.

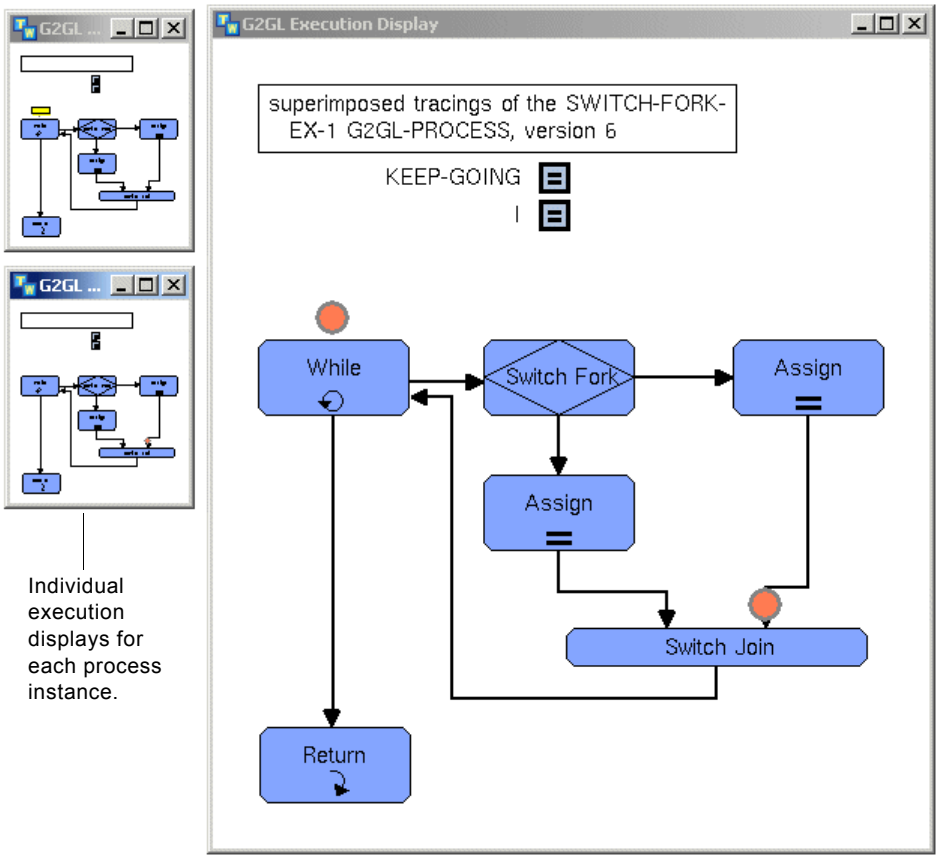
To use superimposed tracing:

1. Enable superimposed tracings.
2. Execute the process.

   The superimposed tracings execution display appears, which is scaled.
3. While the process is still executing, execute the process again.

The superimposed tracings display now shows thread tokens for both executing processes.

Here is a superimposed tracing execution display when two process instances are executing. Individual execution displays for these instances appear in the upper-left and overlap; move the top workspace to expose the workspace underneath. Notice that the superimposed tracing display has two thread tokens. The top thread token is paused at the first activity because a breakpoint has been set in the individual execution display. The other thread token is moving through the process.



Individual execution displays for each process instance.

The superimposed execution display shows two thread tokens, one for each process instance.

Automatically Breaking on Execution Faults

During debugging, you can configure a process to display a breakpoint automatically if it encounters a fault during execution. For example, an unbound variable causes an execution fault. You can also configure a parameter in the G2GL Parameters system table to break on execution faults for all processes.

If a process encounters a fault during execution and it is configured to break on execution faults, it shows the individual execution display with a breakpoint and error message at the fault location. If it is not configured to break on execution faults, there is no visible indication that a fault has occurred. Therefore, we

recommend that you always enable breaking on execution faults until you have fully debugged the process, at which point you can disable it.

To configure an individual process to break on execution faults:

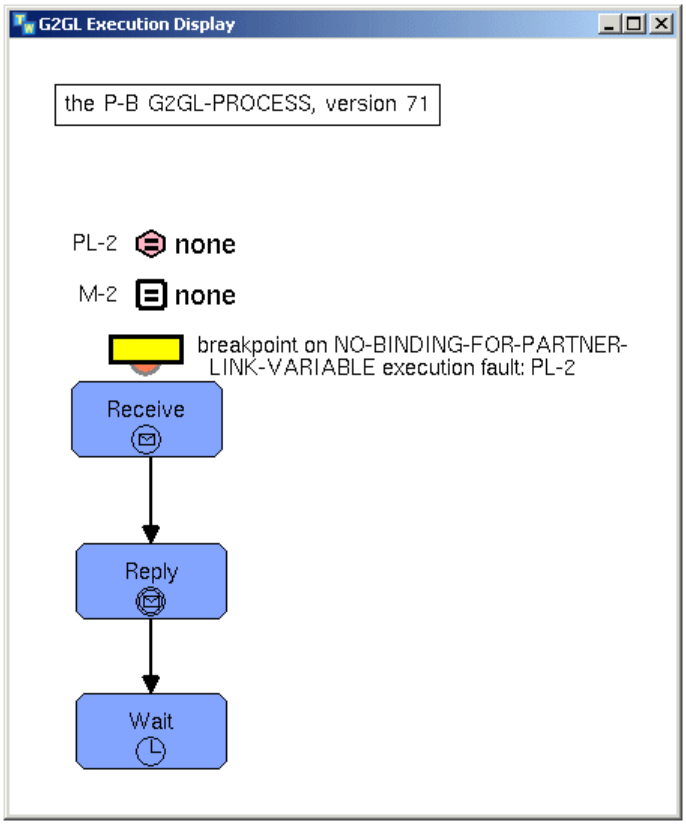
➔ Configure the break-on-execution-fault attribute of the G2GL process to be yes.

To configure all processes to break on execution faults:

➔ Configure the break-on-all-execution-faults attribute of the G2GL Parameters system table to be yes.

Breaking on execution faults occurs if the attribute is enabled for all processes or for the individual process.

Here is an example of an execution fault for a message variable that has no value:



Configuring Debugging for an Individual Execution Instance

Once a process is executing, you might want to change the tracing and breakpoints settings for the individual execution instance.

To configure debugging for an individual execution instance:

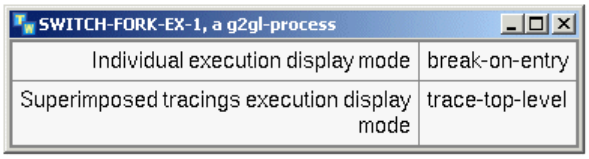
1 Choose Process Display Attributes on an individual execution display or superimposed tracing display workspace.

A table appears with the individual-execution-display-mode and superimposed-tracings-execution-display-mode attributes for the associated process.

2 Configure these attributes, as needed.

The process now uses these values for these attributes.

Here is the table that appears:



Continuing without Debugging

When tracing or breakpoints are set, you might want to continue executing the process without debugging and close the associated individual execution display or superimposed tracing display.

To continue without debugging:

➔ Choose Close and Continue on an individual execution display or superimposed tracing display workspace.

Displaying the Source

When tracing or breakpoints are set, you might want to display the body of the G2GL process associated with an individual execution display or superimposed tracing display.

To display the source:

➔ Choose Bring Up Source on an individual execution display or superimposed tracing display workspace.

This menu choice does not appear for subprocess instances.

Configuring G2GL

To configuring G2GL, you:

- Configure attributes in the G2GL Parameters system table.
- Override default G2GL icons.

Configuring G2GL Parameters

To configure G2GL parameters:

➔ Choose G2GL System Attributes on the individual execution display or superimposed tracing display workspace.

or

➔ Choose Main Menu > System Tables > G2GL Parameters.

For details, see G2 Graphical Language (G2GL) Parameters.

Overriding Default Icons

The G2GL Parameters system table provides the g2gl-object-icon-substitutions attribute for overriding the default icon for G2GL objects, including activities, processes, and process instances.

The syntax for this attribute is a list of this form:

[*g2gl-class*: *object-subclass*;]...

Thus, to override the icon for a G2GL object, create a subclass of the object class with the desired icon, then specify the G2GL class whose icon you want to replace followed by the object subclass as the value of the attribute. You can specify a list of icon substitutions, separated by semi-colons.

Exporting G2GL Processes as XML

You can export a G2GL process to an XML document. The structure of the XML document corresponds to the BPEL specification.

You can also export a G2GL process to a G2 text, which contains the entire XML document. Exporting to a text is significantly faster than exporting to a file. The maximum size of the text containing the XML code is 1 MB. The procedure must complete before the scheduler allows other processing to occur.

G2GL extensions to BPEL are exported to the `http://gensym.com/g2gl/` XML namespace URI and use the `g2gl:` prefix. G2GL uses the `http://gensym.com/g2gl/g2gl-expression` namespace for the `expressionLanguage` and `queryLanguage` attributes.

When exporting a g2gl-process as XML, the value of the g2gl-namespace-map attribute is converted to a set of XML namespace declarations on the process element.

The g2gl-target-namespace attribute on g2gl-process corresponds to the `targetNamespace` attribute on the `<process>` element. It is initialized to an empty string when you create a g2gl-process in G2.

To export a G2GL process as XML to a file:

➔ g2-export-g2gl-process-as-xml
(*process*: class g2gl-process, *file-path*: text)

To export a G2GL process as XML to a text:

➔ g2-export-g2gl-process-as-xml-text
(*process*: class g2gl-process)
-> *document*: text

Importing G2GL Processes from XML Documents

You can create a G2GL process by reading it from an XML document that describes the BPEL process specification.

You can also import a G2GL process from a G2 text, which contains the entire XML document. Importing from a text is significantly faster than importing from a file. The maximum size of the text containing the XML code is 1 MB. The procedure must complete before the scheduler allows other processing to occur.

Note Currently, G2GL only supports the G2GL expression language. It does not support the standard BPEL expression language, XPath, or any other expression language. When importing from XML documents, G2GL discards expressions in non-G2GL languages.

The G2GL process object body contains the following types of objects, which correspond to the BPEL specification:

G2GL Definition	G2GL Class	BPEL Specification	G2GL Features
G2GL argument variables	g2gl-arg-variable	Not implemented in BPEL, except in the body of a Fault Handler	• Argument variables appear at the very top of the process body, where their order implies the order of arguments to the process. • names attribute corresponds with `name` element, with hyphens inserted in front of capital letters, which are lower-cased. • g2gl-variable-type attribute refers to the variable type, which is either a G2GL message type or G2GL type.
G2GL partner link variables	g2gl-partner-link-variable	Partner link variables	• Appear as icons at the top of the G2GL process body below argument variables. • names attribute is the same as G2GL argument variables. • g2gl-variable-type attribute refers to corresponding G2GL partner link type definition.

G2GL Definition	G2GL Class	BPEL Specification	G2GL Features
G2GL correlation variables	g2gl-correlation-variable	Correlation set	• Appear as icons at the top of the G2GL process body below partner link variables. • names attribute is the same as G2GL argument variables. • g2gl-variable-type is a comma-separated list of G2GL message types.
G2GL local variables	g2gl-local-variable	Variables	• Appear as icons at the top of the G2GL process body below partner link variables and correlation variables. • names attribute is the same as G2GL argument variables. • g2gl-variable-type attribute refers to the variable type, which is either a G2GL message type or G2GL type.

G2GL Definition	G2GL Class	BPEL Specification	G2GL Features
G2GL activities	g2gl-activity subclasses	Activities	• Appear below variables in a connected process flow chart. • names attribute is the same as G2GL argument variables. • BPEL Sequence activity is implicit in the order in which the activities are connected in the process flow. • Individual specification and behavior depends on activity. See Summary of Differences Between G2GL and BPEL Activities.
G2GL scopes, fault handlers, alarm event handlers, message event handlers, and compensation handlers	g2gl-activity-with-body subclasses: • g2gl-scope • g2gl-fault-handler • g2gl-alarm-event-handler • g2gl-message-event-handler • g2gl-compensation-handler		• Appear below variables and above connected flow chart activities. • Subworkspace contains variables and activities in a subprocess flow.

To import a G2GL process from an XML file:

➔ g2-import-g2gl-process-from-xml
(*file-path*: text)
-> *process*: class g2gl-process

To import a G2GL process from XML text:

➔ g2-import-g2gl-process-from-xml-text
(*document*: text)
-> *process*: class g2gl-process

Part V

User Interface Components

Chapter 39: Graphs

Presents chart styles and graphs, and show you how to use them.

Chapter 40: Trend Charts

An introduction to and description of trend charts and their use.

Chapter 41: Windows Menus

Provides examples of how to create Windows menus in Telewindows by using one of two techniques: rendering native GMS menus in Telewindows and using the Native Menu System.

Chapter 42: Windows Dialogs

Provides examples of how to create basic and custom Windows dialogs.

Chapter 43: Custom Windows Dialogs

Describes the complete specification for creating custom Windows dialogs, including a description of the various dialog controls.

Chapter 44: Windows Views, Panes, and UI Features

Provides examples of how to create Windows chart views, HTML views, shortcut bars, tree views, and other Windows user interface features.

31

Buttons

Describes action and radio buttons, check boxes, sliders, and type-in boxes.

Introduction

Buttons are user interface items that either perform actions (via action buttons), or provide a value to a variable or parameter. This chapter describes the different kinds of buttons and how to use them.

Types of Buttons

The types of buttons are as follows. All buttons except action buttons are associated with a variable or a parameter:

Button	Description
Action button	Causes G2 to perform one or more actions.
Check box	Assigns an *on* or *off* value to a symbolic, quantitative, logical, or text-value variable or parameter.
Radio button	Assigns one of a mutually exclusive set of symbols, numbers, logical values, or text values to a variable or parameter.
Slider	Assigns numeric values to a variable or parameter as a pointer slides along a horizontal guide.
Type-in box	Lets you enter a symbolic, quantitative, logical, or text value for a variable or parameter.

Note While buttons provide solutions for some of your user-interface requirements, we recommend that you use the GUIDE/UIL product, shipped with G2. GUIDE/UIL provides an extensive set of user-interface tools. See the *G2 GUIDE User's Guide* for more information.

Subclassing Buttons

You can subclass all system-defined buttons. Subclassing buttons lets you redefine their icon descriptions, and provide the default values for any attributes that you require.

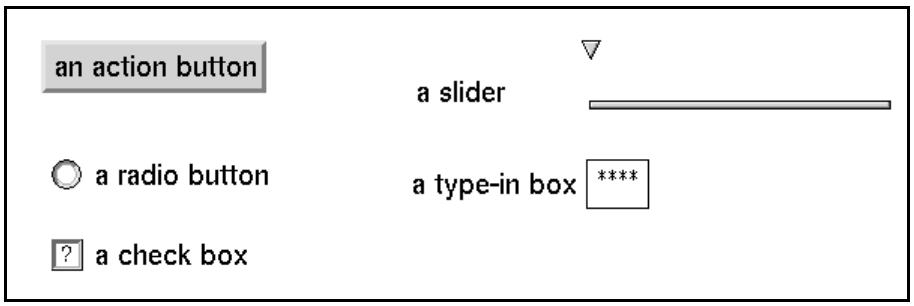
Creating Buttons

To create a button:

➔ Select KB Workspace > New Button > *button-type*.

where *button-type* is a type of button. The new button appears on the workspace connected to the mouse. Move the mouse to position the button and click to place the button on the workspace.

Here are examples of each type of button:



Tip You cannot open the attribute table of a button without pausing the KB. If a button's attribute table is visible while the KB is running, however, you can edit its attributes.

Common Attributes of Buttons

Buttons have the following class-specific attributes. Attributes unique to a particular button are explained in the sections describing that button.

Attribute	Description
label	Provides a description of the button.
Allowable values:	Any text string
Default value:	none
variable-or-parameter	The variable or parameter whose value the button represents. Action buttons do not include this attribute.
Allowable values:	Any variable or parameter
Default value:	none

Providing a Label for the Button

The optional label attribute provides a textual description for the button, which you enter as a text string within quotation marks (" "). The default is none.

For action buttons, the label appears inside the rectangular boundary of the button. The size of the action button increases to accommodate the length of the label.

The label appears to the right of check boxes and radio buttons and to the left of sliders and type-in boxes.

Use the label to describe a button, or to indicate which value the variable or parameter expects when the button is active.

Representing the Variable or Parameter

The variable-or-parameter attribute indicates which variable or parameter receives a value when the user clicks on the button. This is a required attribute for check boxes, radio buttons, sliders and type-in boxes, but is not applicable to action buttons. The default is none.

For radio buttons, several buttons represent different values for one variable or parameter. For check boxes, each box represents a separate variable or parameter.

Buttons associated with a variable or a parameter always attempt to update whenever their variables or parameters receive values from other sources. For example, if a check box is set to on, and a rule sets the variable or parameter to its off value, G2 sets the check box to off. Similarly, if two check boxes exist for the same variable, turning off one turns off the other, unless the check boxes have different on or off values.

Action Buttons

An action button lets you start one or more actions interactively. Each time you click on an action button when G2 is running, G2 highlights the button and executes the specified action(s) once. For information about actions, see Actions.

Selecting an action button can cause forward chaining by updating a variable, or backward chaining by using an expression requiring a variable's value. You can repeatedly click an action button if you wish to repeat the action(s).

The only way to invoke an action button is by clicking on it. You cannot scan, forward chain, or backward chain to an action button, nor can you invoke it programmatically.

Entering the Actions to Execute

The required action attribute is where you specify what action(s) G2 should perform when a user clicks on the button. The grammar and other considerations are similar to those applicable to the consequent of a rule. For further information, see:

- Coding the Consequent.
- Executing Actions in the Consequent in Parallel.
- Executing Actions in the Consequent Sequentially.

An action button is effectively an unconditionally rule with some specific restrictions. For example, while you can use local names within action statements, you cannot use the keywords whenever and if in an action button. The unconditionally statement is implied for simple actions — *unconditionally* perform one or more actions whenever the user clicks on an action button.

The statements you enter in an action button vary slightly from the similar expressions you enter in a rule.

Using Unconditionally Statements

Use the unconditionally statement following a for statement, for example:

```
for any message M upon this workspace
    unconditionally change the border-color of M to red
```

Using For, When, and Then Statements

Use the for, when, and then statements as you would in a rule, for example:

```
for any laptop LT connected to any network-connection NC
    when the power-status of LT is on then
        conclude that the connect-status of NC is connected
```

Using the In Order Statement

Multiple actions in an action button execute in parallel unless you use the in order statement to specify sequential execution, for example:

```
in order
    create a secret-box S and
    transfer X to covert-ws at (50, 75) and
    make S permanent
```

Controlling the Scheduling Priority

The action-priority attribute controls the priority at which the G2 executes the action(s). When you select an action button, G2 schedules the action at the specified priority.

The default priority for an action button is intentionally high (2) because the purpose of an action button is to execute one or more statements upon demand. For a complete description of task priorities, see Task Scheduling.

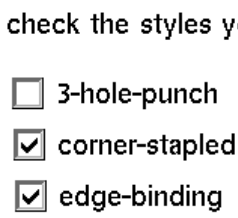
Class-Specific Attributes

The class-specific attributes of action buttons are:

Attribute	Description
action	The compiled statements that are executed when a user clicks the button.
Allowable values:	A statement, or series of statements.
Default value:	No default value
action-priority	The priority at which G2 executes the statements.
Allowable values:	1 – 10
Default value:	2

Check Boxes

Check boxes control autonomous choices that can assign either an on or off value to a symbolic, quantitative, or text variable or parameter. When the box is checked, the assigned value is on, and an X appears in the check box. When the choice is off, the check box is blank. The following example shows the 3-hole-punch and edge-binding options with an on value, and the corner-stapled option with an off value.



When you create check boxes, you assign each box to a separate variable or parameter.

Specifying the Activation Value

The value-on-activation attribute specifies what value, if any, the variable or parameter receives when the check box is activated. The default is none.

Although this attribute is not required, we recommend that you enter a value identical to the on-value attribute if you want the check box to be selected when activated, or the same as the off-value attribute if you want the check box to be blank when activated.

If you do not set the default value, the check box contains a question mark when it is activated.

Allowable values for this attribute depend on the data type of the variable or parameter.

Specifying the On and Off Values

The on-value and off-value attributes are required values for check boxes, indicating the value that G2 assigns to the variable or parameter when the check box is selected and when it is not. The default value is none. Allowable and default values are data-type specific for the variable or parameter.

For this variable or parameter type...	The value can be...
Quantitative	Any float or integer.
Integer	Any integer.
Float	Any float.
Symbolic	Any symbol.
Logical	Any truth value.
Text	Any text string.

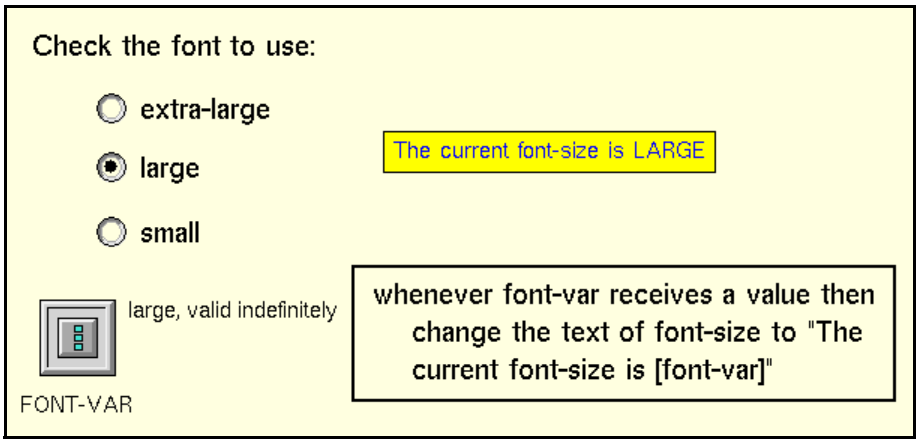
Class-Specific Attributes

The class-specific attributes of check boxes are:

Attribute	Description
value-on-activation	The value, if any, to assign the variable or parameter upon item activation.
Allowable values:	Any appropriate value for the variable or parameter being represented.
Default value:	none
on-value	The value to assign the variable or parameter when the check box is selected.
Allowable values:	Any appropriate value for the variable or parameter being represented.
Default value:	none
off-value	The value to assign the variable or parameter when the check box is not selected.
Allowable values:	Any appropriate value for the variable or parameter being represented.
Default value:	none

Radio Buttons

A radio button represents a single choice within a set of choices that are mutually exclusive. Use radio buttons to assign a symbolic, numeric, or text value to a variable or parameter. This example shows several radio buttons being used to select a font size, and a corresponding rule that updates the text of a message, called font-size, with the current choice.



While several radio buttons typically appear together, it is the designation of a single variable or parameter to each button that groups a series of buttons, not their physical proximity. Moving a radio button in a series to another location does not change its grouping. You assign several radio buttons to a single variable or parameter, each button representing a different value. In the example above, the buttons represent the symbolic values extra-large, large, and small for the symbolic variable font-var.

By selecting one of the buttons, you assign a value to the variable or parameter. Only one radio button can be selected at a time. G2 indicates which button is chosen by placing a black dot in the center of the icon.

Specifying the Value Upon Activation

The value-on-activation attribute specifies what value, if any, the variable or parameter receives when the radio button is activated. The default is none. Allowable values depend on the data type of the variable or parameter.

If you want a radio button to be selected when it is activated, enter the same value that you enter in the on-value attribute. Only one button in a group should have a value for value-on-activation. If you want all buttons to be blank at activation, leave the default none for all buttons in the group.

Defining the Selected Value

The on-value attribute defines the value that G2 assigns to the variable or parameter when you select the radio button. This is a required attribute for a radio button. The default is none. Allowable values are specific to the type of the variable or parameter that the button represents.

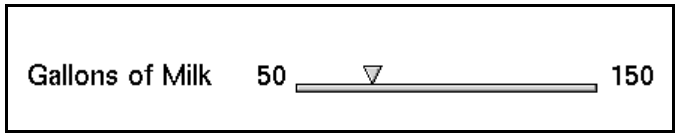
Class-Specific Attributes

The class-specific attributes of radio buttons are:

Attribute	Description
value-on-activation	The value, if any, to assign the variable or parameter upon item activation.
Allowable values:	Any appropriate value for the variable or parameter being represented.
Default value:	none
on-value	The value to assign the variable or parameter when the radio button is selected.
Allowable values:	Any appropriate value for the variable or parameter being represented.
Default value:	none

Sliders

A slider updates the value of a variable or a parameter when you slide the marker. You can use sliders only for numeric variables and parameters. The slider shown here has minimum and maximum values, 50 and 150, respectively.



By dragging the marker along the line of the slider with the mouse, you change the value of the variable or parameter within the range of the minimum and maximum values. These values appear at the left and right ends of the slider,

respectively. The current value of the position occupied by the marker appears below the slider by default. The current value of the slider in the example is 86.

G2 can distinguish up to 200 values between the minimum and the maximum value. If the range is greater than 200, you cannot use the slider to indicate all possible values in the range. For example, if the range is 0 to 1000, the user can select only values in increments of 5 (0, 5, 10... 995, 1000, and so on). Thus, the slider readout displays the value corresponding to the position of the slider, which is not necessarily the variable's exact value.

Specifying the Activation Value

The value-on-activation attribute specifies what value, if any, the variable or parameter receives when the slider is activated. Providing a value for this attribute displays the value below the slider when it is activated, unless you have the when-to-show-value attribute set to never or only while sliding.

Setting the Minimum and Maximum Values

The minimum-value and maximum-value attributes specify the lowest and highest values in the range of values you can specify for the slider. You must provide a value for both of these attributes. The minimum-value appears on the left side of the slider, and the maximum-value on the right side.

Specifying When to Update a Value

The set-value-while-sliding? attribute indicates whether G2 should update the value of the variable or parameter concurrently with slider motion. The default value of no indicates that the value changes only once the slider stops moving. Changing the default to yes causes G2 to update the variable or parameter with each slider motion.

Specifying When to Show a Value

The when-to-show-value attribute specifies when to display the current value of the variable or parameter below the slider. The default is always.

This value...	Indicates that the slider value...
always	Displays at all times, whether the marker is sliding or not.
never	Never displays.
only while sliding	Displays while the marker is sliding, but disappears when it is not.

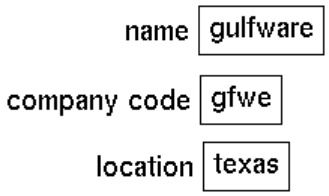
Class-Specific Attributes

The class-specific attributes of sliders are:

Attribute	Description
value-on-activation	The value, if any, to assign the variable or parameter upon item activation.
Allowable values:	Any appropriate value for the variable or parameter being represented.
Default value:	none
minimum-value	The lowest value in the slider's value range.
Allowable values:	Any number.
Default value:	none
maximum-value	The highest value in the slider's value range.
Allowable values:	Any number.
Default value:	none
set-value-while-sliding?	Determines whether G2 updates the variable or parameter whenever the slider moves.
Allowable values:	yes no
Default value:	no
when-to-show-value	Determines the gesture that causes G2 to display the value.
Allowable values:	always never only while sliding
Default value:	always

Type-in Boxes

A type-in box lets you enter values from the keyboard. G2 uses what you enter as the new value of the variable or parameter that you associate with the type-in box. Here are several type-in boxes, representing several symbolic variables:



Type-in boxes are specific to the variable or parameter type they represent. G2 does not permit you to enter a value of the wrong type. For example, if you try to enter text into a type-in box that is assigned to a float variable, the variable retains its previous value, and G2 signals an error.

Specifying the Activation Value

The value-on-activation attribute indicates what value, if any, appears in the type-in box upon activation. The value could be any value of the appropriate type for the variable or parameter that the type-in box represents.

If you do *not* specify a value in the value-on-activation attribute and the variable or parameter has an initial value, that value appears when the type-in box is activated.

If the type-in box represents...	Then when activated...
A variable that does not have an initial value	Asterisks appear in the box.
A parameter that does not have a user-specified initial value	The data type-specific default initial value appears in the box.

As soon as the variable or parameter for the type-in box receives a value, G2 replaces the asterisks or the default data type value with the new value. The size of the type-in box increases or decreases to accommodate the current value.

Specifying the Formatting Style

The format-for-type-in-box attribute specifies the format in which to display the value. The default value is default, indicating that G2 displays the value in the appropriate format for the data type of the variable or parameter.

You can specify any of these formatting styles:

- free text to represent textual values without quotation marks.
- interval to represent quantities as time intervals, such as 1 hour, 30 minutes, and 10 seconds.
- time stamp to represent quantities as G2 timestamps, such as 1 Jan 2004 12:00:00 a.m.
- Any *ddd.dddd-format* to represent floating point numbers with decimal point precision, where a d represents each digit, such as 123.4567. If you enter a value that exceeds the specified format, G2 displays it as an exponential value.

The type-in box in this example sets the value of the text variable named my-var. Notice that the type-in box does not use quotation marks. To accomplish this, the format-for-type-in-box and display-format is configured as free text.

MY-TYPE-IN, a type-in-box

Notes	TYPE-IN-BOX-XXX-7: OK
Names	MY-TYPE-IN
Label	"Enter a text value"
Value on activation	hello
Variable or parameter	my-var
Format for type in box	free text
Blank for type in?	yes
Have edit option buttons for type in?	no

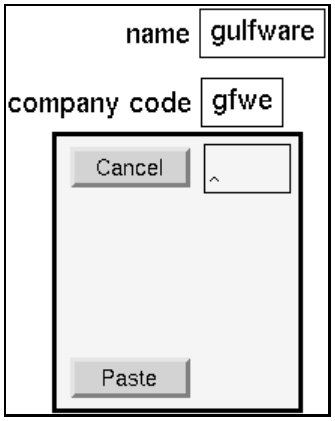
Defining the Selection Status

The blank-for-type-in? attribute defines whether to clear the type-in box when you select it. The default value of yes automatically deletes any value when you click in the box to enter a value. Changing the default to no maintains the previous value, which you can then edit.

Specifying Editor Options

The have-edit-option-buttons-for-type-in? attribute specifies whether certain Text Editor buttons appear with the type-in box when you select it to change its value. The default is no.

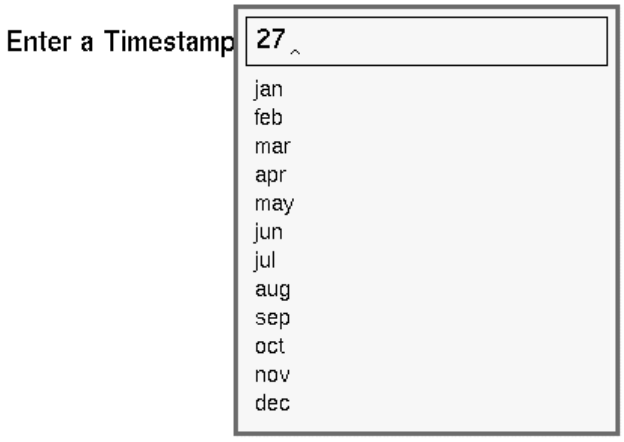
Changing the default to yes displays edit option buttons when you click in the type-in box. The next example show the Text Editor buttons that appear after pressing a space character after the name.



Showing Editor Prompts

The show-prompts-for-type-in attribute controls whether to show formatting prompts when entering text. You might want to display editor prompts when the format requires specific G2 syntax, such as a timestamp or an interval. By default, type-in boxes do not show prompts.

For example, here is a type-in box that is configured to require a timestamp and to show editor prompts:



Enter a Timestamp 27 Nov 2004 12:00:00 a.m.

a type-in-box

Notes	OK
Item configuration	none
Names	none
Label	none
Value on activation	none
Variable or parameter	par
Format for type in box	time stamp
Blank for type in?	yes
Have edit option buttons for type in?	no
Show prompts for type in	yes

Class-Specific Attributes

The class-specific attributes of type-in boxes are:

Attribute	Description
value-on-activation	The value, if any, that appears in the box when the type-in box is activated.
Allowable values:	Any appropriate value for the variable or parameter being represented.
Default value:	none
format-for-type-in-box	Determines the format in which to display the value.
Allowable values:	any *ddd.dddd-format* free text interval time stamp default
Default value:	default
blank-for-type-in?	Determines whether G2 clears the type-in box when a user selects it.
Allowable values:	yes no
Default value:	yes
have-edit-option-buttons-for-type-in?	Determines whether limited Text Editor buttons appear for editing.
Allowable values:	yes no
Default value:	no

Attribute	Description
show-prompts-for-type-in	Determines whether Text Editor prompts appear when editing type-in box values.
Allowable values:	yes no
Default value:	no

32

Text Items

Describes how to create text items and how to use text inserts.

Introduction

You can create two types of text items: text items that you use to label your KB, and text items that you use to insert text directly into the text editor.

Using Free Text to Label Your KB

You can create a type of text item called **free text** to label various items in your KB. Using free text lets you label your KB informatively and attractively.

The two types of free text are:

- **Free text**, which includes the text and a border:

  Gensym

- **Borderless free text**, which includes just the text:

  Gensym

Free text is an item, which means you can perform the normal set of operations on it, such as changing the color, cloning, transferring, and changing the minimum size. You can also change the background color of the text. For free text with a border, you can also control the color of the border.

You control the font size of the free text through the Fonts system table.

Creating Free Text

To create free text:

1 Select one of the following commands:

KB Workspace > New Free Text > free-text

or

KB Workspace > New Free Text > borderless-free-text

G2 opens the text editor.

2 Enter the text and select the End button.

You do not need to enter quotes. G2 capitalizes the text the way you enter it.

3 Position the text outline, and click to place it on the workspace.

Changing the Color of Free Text

You can change the following color regions for the two types of free text:

- For free text, you can change the text color, the background color, and the border color.
- For borderless free text, you can change the text color and the background color.

To change the color of a region of free text:

1 Click just outside the border of the free text, and select the color menu choice.

2 Choose the region whose color you want to change.

3 Choose a color from the color palette.

This example shows bordered and borderless free text before and after changing the background color to gray:

Changing the Font of Free Text

By default, free text uses a large font size. You can change the default font size used for all free text, or you can change the font size for individual text. You can use extra-large, large, or small fonts:

Gensym Gensym Gensym

To change the default font size of free text:

1 Select Main Menu > System Tables > Fonts to display the Fonts system table.

2 Edit the font-for-free-text attribute in the system table by specifying one of the font sizes above.

To change the font size of an individual free text:

➔ Mouse right on the free text and choose font, then choose extra large, large, or small.

Using Text Inserters to Insert Text into the Text Editor

You can create a type of text item called a **text inserter**, which allows you to click on an item to insert text directly into the text editor. You use this feature when you must enter text repeatedly, for example, when creating menus or expressions.

The four types of text inserters are:

- *Text* inserter: Inserts all of its text into the text editor.
- *Word* inserter: Inserts a single word at a time.
- *Character* inserter: Inserts the selected character only.
- *Character sequence* inserter: Inserts a sequence of characters that you select.

Creating and Editing a Text Inserter

You create a text inserter from the New Free Text menu. Once you have created a text inserter, you can edit its text, if necessary.

To create a text inserter:

1 Select KB Workspace > New Free Text > text-inserter.

2 Choose the type of text inserter.

G2 displays the text editor for entering the text.

3 Enter the text to appear in the text inserter, then click the End button.

4 Position the text inserter, and click to place it on the workspace.

To edit the text of a text inserter:

1 With the text editor closed, click the text inserter, then click table.

The text inserter's table appears. Its text appears at the bottom of the table.

2 Click the text of the text inserter.

The text editor opens on the text of the text inserter.

3 Edit the text of the text inserter, then click the End button.

Using Text Inserters from the Scrapbook

Whenever you cut or copy text from the text editor, and whenever you drag your cursor over visible text to insert it into the text editor, G2 creates a text inserter and places it in the scrapbook. The **scrapbook** is a non-KB workspace that G2 creates, which you can access as a named workspace.

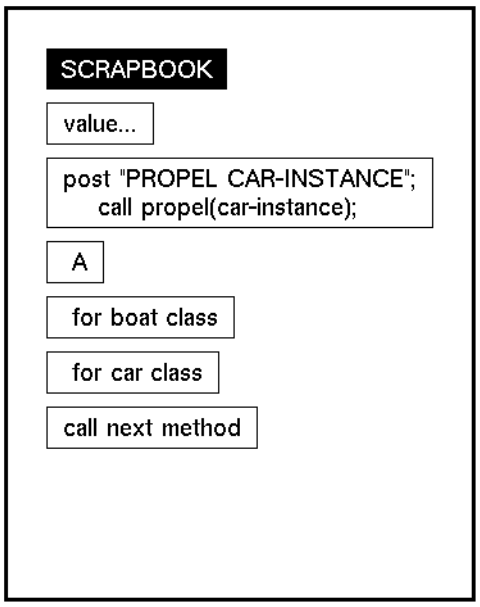
You can use the text inserters in the scrapbook directly, to insert text into the text editor, or you can transfer text inserters from the scrapbook to another workspace to make them a permanent part of your KB.

For more information about the scrapbook and about cutting and pasting text, see Using the Clipboard and Scrapbook.

To use text from the scrapbook as a text inserter:

➔ Choose Main Menu > Get Workspace > scrapbook.

G2 displays a workspace named scrapbook containing text inserts such as the following:



4 To insert text from a text inserter:

   a Open the text editor.

   b Click on a text inserter.

      G2 inserts the entire text at the current cursor location in the text editor.

5 To transfer the text inserter to another workspace and make it permanent:

   a With the text editor closed, click on a text item in the scrapbook to display its menu; or with the text editor open, click on the border of the text item.

   b Select the transfer command to transfer it to another workspace.

      The text item from the scrapbook is not a permanent part of your KB.

Using Text Inserters to Insert Text

You can use text inserters to insert text into the text editor when you are editing any type of text. You might want to create a workspace that contains all your text inserters so they are available to you at any time.

If you do not place them on a separate workspace, you may want to label them so that they are not confused with regular text boxes.

To insert text into the text editor by using a text inserter:

1 Open the G2 text editor.

For example, you open the text editor when creating a rule or procedure, or when editing the attributes of a table. For more information, see Opening the Text Editor.

2 With the text inserter visible, click or select the desired text in the text inserter, depending on the type of text inserter:

When using this type of text inserter...	Do this...
Text inserter	Click anywhere on the text inserter to insert the entire text.
Word inserter	Click on a single word in the text inserter to insert the word and the following space.
Character inserter	Click on a character in the text inserter to insert the character.
Character sequence	Drag the cursor over a sequence of characters in the text inserter to insert the character sequence.

G2 inserts the specified text from the text inserter into the text editor at the current cursor position.

You can also drag the cursor over a sequence of characters when inserting text from a text, word, or character inserter. G2 inserts the selected character(s), including the following space if you drag over a word.

Note that when dragging the cursor over a *text* inserter, however, you must begin selecting from the first character, whereas when dragging the cursor over any other type of inserter, you can begin selecting anywhere.

The character sequence inserter also has the characteristics of a word inserter; you can click on a word to insert it.

33

User Menu Choices

Describes how to define application-specific menu choices.

Introduction

A user menu choice is an item of the user-menu-choice class, which lets you define application-specific menu choices that specify one or more actions. G2 executes the action(s) specified by the user menu choice when a user chooses the option interactively. For information about actions, see Actions.

Working with User Menu Choices

A user menu choice performs an action when the user selects it. You associate a user menu choice with a particular class so that it appears on the menu of the specified class and its subclasses.

A user menu choice is visible in an item's menu only when the KB is running and the value of the menu choice's condition attribute is true or unspecified.

If the KB is reset or paused while a menu containing a user menu choice is displayed on the screen, or if the condition for a menu choice becomes false after the menu has been displayed, the menu choice remains visible but G2 will not execute the action if the user chooses it.

User menu choices for a class appear in an arbitrary order after the system-defined choices. Furthermore, the order is not guaranteed to be stable across G2 sessions in which user menu choices are defined or modified.

To create a new user menu choice:

➔ Select KB Workspace > New Definition > user menu choice.

The KB must be running for user menu choices to work.

Labelling the Menu Choice

The label attribute determines the name of the menu choice as it appears on the item menu. By default, the attribute display of the label attribute appears at the upper right-hand side of the user menu choice.

The user menu choice label cannot be the name of a system-defined menu choice, nor should it be identical to another user menu choice for the same class. If the label is the same as a system-defined menu choice, G2 executes only the system-defined choice, not the user-defined menu choice.

If the label is identical to another user menu choice for the same class, G2 executes one of the choices randomly when one is selected.

Capitalization in Menu Entries

By default, G2 removes any hyphens that you enter as part of the label symbol, and converts the name to the appropriate case for the class upon which the menu choice appears. For example, if you enter the label as:

MY-CHOICE

for a kb-workspace class, the menu entry appears as:

My Choice

since all KB Workspace menu choices use initial capital letters. The same choice assigned to an object class such as g2-variable would appear on the menu as:

my choice

because menus for item classes other than KB Workspace are lowercase.

Entering escape characters to force capitalization of menu choice labels does not affect how they appear. For example, entering this:

TEST-@A@B@C-TEST

to appear as:

Test ABC Test

does not take effect. The menu entry appears as:

Test Abc Test

on a KB Workspace menu, and as:

```
test abc test
```

on an item menu class other than KB Workspace.

Hint A user menu choice has an incomplete status until you complete the label, applicable-class and action attributes, which are all required attributes.

Defining the Applicable Class

The applicable-class attribute defines the class to which the user menu choice applies. You must specify an applicable class, or the menu choice will not work. The applicable class must be item or a subclass of item.

In general, the user menu choice also applies to subclasses of the applicable class. The exception is when the subclass does not meet the condition of the user menu choice, in which case the user menu choice does not appear for the subclass.

Controlling When the Menu Choice is Available

The condition attribute controls when the menu choice is available or when G2 will execute the action.

The condition of a user menu choice is either none, or any *logical-expression* returning a truth-value. If you do not provide a condition and the condition remains none, the user menu choice always appears on the applicable item menu and G2 always executes the action when the user selects this choice.

If the condition contains an expression, G2 evaluates the expression at the time the user clicks on an object to get the item menu. If the condition is false or unknown, the user menu choice does not appear on the item menu.

If the condition is true, the user menu choice appears on the item menu and the user can select it. If the user chooses the user menu choice, G2 evaluates the condition again. If the result is still true, G2 performs the action. If the result is false or unknown at the second evaluation, G2 does not perform the user menu choice action.

Specifying the Action to Execute

The action attribute specifies the action G2 executes when the menu choice is selected. Enter any G2 action(s) in this attribute. The grammar and other considerations are similar to those applicable to the consequent of a rule. For further information, see:

- Coding the Consequent.
- Executing Actions in the Consequent in Parallel.

- Executing Actions in the Consequent Sequentially.

You can refer to the item to which the menu choice applies using the statement the item, as shown in the next example. The statement the item refers to the item the user clicked.

```
move the item by (10, 10)
```

Similarly, you can also use the statement this workspace in a user menu choice action, indicating the G2 window in which the user selection took place.

Specifying the Scheduling Priority

The action-priority attribute specifies the priority at which G2 schedules the action to execute.

The default priority for a user menu choice is intentionally high (2), because the purpose of the menu choice is to execute one or more statements upon demand.

For a complete description of task priorities, see The G2 Scheduler.

User Menu Choice Attributes

The class-specific attributes of user menu choices are:

Attribute	Description
label	The name of the menu choice as it appears in the item's menu.
Allowable values:	*symbol*
Default value:	none
applicable-class	Specifies the class to which the menu choice applies.
Allowable values:	Any class
Default value:	none

Attribute	Description
condition	The condition under which the menu choice is available, or when G2 executes its action.
Allowable values:	*logical-expression*
Default value:	none
action	The action to execute when a user selects the menu choice.
Allowable values:	Any G2 action.
Default value:	No default value
action-priority	Indicates the priority at which G2 executes the action for the menu choice.
Allowable values:	1 – 10
Default value:	2

34

External Images

Explains how to use external images in workspace backgrounds and icons.

Introduction

An **external image** is a bitmap or other graphical image created outside G2. You can import an external image into a KB and use it as an icon, part of an icon, or the background image of a workspace.

Supported Graphics Formats

G2 supports JPEG, X Bit Map, (XBM, black and white only) and Graphics Interchange Format images as follows:

Image File	Icons	Workspaces
JPEG	✓	✓
XBM (black and white)	✓	✓
GIF87A (black and white, non-interlaced)	✓	✓
GIF87A (color, non-interlaced)		✓
GIF89A (color, transparent)	✓	

G2 supports the 89A GIF format, which permits any color in the image to be transparent. Where such a GIF exists, G2 draws nothing wherever the transparent color appears, thus making visible whatever is below the transparent section on the screen. The non-transparent GIF formats paint the designated transparent color to the screen, making the color opaque.

Note The use of transparent GIFs applies only to color GIF files that are displayed as the background layer of icons. It does not apply to any color GIFs that a workspace may be using as a background.

Various GIFs that can be used in image definitions are available in the G2 `demos` directory, as described under GIF Files.

Note JPEG images do not currently work when displayed on an X server with 8 bits/pixel. To work around this limitation, switch the X server's display to true color, using 16, 24, or 32 bits/pixel.

Caution Bitmap images will not display correctly on Windows computers if the Color Palette settings are changed while G2 or TW is running. You can rectify an incorrect display state by shutting down and then restarting G2 or TW.

Working with External Images

The general technique for using an external image in a KB is:

- Create an image-definition item to represent the image within the KB.
- Edit the image definition's attributes to give the image a name and designate the file that contains it.
- Use the image name to include the image in an icon or workspace.

Creating an Image Definition

An image definition is an instance of the system-defined class image-definition.

To create an image definition:

➔ Select KB Workspace > New Definition > image-definition.

The class-specific attributes of an image definition are as follows:

.

Attribute	Description
names	Any symbol representing the name of the image that you want to use.
Allowable values:	Any symbol
Default value:	none
Notes:	You must provide an image definition name since you reference it when you use the definition in a KB.
file-name-of-image	A text string of the pathname of the image file you want to use.
Allowable values:	Any valid filename of a JPEG, GIF, or XBM image.
Default value:	none

Attribute	Description
save-image-data-with-kb	Determines whether an image file is stored as part of a KB, or resides externally to a KB in an appropriate directory. See Saving an Image with a KB for more information about this. *Allowable values:* yes no *Default value:* no
format-of-image	The file format. G2 supplies this information dynamically after you enter (or update) a valid image filename. You cannot edit this attribute. *Allowable values:* gif jpeg xbm *Default value:* none
width-of-image	The width of the image as an integer value representing workspace units. G2 supplies this information dynamically after you enter (or update) a valid image filename. You cannot edit this attribute. Use this and the height-of-image values to determine the appropriate icon size in the icon editor. For more information, see Image Size and Icon Size.

Attribute	Description
height-of-image	The height of the image as an integer value representing workspace units. G2 supplies this information dynamically after you enter (or update) a valid image filename. You cannot edit this attribute. Use this and the width-of-image values to determine the appropriate icon size in the icon editor. For more information, see Image Size and Icon Size.
depth-of-image	The depth of the image as an integer value representing bits. This value is always 1 for XBM images, ranges from 1 to 8 for GIF images, and is either 8 or 24 for JPEG images. G2 supplies this information dynamically after you enter (or update) a valid image filename. You cannot edit this attribute.

Specifying the Name of the Image

In order to use an image in a KB, you must give the image a name, so that other KB components can refer to it. You can use any available name.

To give an image a name:

➔ Edit the names attribute of the image definition to specify the desired name.

Specifying the Pathname of the Image File

In order to use an image in a KB, you must tell G2 where in the file system to find the file that contains the image.

To designate the pathname of the image:

➔ Edit the file-name-of-image attribute to specify the pathname of the image in the file system.

You can enter the pathname as an absolute pathname, a pathname relative to the current KB, or a filename if the image file is located in the directory that holds the executing KB. If you have not loaded a KB, G2 searches for the file in the current directory.

G2 reads the image file into the current KB when you enter (or update) a valid filename for this attribute.

Using an Image in a KB

An image definition is ready for use as soon as you enter a valid name in the names attribute and a valid filename in the file-name-of-image attribute.

To use an image in a KB:

➔ Include the name of the image definition in the:

- image attribute of the icon editor.
- icon-description attribute of a class definition.
- background-images attribute of a workspace table.

For details on using external images with icons, see Including Externally Created Images.

For details on using external images as the backgrounds for workspaces, see Using a Graphic as a Background Image.

Note The size of the image to be used as a workspace background cannot exceed 65,536 by 65,536 pixels.

Saving an Image with a KB

The save-image-data-with-kb attribute of an image definition determines whether the image file is saved as part of your KB. The default for this attribute is no. Change this attribute to yes to save the image as part of your KB.

Advantages and Disadvantages

As a general guideline, you will probably not want to save image files with your KB during development, since changes to the image file will not be reflected in the KB. Note that any KB that requires an image file, but does not save the image with the KB, is inherently incomplete.

Saving an image file with your KB protects the integrity of the image file for deployment purposes, and ensures that the KB is complete. However, the size of your KB will increase by significantly more than the size of the image file because:

- File compression is not currently used for images within a KB.
- Storing an image carries an overhead above that of the data in the image.

If KB size is an issue, you may not want to save image files with the KB.

Deploying a KB without saving its images requires that you determine an appropriate directory location for the image files to be stored at a customer site.

Since directory specifications are operating-system specific, you can increase KB portability by saving the image with the KB. Also, consider the fact that image files in a directory at a customer site are subject to change and deletion.

Omitting the Pathname of an Image Saved with a KB

If you specify a valid image definition, then use the save-image-data-with-kb attribute to save the image with the KB, you can thereafter change the definition's file-name-of-image attribute to be none. G2 knows that the image has been saved in the KB, and therefore does not describe the image definition as incomplete in its notes attribute, as it would if no pathname were specified and no image had been saved.

Updating an Image in a KB

G2 reads an image into a KB in three instances:

- You enter a valid filename value in the file-name-of-image attribute.
- You load a KB and G2 has finished loading all of the definitions.
- You run the g2-refresh-image-definition system procedure.

If you save the image with the KB and then want to update the image, enter the new filename in the image definition after loading the KB.

35

Messages

Describes how to work with messages.

Introduction

A message, an item of class message, displays text that provides information to the user. For example, as a result of an inform or post action for the operator, G2 creates and displays a message on the message board.

Using Messages

You can create messages in two ways by:

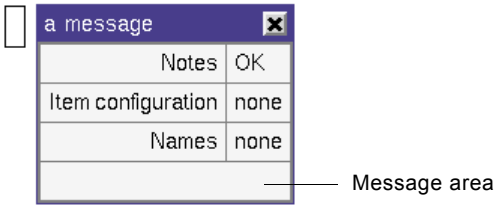
- Choosing a system-defined message class.
- Creating a user-defined message class.

Creating a Message

To create a generic message:

➔ Select KB Workspace > New Free Text > message.

The message appears on the workspace. Open its table to enter text into the message area, as shown here.



Creating a New Message Class

You can create your own message classes to define a particular font size, font and background color, and so on. For example, you could create a message class called user-warning-message that uses an extra-large font and has a red background. You could then use instances of this class to alert users to problems while the KB is running.

As with object classes, you create message classes in a hierarchy so that each subclass inherits the attributes of its superior class. For information on creating your own message classes, see Creating Message Classes.

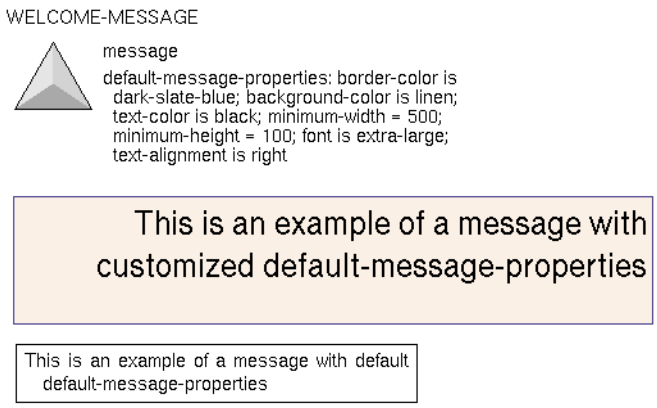
To create an instance of the message class:

➔ Select KB Workspace > New Free Text > message > *message-class*.

where *message-class* is any message class.

Message classes have an initializable system attribute, the default-message-properties attribute, which controls how instances of a message class appear. You can set the default appearance of a message by initializing this attribute appropriately, as described under Specifying Default Message Properties. You can also change message properties programmatically, as described under Using Actions with Messages.

The next figure illustrates a user-defined message class, welcome-message, showing attribute displays of the direct-superior-classes and attribute-initializations attributes of the definition, an instance of that class, and a generic, system-defined message.



Messages do not have any class-specific attributes. They contain a text area that you can type a message into interactively, or use change the text of actions to edit programmatically.

Using Actions with Messages

You can create, manipulate, and delete messages by using actions. When you create a message with a create action, the message is transient and can be deleted with the delete action. You can transfer such messages to a particular workspace, and also change the text, text-color, and background-color by using a change the text of or a conclude action.

Use these actions when working with messages:

- change the color of
- change the text of
- conclude that...
- create
- transfer
- delete

The next sections provide functional descriptions of each of these actions. For a complete description of all G2 actions, see Actions.

Changing the Color Attributes of Message Properties

Messages have three color attributes that you can change programmatically:

- background-color
- border-color
- text-color

You can change one or more color attributes with the change action, as the next two examples show. In the first example, the action changes the text-color color attribute. In the second, use the change the color-pattern of action to change all the message's color attributes.

```
change the text-color of corporate-welcome-message to blue

change the color-pattern of corporate-welcome-message so that
    text-color is red, background-color is yellow, and border-color is purple
```

Changing the Text of a Message

Use the change the text of action to change the contents of a message, as the following example illustrates.

```
change the text of corporate-welcome-message to "new message text"
```

The change action lets you change the text of any free-text or message programmatically.

Concluding Message Text into a Variable or Parameter

You can conclude a new value for a text variable, text parameter, or any attribute that receives a value from a text variable or text parameter, using a *text-expression*. The *text-expression* could be the text of a message, as the next example illustrates.

```
conclude that welcome-text-variable =
    the text of corporate-welcome-message
```

Creating and Transferring Transient Messages

Create and transfer messages with the create and transfer actions. The next example shows you how to create a transient message and transfer it to the workspace:

```
create a welcome-message M;
transfer M to this workspace at (100, 50);
change the text of M to "This is a transient message"
```

Transient messages are deleted automatically whenever you reset the KB.

Deleting Transient Messages

Use the delete action to delete a transient message. The next example deletes every transient message upon a particular workspace.

```
for any welcome-message M upon this workspace unconditionally delete M
```

36

Readout Tables, Dials, and Meters

Describes the display items readout tables, dials, and meters.

Introduction

Readout-tables, dials, and meters, items of the readout-table, dial, and meter classes, respectively, are displays that show values as they change.

Updating each of these display items causes data-seeking, which can establish new valid data and cause event updates to occur.

Caution Do not confuse the display meters described in this chapter with G2-meters. A G2-meter is a special class of quantitative variable you can create to monitor G2 performance and memory utilization, as described in G2-Meters.

The G2 Dynamic Displays utility allows you to add dynamically generated radial, linear, and floating graphical attribute readouts to icons. For more information, see the *G2 Dynamic Displays User's Guide*.

Working with Displays

To create a readout-table, dial, or meter item:

➔ Select KB Workspace > New Display > *display-item*.

where *display-item* is the display of your choice.

Among others, G2 provides these displays:

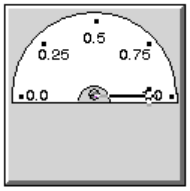
- Readout-table: shows a variable, parameter, or expression, and its value.

the value of integer-param	3345

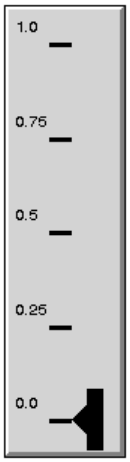
- Digital-clock: a subclass of readout-table that displays the time according to the scheduler.

scheduler time	29 Sep 1999 11:41:06 a.m.

- Dial: represents an arithmetic value graphically. The dial needle moves clockwise as the value increases and counterclockwise as it decreases.



- Meter: represents an arithmetic value graphically. Meters have a bar that rises and falls vertically as the value increases and decreases.



Note Unlike most items, G2 does not display the name of a display adjacent to it on the workspace.

Specifying Tracing and Breakpoints

The tracing-and-breakpoints attribute lets you override the display item's default settings for these facilities.

For more information about using the tracing and breakpoints facilities in G2, see Debugging and Tracing.

Specifying the Display Expression

The expression-to-display attribute specifies the variable or parameter, or expression you want the display to show. Some examples are:

```
Q + 1
the level of tank-1
```

Specifying the Update Interval

The display-update-interval attribute indicates how long G2 waits before updating the display. For example, if you specify 3 seconds, G2 updates the display every three seconds.

Specifying the Display Update after G2 Start-Up

The display-wait-interval attribute specifies the amount of time, after KB start-up, that G2 waits before updating the display.

We recommend that you give readout-tables different display-wait-intervals to stagger updating and even out the CPU load after start-up.

Defining the Update Priority

The display-update-priority attribute defines the display update priority. G2 schedules the execution of tasks according to priority, executing those with the highest priority (1) first.

Specifying Simulated Value Display

The show-simulated-values? attribute determines whether to display the simulated value of the variable referenced in the expression-to-display attribute.

The G2 Simulator, which can supply simulated values, is a superseded capability. For more information, see Appendix F, Superseded Practices.

Common Attributes of Readout Tables, Dials, and Meters

Readout tables, dials, and meter displays each have the following attributes:

Attribute	Description
tracing-and-breakpoints	Use this attribute to override the default tracing and breakpoints for the readout.
Allowable values:	For a complete description of possible tracing and breakpoints values, see System Tables.
Default value:	default
expression-to-display	Specifies the variable, parameter, or expression whose value you want to display.
Allowable values:	any valid G2 expression
Default value:	No default value
display-update-interval	Specifies the frequency with which you want G2 to update the display.
Allowable values:	any positive *time-interval*
Default value:	5 seconds
display-wait-interval	Specifies a time interval that G2 waits, when you start (or restart) your KB, before updating the display.
Allowable values:	any positive *time-interval*
Default value:	2 seconds

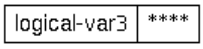
Attribute	Description
display-update-priority	Specifies an integer between 1 and 10 that indicates the priority at which G2 updates the display.
Allowable values:	1 – 10
Default value:	2
show-simulated-values?	Specifies whether a variable referred to in the expression-to-display attribute gets its value from the simulator or inference engine.
Allowable values:	yes no
Default value:	no
Notes:	The G2 Simulator, which can supply simulated values, is a superseded capability. For more information, see Appendix F, Superseded Practices.

Readout Tables

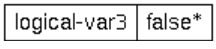
The two kinds of readout tables are: **readout table** and **digital clock**.

A readout table is a rectangular, divided box that shows a label or an expression on the left-hand side, and displays any quantity variable, parameter, or expression on the right, as shown next.

A readout-table causes data-seeking only if the workspace it is on is showing (not hidden). When the expression for the readout table has never been updated, G2 displays **** (asterisks).



When the expression has a value, G2 displays that value. When the value of the expression is no longer valid (because it expires or attempts to update it fail), G2 continues to display the value but displays an asterisk beside it.



Readout tables do not support attribute displays, nor do they update through events. For example, you cannot write a whenever rule that concludes a new value for the readout table. The two ways to update a readout table are:

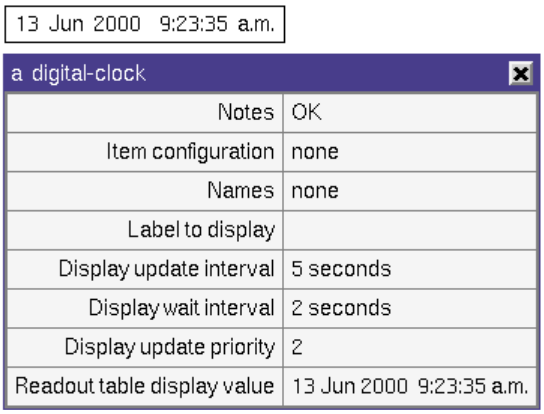
- An update action.
- Scanning, when the readout table is showing.

You cannot disable scanning.

Tip Internally, G2 classifies readout tables as a form of table. If you include a configuration statement for tables in the KB Configuration system table, G2 imposes that configuration on readout tables, too.

Digital Clocks

A digital clock is a special kind of readout table. It displays the value of the expression the current time in the time stamp format. Digital clocks have a subset of a readout-table's attributes.



A digital clock displays the word time when disabled. Note that the value of the current time may drift out of synchronization with the current real time if, for example, the scheduler is running in as fast as possible or simulated time mode. For more information about time expressions, see By Iterating Over a Set.

Specifying the Label to Display

The label-to-display attribute specifies what appears in the left-hand side of the readout display. If you do not enter a value for this attribute, G2 displays the expression-to-display on the left side of the readout-table.

Specifying the Display Format

The display-format attribute specifies which way to display the expression value. Keeping the default value displays the value as it appears for integers, quantities, truth-values, symbols, and text.

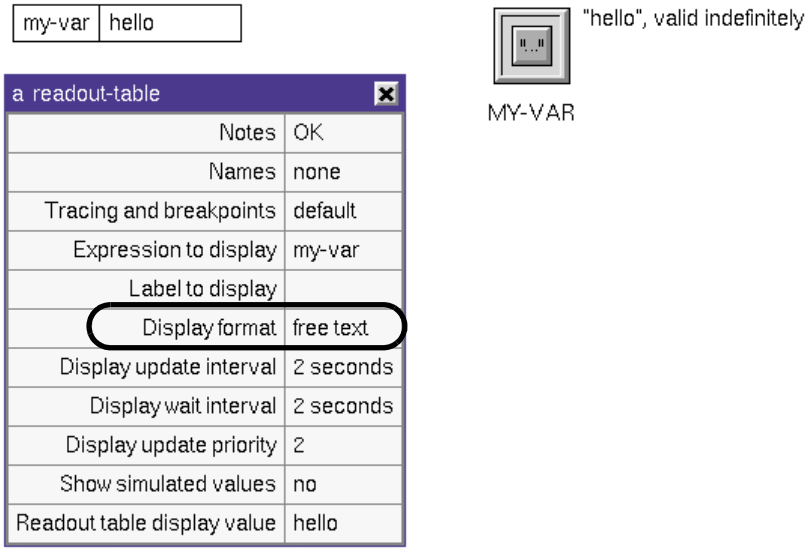
You can specify any of these formatting styles:

- free text to display textual values without quotation marks.
- interval to display quantities as time intervals, such as 1 hour, 30 minutes, and 10 seconds.
- time stamp to display quantities as G2 timestamps, such as 1 Jan 2004 12:00:00 a.m.
- Any *ddd.dddd-format* to display floating point numbers with decimal point precision, where a d represents each digit, such as 123.4567. By default, readout tables display up to three decimal places.
- Any of the following date and time formats:

  formatted as mm-dd-yyyy-hh-mm-ss
  formatted as dd-mm-yyyy-hh-mm-ss
  formatted as yyyy-mm-dd-hh-mm-ss
  formatted as mm-dd-yyyy-hh-mm-ss-am-pm
  formatted as mm-dd-yyyy-hh-mm-am-pm
  formatted as yyyy-mm-dd-hh-mm-ss-am-pm
  formatted as dd-mm-yyyy-hh-mm-ss-am-pm
  formatted as dd-mm-yyyy-hh-mm-am-pm
  formatted as yyyy-mm-dd-hh-mm-am-pm
  formatted as mm-dd-yyyy-hh-mm
  formatted as dd-mm-yyyy-hh-mm
  formatted as yyyy-mm-dd-hh-mm
  formatted as mm-dd-yyyy
  formatted as dd-mm-yyyy
  formatted as yyyy-mm-dd
  formatted as mm-yyyy
  formatted as yyyy-mm
  formatted as dd-hh-mm-ss as an interval
  formatted as hh-mm-ss as an interval
  formatted as hh-mm as an interval
  formatted as mm-ss as an interval
  formatted as hh.hh as an interval

  For examples of the date and time formats, see Formatting Class-Specific Attributes.

The readout table in this example sets the value of the text variable named my-var. Notice that the readout table does not use quotation marks. To accomplish this, the format-for-type-in-box and display-format is configured as free text.



Reading the Current Value

The readout-table-display-value attribute indicates the most recent value computed for the display. You cannot change this attribute; it is provided by G2.

G2 does not update a readout table automatically, but according to its display-update-interval attribute specification.

Class-Specific Attributes of Readout Tables

Readout tables have the same attributes as other displays, as described in Common Attributes of Readout Tables, Dials, and Meters. The class-specific attributes of readout tables are:

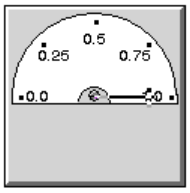
Attribute	Description
label-to-display	Specifies the label shown on the left side of the readout-table.
Allowable values:	Any symbolic name
Default value:	none

Attribute	Description
display-format	The format in which to display the value of the Expression-to-display.
Allowable values:	any *ddd.dddd-format* free text interval time stamp default
Default value:	default
readout-table-display-value	The most recent value computed for the display.
Allowable values:	The current value
Default value:	****

Dials and Meters

Dials and meters, items of the dial and meter classes, show the value of any quantity variable, parameter, or expression as a pointer on the display. Both items cause data-seeking if the workspace upon which they reside is not hidden.

A dial is a semi-circular clock-like display containing a pointer and five numeric values, ranging from the far left-hand side to the right. You set the range of the dial values in the attribute table. The pointer moves clockwise from left to right as the expression value increases, and reverses as it decreases.



A meter is a vertical measurement display. The meter also has five numeric values, along with a bar that moves up and down as the expression value increases or decreases.

Display meters are not the same as G2-meters. A G2-meter is a special class of quantitative variable you can create to monitor G2, as described in G2-Meters. However, you could use a display meter to show the value of a G2-meter.

Setting the Meter's Lower Value

The low-value-for-dial-rule attribute (low-value-for-meter-ruling for meters) sets the lowest value for the ruling of the dial or meter.

Determining the Meter's Dial Increment

The increment-per-dial-ruling attribute (increment-per-meter-ruling for meters) determines the increment of the dial or meter rulings.

For example, if the low-value-for-dial-ruling is 0 and the increment-per-dial-ruling is 10, then the dial or meter will have the rulings: 0, 10, 20, 30, 40.

Class-Specific Attributes of Dials and Meters

Dials and meters have functionally identical attributes, though the names differ slightly depending on whether the item is a dial or a meter. Common attributes are described in Common Attributes of Readout Tables, Dials, and Meters. The class-specific attributes of dials and meters are:

Attribute	Description
low-value-for-dial-ruling **low-value-for-meter-ruling**	Specifies the lowest calibration (ruling) value for the dial or meter.
Allowable values:	Any dial- or meter-ruling-parameter
Default value:	0.0
increment-per-dial-ruling **increment-per-meter-ruling**	Specifies the increment to use for dial and meter rulings.
Allowable values:	Any dial- or meter-ruling-parameter
Default value:	0.25

37

Freeform Tables

Describes how to use freeform table display items.

Introduction

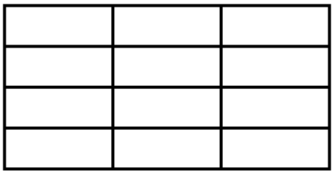
A freeform table, an item of the freeform-table class, is a tabular display that is similar to a spreadsheet. The table consists of cells, which are arranged in rows and columns. By default, freeform tables do not cause data seeking.

Creating a Freeform Table

To create a freeform table:

➔ Choose KB Workspace > New Display > freeform-table.

By default, G2 creates a freeform table having four rows and three columns, as shown in the next diagram.



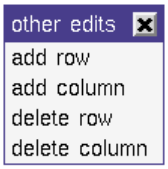
Specifying the Table Size

The table-size attribute specifies the size of the freeform table as a number of rows and columns, which you enter as follows:

```
the number-of-rows = 4;
the number-of-columns = 3
```

where 4 and 3 are the defaults that you can change. Increasing these values adds new rows to the bottom of the table and new columns to the right-hand side. Decreasing these values removes rows and columns from the same locations.

You can also change the size of the freeform table by choosing other-edits from the item menu and then one of the choices shown below:



When you select add column, G2 adds a column to the left; when you select add row, G2 adds a row above the cell from which you chose the other edits menu. By design, these choices are the opposite of adding rows and columns via the table-size attribute, thus providing a means for you to add rows and columns in either direction.

Specifying Default Formats for Table Cells

The default-cell-format attribute specifies a default set of formats for all table cells. You can specify all of the formatting options that Changing Formatting Attributes describes. An example is:

```
the text-color is green
```

Determining the Default Evaluation Settings

The default-evaluation-setting attribute controls how G2 evaluates an expression. An example is:

```
evaluated every 2 seconds at priority 5
```

You can specify all of the evaluation settings that Changing Evaluation Settings describes.

Note You cannot use the change the text of action to change the default-evaluation-setting attribute value.

Formatting Freeform Tables

You can format freeform tables by setting defaults, which affect every table cell, or by setting cell values individually. Individual cell values override default values.

Additionally, freeform tables have two special kinds of attributes:

- **formatting attributes**: For formatting various visual aspects of the freeform table using one or more statements.
- **evaluation settings**: For specifying one or more statements about how G2 evaluates a freeform table cell expression, using one or more statements.

G2 provides a set of formatting and evaluation setting values that are inherent with every freeform table. While the default formatting and evaluation setting values are not visible, they exist as part of that item's knowledge. For a list of these default values, see Changing Formatting Attributes and Changing Evaluation Settings later in this chapter.

To set default values for a freeform table, click on the freeform table and open the freeform table's attribute table, shown next.

a freeform-table

Notes	OK
Item configuration	none
Names	none
Table size	the number-of-rows = 4; the number-of-columns = 3
Default cell format	none
Default evaluation setting	none

To provide a cell expression and set any individual cell values:

➔ From the freeform table's menu choose:

- edit cell expression to provide an expression and, optionally, set evaluation settings.
- edit cell format to set formatting attributes).

To add and delete freeform table rows and columns:

➔ Choose the other edits menu option.

Expressions for Freeform Table Cells

Each table cell displays the value of an expression. You enter the expression using the edit cell expression option from the freeform table's menu.

G2 updates the expression that a freeform table cell displays only when it changes. If you are displaying a parameter value, for example, that value may only change when an event update occurs.

Using evaluation settings, you can optionally configure either the entire freeform table or an individual cell expression to use scanning, event updating, and data seeking, to name just some of the evaluation options.

Several expression examples are:

```
the status of tank-2
the current time; evaluated every 2 seconds
the level of tank-2; evaluated with these settings: may-request-data-seeking is true;
scan-interval is 4
```

For descriptions and examples of all of the evaluation settings you can specify, see Changing Evaluation Settings.

Note You cannot refer to the values of expressions in freeform tables.

Changing Formatting Attributes

Formatting attributes let you control the visual appearance of a freeform table. You set default values for formatting attributes in the default-cell-attribute of the freeform table, and individual cell values by choosing the edit-cell-format menu option when you click in any table cell. You can use all of the formatting attributes to format individual cells except for border-color, height, and width, which are applicable only in the default-cell-format attribute.

The names and descriptions of the freeform table formatting attributes, along with their default values, are:

This attribute...	Has this default...	And specifies...
background-color	transparent	Any color, meta-color, or a symbolic expression as the freeform table's background color.
border-color	foreground	Any color or meta-color as the border color.
height	35	An integer as the height (in pixels) of the freeform table's cells.
width	150	An integer as the cell width (in pixels).
text-alignment	right	The value left, center, or right as the alignment for text within cells.
text-color	foreground	Any color or meta-color as the text color of cells.
text-size	small	The cell font size as small, large, or extra-large.

Formatting attributes for an individual cell override the default value. For example, if the default-cell-format attribute specifies the background-color as yellow, and you edit the background-color of an individual cell to:

```
the background-color is blue
```

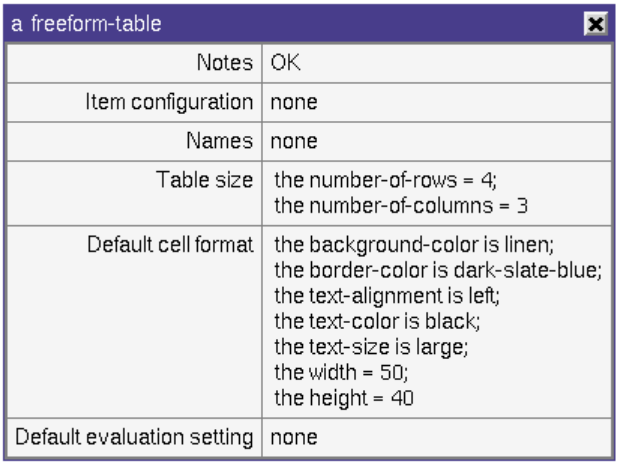
all the freeform table cells will have a yellow background, except for the cell whose format you edited, which will have a blue background.

When entering formatting attributes, the Text Editor includes in its freeform table prompts the line-color is attribute, which is applicable only to charts. Also, the Text Editor does not prevent you from entering a formatting attribute more than once. If you do so, however, G2 uses the last one. For example, if you enter both of these statements in the default-cell-format attribute:

```
the background-color is blue;
the background-color is red
```

G2 discards the first statement and uses only the last statement; the background color of every cell would then be red.

Here is part of a freeform table's attribute table after you set every available formatting attribute from the default-cell-format attribute:



a freeform-table

Notes	OK
Item configuration	none
Names	none
Table size	the number-of-rows = 4; the number-of-columns = 3
Default cell format	the background-color is linen; the border-color is dark-slate-blue; the text-alignment is left; the text-color is black; the text-size is large; the width = 50; the height = 40
Default evaluation setting	none

Changing Evaluation Settings

Evaluation settings control the way G2 evaluates freeform table cell expressions.

This section describes each of the available evaluation settings, which affect several activities, including, but not limited to:

- Data seeking
- Event updating
- Scanning

Entering Evaluation Settings

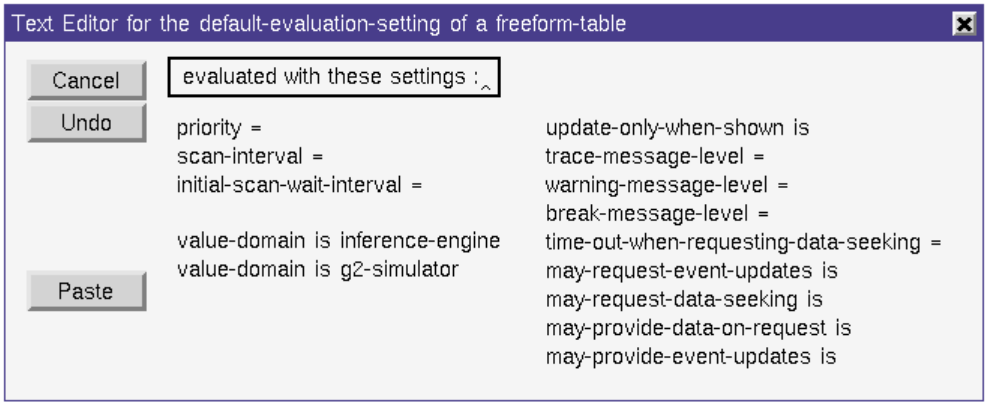
You can set one or more default evaluation settings in the freeform table's default-evaluation-setting attribute, or tailor the evaluation settings for individual cells, using the edit cell expression menu option from any table cell, and entering one or more statements after the cell expression.

The ways to enter evaluation settings are:

To provide....	Enter a statement like this...
A priority level of your choice.	evaluated at priority 3
A specific evaluation time interval that you provide.	evaluated every 10 seconds

To provide....	Enter a statement like this...
A priority level and evaluation time interval of your choice.	evaluated at priority 3 every 10 seconds
One or more evaluation attributes as shown in the next figure.	evaluated with these settings:
A specific evaluation time interval, and one or more evaluation settings.	evaluated every 10 seconds with these settings:

The evaluation settings you can enter after the evaluated with these settings: statement are:



Not all of the evaluation settings that appear in the Text Editor prompt are applicable for freeform tables. The names and descriptions of the valid evaluation settings, along with their default values, are:

This evaluation setting...	Has this default...
priority	2
scan-interval	none
initial-scan-wait-interval	5 seconds
value-domain	inference-engine
update-only-when-shown	true
trace-message-level	none
warning-message-level	none

This evaluation setting...	Has this default...
break-message-level	none
timeout-when-requesting-data-seeking	infinite
may-request-event-updates	true
may-request-data-seeking	false

Data Seeking Evaluation Settings

Data seeking can occur within a KB whenever G2 encounters an unknown variable value. Backward chaining is one kind of data seeking, obtaining data from a data server is another. There are many ways to cause data seeking including:

- Setting a g2-variable's update-interval attribute.
- Showing a display (such as a readout-table) that needs a value.
- Executing a rule that needs a value.
- Executing a wait statement in a procedure that tests a waiting predicate.

These evaluation settings control when G2 seeks data:

Evaluation setting	Description
may-request-data-seeking	Enables or disables a computation's permission to request data seeking when it encounters an unknown value. Set this attribute to true to enable permission to data seek or false to disable it.
timeout-when-requesting- data-seeking	Specifies a time limit on data seeking for a value. When data seeking times out, G2 notifies the requesting computation.

Using Data Seeking Evaluation Settings

The following example shows you how to use evaluation settings to allow or disallow data seeking. Consider a freeform table cell that contains the expression X+Y. If G2 attempts to update the value of this expression and encounters unknown values for X and Y, it may request data seeking if the cell expression contains the following evaluation setting:

```
X + Y;
evaluated with these settings:
may-request-data-seeking is true
```

This allows G2 to data seek for values for X and Y. Note that you could also have specified the evaluation setting may-request-data-seeking as a value in the default-evaluation-setting attribute of the freeform table's attribute table.

Consider the quantitative-variables X and Y. In this example, the values for these variables are unknown. Given the above evaluation setting, G2 attempts to data seek for values. The attribute tables for both X and Y are shown below.

X, a quantitative-variable

Options	do not forward chain, breadth first backward chain
Notes	OK
Item configuration	none
Names	X
Tracing and breakpoints	default
Data type	quantity
Initial value	none
Last recorded value	no value
History keeping spec	do not keep history
Validity interval	supplied
Formula	the current time + 2
Simulation details	no simulation formula yet
Initial value for simulation	default
Data server	inference engine
Default update interval	1 second

Y, a quantitative-variable

Options	do not forward chain, breadth first backward chain
Notes	OK
Item configuration	none
Names	Y
Tracing and breakpoints	default
Data type	quantity
Initial value	none
Last recorded value	no value
History keeping spec	do not keep history
Validity interval	supplied
Formula	none
Simulation details	no simulation formula yet
Initial value for simulation	default
Data server	inference engine
Default update interval	1 second

Because the options attribute is set to do not backward chain, the X variable may not provide data on request to G2. Because X will not accept the data seeking (backward chaining) request, G2 is unable to obtain a new value for X.

For the Y variable, however, the options attribute is set to depth first backward chain (it may also be breadth first backward chain). Y accepts G2's data seeking request, evaluates its formula (the current time), and then provides a new value.

Event-Updating Evaluation Settings

When one part of G2 notifies another of an event, the notification process is called **event-updating**. Forward chaining is one example of event updating. When values change, they may provide events that allow expressions, depending upon those values, to be updated. You can control this process by using event updating evaluation settings.

When a value changes, it sends an event to those expressions relying on the value and informs them of the event. Expressions can request event updates, so that when these values change, the expressions automatically receive the forward chaining event to get the new value. With the following evaluation settings, you can control when G2 uses events to update values:

Evaluation setting	Description
may-request-event-updates	Enables or disables a computation's permission to request that locations it references provide it with updates.

Scanning Evaluation Settings

G2 allows you to configure some computations to occur periodically by giving them a scan interval. By using the following evaluation settings, you can control how often and when G2 attempts to evaluate an expression:

Evaluation setting	Description
scan-interval	Specifies an interval (in seconds) that G2 waits between evaluations of a scanned computation. If the value of this attribute is none, no scanning occurs.
initial-scan-wait-interval	Specifies the initial time interval that G2 waits immediately following activation before executing the first scanned evaluation. G2 only waits the initial-scan-wait-interval once, and this allows you to stagger the scan intervals for expressions, making better use of resources.

Debugging and Tracing Evaluation Settings

These attributes control the interactions between G2, when it attempts to evaluate expressions, and the tracing and debugging facilities. G2 provides the following debugging and tracing evaluation settings:

Evaluation setting	Description
break-message-level	Controls the breakpoint for a particular item; the default value is the value found in the Debugging Parameters system table.
trace-message-level	Controls the tracing messages for a particular item; the default value is the value found in the Debugging Parameters system table.
warning-message-level	Controls the warning messages for a particular item; the default value is the value found in the Debugging Parameters system table.

The values for these attributes are the same as you would specify in the Debugging Parameters system table. However, specifying debugging and tracing evaluation settings for a particular item allows you to override the setting in the system table.

Note In the Debugging Parameters system table, the tracing-and-breakpoints-enabled? attribute may be set to no, in which case G2 disables all debugging and tracing attributes (those specified in the system table and those specified as debugging and tracing evaluation settings). For more information about debugging and tracing, see Debugging and Tracing.

Scheduling Evaluation Settings

By using the scheduling evaluation setting, you can prioritize expressions. G2 evaluates those expressions of higher priority before evaluating those of lower priority. In G2, 1 is the highest priority and 10 is the lowest. Use the following scheduling evaluation setting to set priorities for expressions:

Evaluation setting	Description
priority	Sets the priority for the expression, which G2 uses in determining when to evaluate it. A value of 1 is the highest priority; however, you can specify any number from 1 through 10. Note that you cannot use this attribute to change the order of evaluation. You can use it only to affect how soon a given expression is evaluated when its task reaches the top of the task queue.

Other Evaluation Settings

G2 provides a value-domain attribute for specifying where G2 gets the values for the expressions it is evaluating, and an update-only-when-shown attribute to suppress updating when the expression is not visible. These attributes are described below:

Evaluation setting	Description
value-domain	Specifies where G2 gets the values when evaluating expressions. The possible values for this attribute are inference-engine or g2-simulator. The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices.
update-only-when-shown	Suppresses updating if the expression is not visible. This functionality is similar to readout tables, which do not data seek unless they are visible on the workspace.

Changing Freeform Tables Programmatically

You can use the attribute access facility to update any part of a freeform table programmatically. For example, the next procedure accepts as its arguments any freeform table, and the number of columns and rows to which the table should be sized. The procedure also sets several of the evaluation settings.

The attribute descriptions of freeform tables, presenting their internal structure, is presented in the *G2 Class Reference Manual*.

```
update-freeform-table (F: class freeform-table, columns: integer, rows: integer)
evaluation-attributes, size: structure;
begin
    size =
        structure(number-of-columns: columns, number-of-rows: rows);
    evaluation-attributes =
        structure(priority: 4, may-request-event-updates: true,
                  may-provide-data-on-request: true,
                  may-provide-event-updates: true);
    conclude that the table-size of F = size;
    conclude that the default-evaluation-setting of F =
        evaluation-attributes
end
```

The Freeform Table Class

The class-specific attributes of freeform tables are:

Attribute	Description
table-size	The number of columns and rows in the freeform table.
Allowable values:	1 – 100 (for both number-of-rows and number-of-columns)
Default value:	the number-of-rows = 4; the number-of-columns = 3
default-cell-format	Specifies the default formatting attributes that G2 uses for table cells.
Allowable values:	See description following table.
Default value:	none

Attribute	Description
default-evaluation-setting	Specifies the default evaluation settings (if any) which each cell expression uses.
Allowable values:	See Changing Evaluation Settings.
Default value:	none

38

Charts

Presents chart styles and graphs, and show you how to use them.

Introduction

Charts, instances of the chart class, are display items that plot numerical data in the form of one or more data series. A data series consists of one or more quantities, which must be elements of a quantity list or quantity array.

Charts do not cause data seeking and have no means of updating automatically. Instead, you use the update action to plot the data series on a chart. Unlike most items, charts do not support attribute displays.

Note Do not confuse charts with trend charts, which plot the history values of one or more variables and parameters over time. Trend charts are described in Trend Charts.

Using Charts

To create a chart:

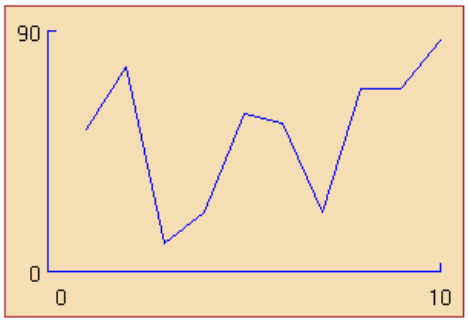
➔ Select KB Workspace > New Display > chart.

Chart Styles

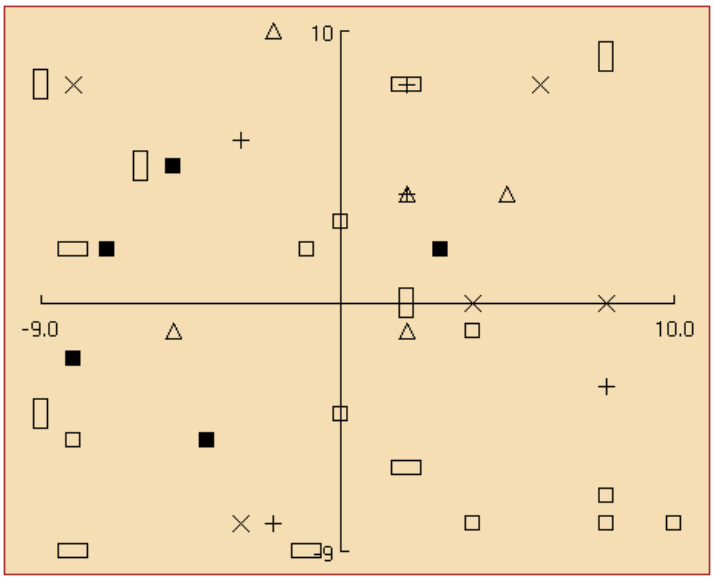
There are three system-defined chart styles:

- Line chart
- Scatter chart
- Column chart

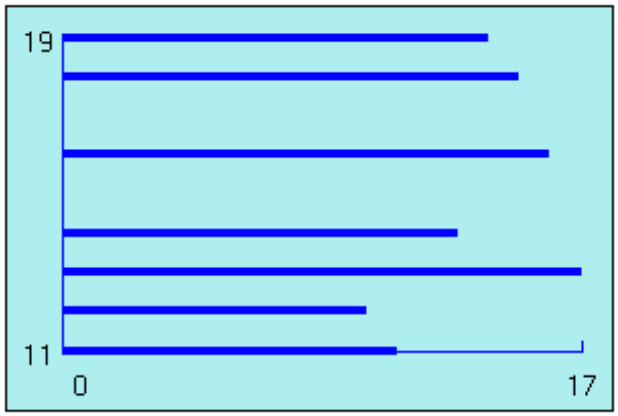
The chart-style attribute of a chart determines its style, each style providing a set of default attributes. The following diagram shows an example of each chart type. Chart attributes are described in The Chart Class.



Line-chart-style



Scatter-chart-style



Column-chart-style

Specifying the Chart Style

The chart-style attribute specifies what style of chart to use. You can further enhance the chart style by using annotations. For example, when using a column-chart-style, you can optionally add an annotation indicating that a data series should be represented as either a bar (horizontal) or a column (vertical) style.

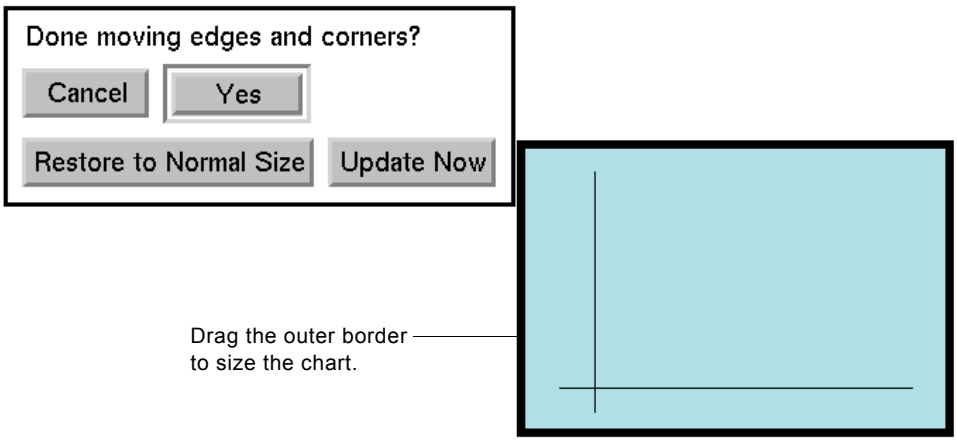
Sizing a Chart

G2 limits the size of a chart so that it cannot become so large that it consumes all available system resources.

To change the size of a chart:

➔ Select change size from the chart's menu.

G2 draws a size box around the chart and displays update dialog, shown here. Move the edges of the size box so that the chart is the size you want and click Update Now in the dialog.



Defining the Data Series for the Chart

The data-series attribute defines what to plot on the chart. Each data series must evaluate to a quantity list or array.

The order in which you enter each data series determines its numeric reference, which you use in statements. For example, if you enter two data series in this attribute, you would refer to them in an expression with a statement such as:

```
data series 1, data series 2
```

where the data series number corresponds to the order in which you entered them.

By using the *versus* statement, you can plot two data series against each other. An example of plotting two data series is:

```
plot q1; plot q2 versus q3
```

where *q1*, *q2*, and *q3* are each quantity-list or quantity-array items.

Displaying and Updating a Chart

To display one or more plots on a chart, you use the update action. Because charts do not have a scan interval, G2 does not automatically update a chart when an element of the quantity-list (or quantity-array) for the chart changes. Instead, you must use the update action whenever you want to update a chart. Some examples are:

```
update power-series-chart
update every chart upon this workspace
```

Note Updating a chart does not cause data seeking or event updating. A chart is updated once when activated and once whenever it is edited.

Using Chart Annotations

After creating a chart and determining its style, you can optionally format the chart using **annotations**.

Charts are composed of several components:

- Axis
- Chart
- Data series
- Data point

Each component consists of formatting attributes that define the chart's appearance. For example, the axis component includes the axis-minimum, axis-maximum, and axis-crossover attributes. Formatting attributes are a special kind of attribute used only in formatting freeform tables and charts. Annotations are statements that describe the value of one or more formatting attributes of a chart component.

Note When referring to component attributes, this chapter uses the term *attribute* somewhat loosely. Unlike most other item attributes, a chart's component attributes are not accessible as individual attribute names on attribute tables. Instead, you provide values for these attributes through annotation statements.

Unlike trend chart components, which are organized into component subtables, chart components are accessible only as annotations. Syntactically, chart annotations are almost identical to those found on trend charts. They differ, however, in their organizational and descriptive statements.

Trend chart annotations explicitly state the components to which they refer. They begin with a component statement, followed by one or more component attributes and values, such as:

```
1 value-axis;
the label-color of value-axis 1 is red
```

indicating that there is one value axis, and that its label-color attribute has a value of red.

In contrast, chart annotations do not explicitly state the component to which they refer. The attribute name indicates the relevant component, as these annotations do:

```
the grid-color of axis 1 is purple;
the indicator of any data-point of data-series 1 is solid-column
```

In this example, grid-color is a chart component attribute, and indicator is an attribute of the chart's data point component.

While you can enter chart annotations in any order, G2 may regroup certain attributes so that they appear together for one component.

Default Chart Annotations

Each chart style that G2 provides includes a set of default values for each of the components. Though you cannot see the default values as annotations in the chart's attribute table, the defaults exist as part of that item's knowledge. The component attribute defaults for each chart style are shown next. The notation N/A indicates that the default is not applicable.

		Chart Style		
Component	**Attribute**	**Line-Chart**	**Scatter-Chart**	**Column-Chart**
all	line-color	black	black	black
axis	axis-minimum (1)			
	axis-maximum (1)			
	axis-crossover	zero	zero	zero
	number-of-significant-digits	-1	-1	-1
	number-of-tickmarks			
	tickmark-interval			
chart	background-color	transparent	transparent	transparent
	border-color	black	black	black
	grid-color	black	black	black
	grid-visible	false	false	false
data-series	connection-line-visible	true	false	false
	line-from-last-to-first-point-visible	false	false	true
data-point	height	5	5	N/A
	indicator	square	square	bar
	indicator-visible	false	true	true
	width	5	5	5

Shaded component attributes without values are those that have an unspecified default. For these attributes, G2 calculates an appropriate value based upon the data that the chart is plotting.

Axis Component Attributes

Here are the axis component attributes.

Specifying the Minimum and Maximum Axis Values

The axis-minimum and the axis-maximum attributes specify a number on the axis that shows the beginning and end of the range of data. Axis 1 is the horizontal axis and axis 2 is the vertical axis. An example is:

```
the axis-minimum of axis 1 = -20
the axis-maximum of axis 1 = 10
```

Specifying Where Axes Cross

The axis-crossover attribute specifies a point on the axis where the other axis crosses it. The point is an integer, whose default value is zero. If zero is not on the axis, the default is the axis-minimum. If you specify an axis-crossover value that is outside the axis range, G2 overrides it and uses the axis-minimum value. An example is:

```
the axis-crossover of axis 1 = 4.7
```

Specifying the Number of Significant Digits of Tickmark Labels

The number-of-significant-digits specifies whether tickmark-label quantities should be displayed using exponential representation. The specified value can be an integer between 2 and 15 or it can be -1.

The default value of -1 specifies that floating point and integer values for labels be converted to exponential representation under these conditions:

- When the quantity is an integer value over six digits and its exponential representation is shorter than its integer representation.
- When the quantity is a floating-point value and its exponential representation is shorter than its floating-point representation.

Providing a value between 2 and 15 overrides the default behavior by specifying that the label quantity should not be converted to exponential representation, unless the value specified is not large enough to accurately represent the quantity. An example is:

```
the number-of-significant-digits of axis 2 = 13
```

Specifying the Number of Tickmarks on an Axis

The number-of-tickmarks attribute specifies the number of tickmarks on the axis. An example is:

```
the number-of-tickmarks of axis 1 = 4
```

Specifying the Tickmarks Interval

The tickmarks-interval attribute specifies the distance between successive tickmarks on an axis. An example is:

```
the tickmark-interval of axis 2 = 5
```

Chart Component Attributes

These are the chart component attributes.

Setting the Background Color

The background-color attribute determines the background color of the entire chart. Unlike trend charts, there is no way to separate the background color of the actual data window, in which the data series values display, from the outer edge of that area. The default background color is transparent.

Setting the Border Color

The border-color annotation determines the chart's external border color. There is no annotation to set the border that bounds the data window area, in which the data series values appear. The default is black.

Setting the Grid Color

The grid-color attribute specifies the color of the grid for either axis. Conceptually, a grid is an extension of an axis tickmark to the full size of the chart. You can specify a grid-color for both axes, or for either axis individually. Some examples are:

```
the grid-color is light-blue
the grid-color of axis 1 is pale-green
```

Specifying Whether Grid Lines are Visible

The grid-visible attribute turns grid lines on or off (true and false, respectively) for either or both axes. Some examples are:

```
the grid-visible is true
the grid-visible of axis 2 is false
```

Data Point Component Attributes

These are the attributes that compose the data point component.

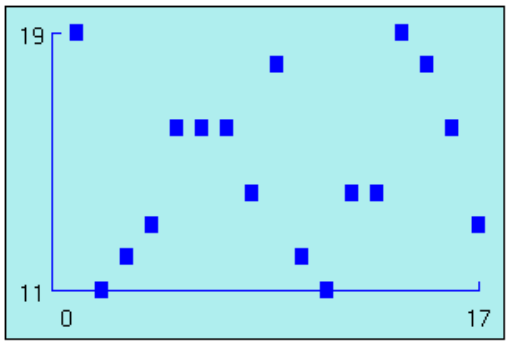
Specifying the Height and Width of Data Points

The height and width attributes specify the height and width of a data point in pixels. Enter this attribute as an integer value. The height and width attributes are

not applicable to line charts. The default value varies depending on the indicator you use. To change the default, enter an annotation like this:

```
height of any data-point = 10;
width of any data-point = 10;
```

The following diagram shows the effect of changing the data-point height and width from the default value to 10:



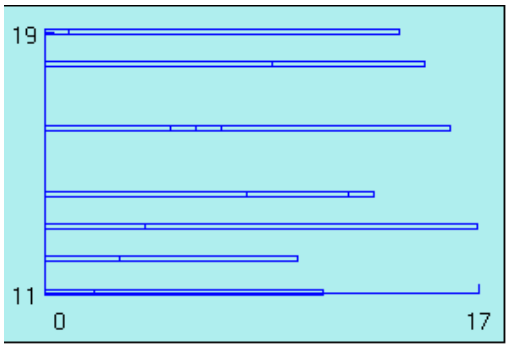
Defining the Indicator Type

The indicator attribute defines what G2 displays at each data point in a data series. The next table lists the possible values for this attribute and the chart style they are appropriate for.

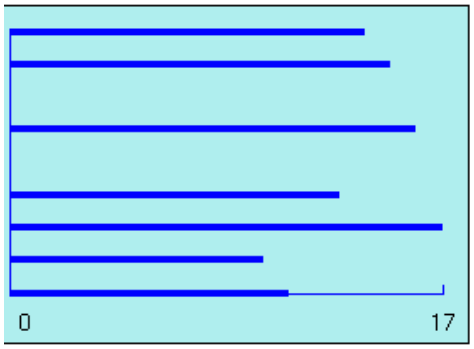
Indicator Value	Column Chart Style	Scatter Chart Style
Square		✓
Rectangle		✓
Triangle		✓
Cross		✓
X		✓
Bar	✓	
Column	✓	
Solid-bar	✓	
Solid-column	✓	

Hint Adding an indicator annotation to a line-chart effectively changes the chart style to a column- or scatter-chart, depending on the value you specify.

Here are examples of an indicator of bar and solid-bar:

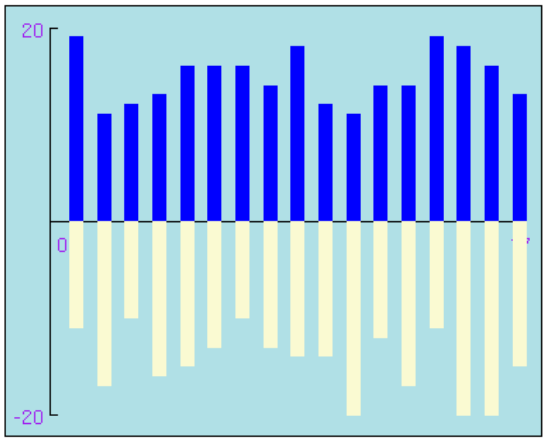


The indicator of any data-point is bar.

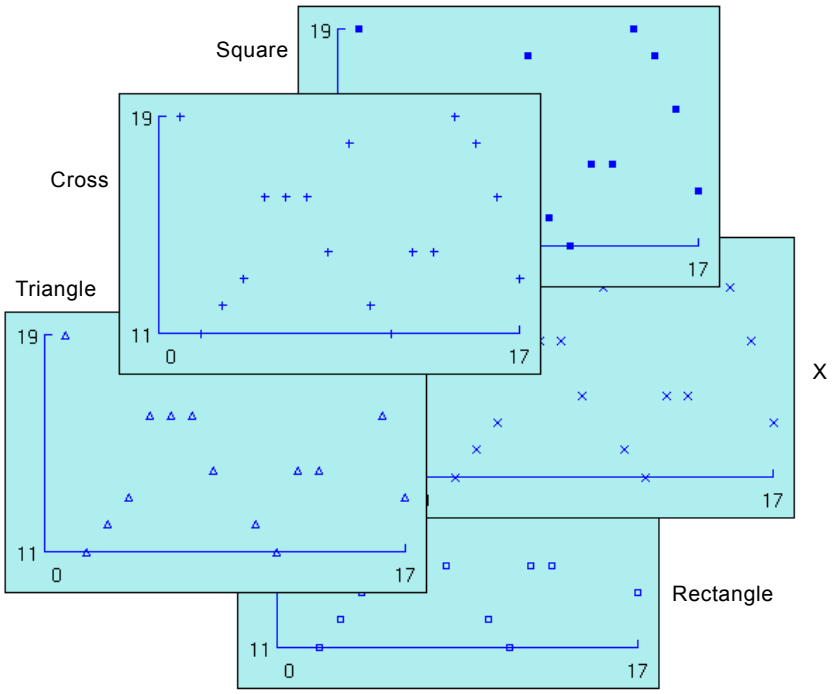


The indicator of any data-point is solid-bar.

The next diagram shows a column chart with solid-column data-point indicators for two data series. The chart appears with upper and lower values since the list element values it represents are in opposite numeric ranges (random 10 to 20, and random -20 to -10).



Here are examples of each of the indicator attribute values appropriate for scatter charts:

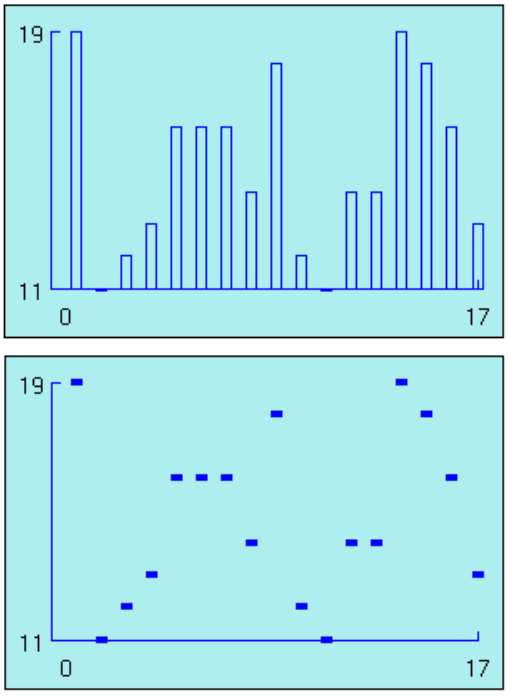


As the previous table notes, indicator attribute values are *appropriate* for scatter or column chart types. The term appropriate, however, does not indicate that each value is exclusive to a chart type. On the contrary, changing the indicator attribute on a scatter or column chart effectively changes the chart style.

Consider the following column charts. The left-hand chart is one without a specific indicator attribute (the default is bar). The right-hand chart shows the effect of adding the annotation:

```
the indicator of any data-point is square
```

Such an annotation essentially changes the appearance of the column chart into a scatter chart:



You can specify a separate indicator for each data series on the chart, but only one indicator can be in effect at a time. If a chart includes one indicator annotation and you add another, closing the edit on the change causes G2 to delete the first annotation and keep the last change.

Two examples of indicator attribute annotations are:

```
the indicator of any data-point is rectangle
```

```
the indicator of any data-point is bar-column
```

Specifying Whether the Indicator is Visible

The indicator-visible attribute specifies whether the indicator is visible. By default, an indicator is visible (true). Changing the indicator-visible attribute to false effectively hides the data-series display. For example:

```
the indicator-visible of any data-point of data-series 1 is true
```

Data Series Component Attributes

Here are the attributes that comprise the data series component.

Specifying Whether Data Points are Connected

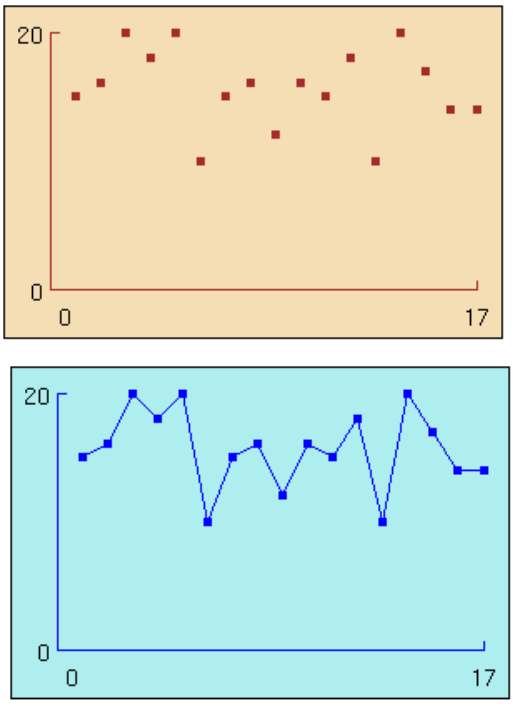
The connection-line-visible attribute specifies whether there is a line between data-points in a data series. You can set this attribute to true or false. The connection-line-visible attribute is not applicable to column charts.

An example of this annotation is:

```
the connection-line-visible of any data-point is true
```

For a line chart, this attribute is true by default. Changing it to false effectively hides all data points.

The next diagram shows the effect of changing the connection-line-visible attribute to true upon a scatter chart:



Controlling Connecting Lines

The line-from-last-first-point-visible attribute controls the appearance of a single connecting line from the last point in the data series to the first point. Use this when plotting data that samples one cycle of a cyclic process. Set this attribute to true or false. An example is:

```
the line-from-last-to-first-point-visible of any data-point is false
```

Defining the Line Colors

The line-color attribute is applicable to several components, including the chart's axis, data point and data series components.

You can specify the line color for a chart component attribute as a color or a meta-color as follows:

```
the line-color of axis 1 is purple;
the line-color of data-series 1 is red;
the line-color of any data-point is brown
```

Updating Charts Programmatically

You can use the attribute access facility to update chart-style, data-series, and annotations of a chart programmatically.

The attribute descriptions of charts, presenting their internal structure, is presented in the *G2 Class Reference Manual*.

This code example concludes the chart-style attribute.

```
conclude that the chart-style of series-chart = the symbol column-chart-style
```

The Chart Class

The class-specific attributes of charts are:

Attribute	Description
chart-style	The style of the chart.
Allowable values:	{line-chart-style | scatter-chart-style | column-chart-style}
Default value:	line-chart-style
data-series	One or more data series that you want to plot on the chart.
Allowable values:	{none | plot {*g2-list* | *g2-array*} [,...] [versus plot {*g2-list* | *g2-array*}]}
Default value:	none

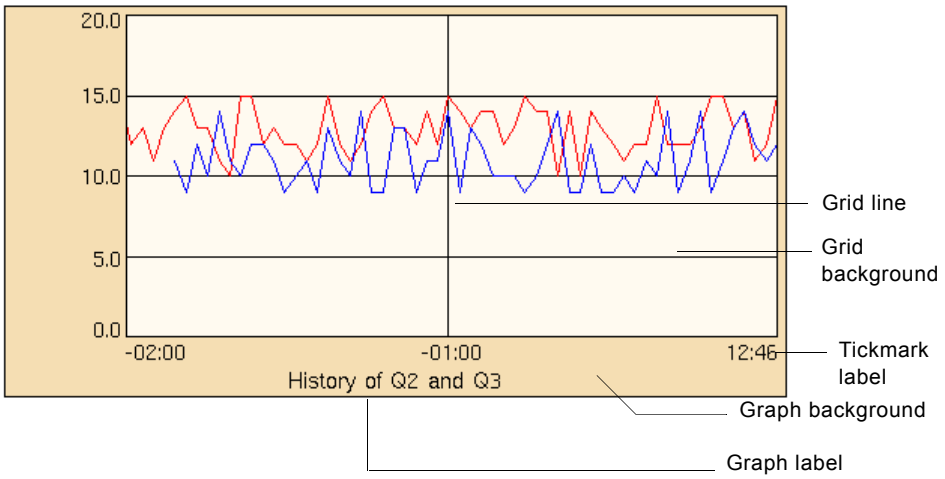
Attribute	Description
annotations	A syntactical description of how to format the chart. A chart can have no annotations, or a great many of them describing multiple chart attributes. Using Chart Annotations describes the possible values of this attribute in detail.
Allowable values:	Any applicable chart annotations.
Default value:	none

Graphs

Presents chart styles and graphs, and show you how to use them.

Introduction

A **graph** plots the histories of one or more quantitative variables and parameters. Here is a graph with its various attributes labeled:



Hint To graph the histories of variables and parameters, you must complete the history-keeping-spec attribute of those items in their corresponding attribute tables.

Creating a Graph

To create a new graph:

➔ Select KB Workspace > New Display > graph.

The class-specific the attributes of graphs are:

Attribute	Description
desired-range-for-horizontal-axis	The length of time that the graph represents. The maximum value for this attribute is 24 weeks.
Allowable values:	{none | *time-interval* | any *time-interval* to *time-interval* [without tickmark labels]
Default value:	none
desired-range-for-vertical-axis	The range of values that the graph plots.
Allowable values:	{none | *number* [to *number*]} [with all intervals the same | without tickmark labels]
Default value:	none
scroll-continuously?	Determines whether the values in the graph data window scroll to display new data.
Allowable values:	yes no
Default value:	no

Attribute	Description
percentage-extra-space-to-leave	The percentage of data points to display before scrolling.
Allowable values:	Any integer
Default value:	0
show-grid-lines?	Determines whether to display grid lines in the grid background.
Allowable values:	yes no
Default value:	yes
interval-between-horizontal-grid-lines?	Determines the interval between the tickmarks on the vertical axis.
Allowable values:	{compute automatically | number beginning at number}
Default value:	yes
extra-grid-lines?	Determines the color and placement of extra grid lines.
Allowable values:	at any *number* in *any-color* none
Default value:	none
background-colors	Determines the background colors for the graph and the grid.
Allowable values:	graph background: *color*, grid background: *color*
Default value:	graph background: white, grid background: white

Attribute	Description
expressions-to-display	The expressions whose value the graph represents.
Allowable values:	Any expression that evaluates to a variable or a parameter.
Default value:	No value
label-to-display	The label that appears below the graph.
Allowable values:	Any label of your choice
Default value:	No value
display-update-interval	The frequency with which you want G2 to update the graph.
Allowable values:	*time-interval*
Default value:	5 seconds
display-wait-interval	The interval of time that G2 waits before updating the graph whenever you start or restart the KB.
Allowable values:	*time-interval*
Default value:	2 seconds
display-update-priority	The priority at which G2 evaluates the graph expressions.
Allowable values:	1 – 10
Default value:	2

Attribute	Description
show-simulated-values	Determines whether to show a variable's simulated value, rather than its non-simulated value.
Allowable values:	yes no
Default value:	no

Tip Internally, G2 classifies graphs as a form of table. If you include a configuration statement for tables in the KB Configuration system table, G2 imposes that configuration on graphs, too. Also, like some other display items, graphs do not support attribute displays.

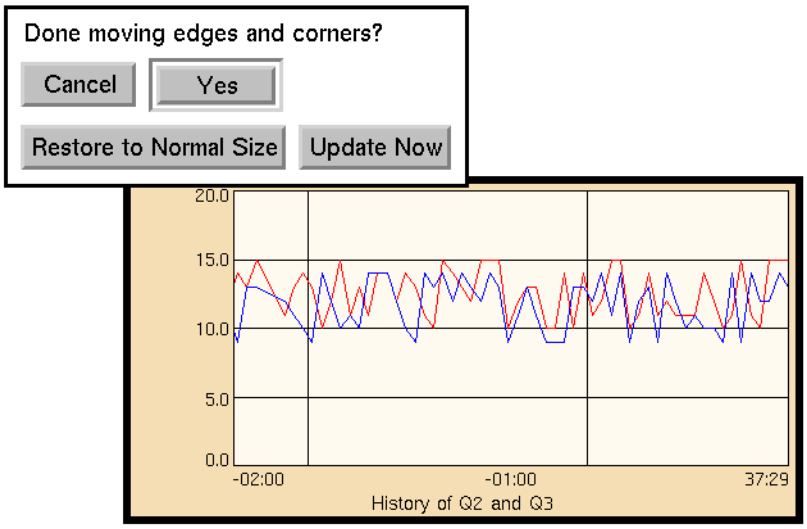
Sizing a Graph

G2 limits the size of a graph so that it cannot become so large that it consumes all available system resources.

To change the size of a graph:

➔ Choose change size from the graph's menu.

G2 draws a size box around the graph and displays the update dialog, shown next:



Move the edges and corners of the size box to the shape and size you want for the graph.

Hint When you change the size and shape of a graph, the tickmark labels and graph label remain the same size while the overall graph size changes. The minimum size of a graph is limited by its label; the graph cannot be narrower than its expression-to-display or label-to-display, whichever is shown as the label of the graph. Also, as the graph gets smaller, its labels may overlap, causing fewer labels to show.

Specifying the Data Window Time Span

The desired-range-for-horizontal-axis attribute specifies the time span that the data window of the graph represents.

The graph's horizontal axis represents time. G2 plots the history values of variables and parameters on the vertical axis (representing values) against the time that the horizontal axis represents.

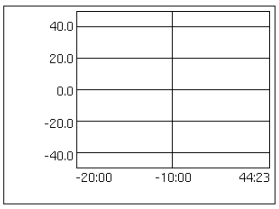
Graphs have a maximum time range of 24 weeks, in contrast with trend-charts, whose maximum time range is 10 years. Specify the value as follows:

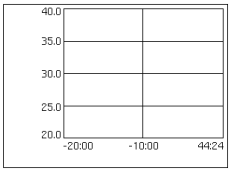
Entering a statement like this...	Causes G2 to...
none	Determine an appropriate time range for the recorded data that the graph displays.
30 seconds	Plot the data that was recorded during that last time interval. In this case, G2 plots the values that were recorded during the last 30 seconds on the graph.

Entering a statement like this...	Causes G2 to...
30 seconds to 1 minute and 30 seconds after start	Specifies a time interval range; G2 only displays those values that were recorded during that time period. Note that the history-keeping-spec must record enough datapoints to satisfy the time interval range.
without tickmark labels	Selectively suppresses the display of tickmark labels for each graph axis, when you add this statement at the end of a range specification. Suppressing tickmark labels results in a larger interior drawing area for the graph's data and smaller margins around it. This format is useful when displaying a large number of graphs so that you can fit as much data as possible on the display.

Specifying Numerical Bounds for the Value Axis

The desired-range-for-vertical-axis attribute specifies the numerical bounds for the value axis, which is the vertical axis. Complete this attribute as follows:

Entering a statement like this...	Causes G2 to...
none	Determine an appropriate range for the recorded data.
100  	Use the number you enter as the zero-centered value axis range. For example, entering 100 causes G2 to center the range of 100 around zero and show a range of values from -50 to 50, as shown at the left. Grid lines appear for -50 and 50; however, the tickmark labels may not reflect the exact values specified for the range.

Entering a statement like this...	Causes G2 to...
20 to 40 	Use that range of values that you provide for the value axis. For example, if you specify 20 to 40, G2 positions an appropriate number of axis values beginning at 20 and ending at 40 as shown to the left.
with all intervals the same	Use the range of values that you specify, but keep all grid intervals the same and make the top and bottom grid lines coincide with the top and bottom of the graph. This statement is optional.
without tickmark labels	Selectively suppress the display of tickmark labels for the graph's value axis. This statement is optional. Suppressing tickmark labels results in a larger interior drawing area for the graph's data and smaller margins around it. This format is useful when displaying a large number of graphs so that you can fit as much data as possible on the display.

Specifying Graph Scrolling

The scroll-continuously attribute specifies yes or no from the menu that appears. If no, when the data line reaches the end of the graph, the data shifts backward to the previous grid line; the grid lines remain in place. If yes, then the data and grid lines both shift backward in increments large enough to show the new data. The default is no.

Defining the Graph Percentage to Extend

The percentage-extra-space-to-leave attribute defines any positive number that indicates a percentage of the graph that G2 extends before scrolling. To increase performance, use this attribute when the scroll-continuously? attribute is set to yes so that G2 can display more data points before scrolling.

Specifying Whether Grid Lines are Visible

The show-grid-lines attribute specifies yes or no from the menu that appears. If yes, the grid lines appear on the grid background. If no, the grid lines do not appear. The default is yes.

Defining the Interval between Tickmarks

The interval-between-horizontal-grid-lines attribute defines the interval between the tickmarks on the horizontal time axis (which are the horizontal grid lines on the grid background). Complete this attribute as follows:

Entering a statement like this...	Causes G2 to...
compute automatically	Determine the interval between horizontal grid lines automatically. G2 uses an interval that lets it display between four and seven horizontal grid lines on a graph, depending on the range of the vertical axis.
4 beginning at 2	Use the interval and the origin point (beginning at *2*) that you specify for the horizontal grid lines. In this example, the tickmarks begin at 2, with intervals of 4, until they reach the limits that the desired-range-for-vertical-axis attribute specifies.

Specifying the Number and Style of Grid Lines

The extra-grid-lines attribute specifies the color and placement of extra grid lines. The default is none.

Entering a statement such as this:

```
at 10 in green
```

specifies where on the vertical axis you want to place extra grid lines, and what color they should be. The color specification is optional.

Tip You can show extra grid lines even if the show-grid-lines? attribute is set to no.

Defining a Graph's Background Color

The background-colors attribute specifies the background colors for the graph and for the grid. Enter the statements like this:

graph background: *color*, grid background: *color*

The values for *color* can be any color, but not a meta-color or a color expression.

Specifying the Expression to Display

The expression-to-display attribute specifies the variable or parameter whose history values you want to display on the graph. You can enter any variable or parameter by name, or any expression that evaluates to a variable or a parameter. Enter multiple expressions by separating them with a semicolon. Appending one or more of the following optional statements lets you format the display of the expression:

Entering a statement like this...	**Causes G2 to...**
quant-variable-1 in red	Display the data line and any plot markers for the specified expression in the color you specify.
quant-variable-1 using triangle plot marker	Display the markers for the data line as the shape you specify. Possible shapes are: plus-sign, square, or triangle.
quant-variable-1 using line width 10	Display the data line as wide as the number of workspace units you enter. Note that the line width does not affect the size of the plot markers.
quant-variable-1 using shading in blue	Shade the area below the data line with the color you specify. If you do not specify a color, G2 uses black for the shading.
quant-variable-1 with range from 0 to 5	Show the data with its range scaled to that of the graph's vertical axis. For example, if the range is from zero to five and a datapoint falls at 2, G2 displays that datapoint at 40 (for 40 percent of the range), provided that the vertical axis goes from 0 to 100.

The next example shows an expression using three variables, X, Y, and Z, each with different formatting options:

```
X in blue using square plot marker and using line width 3 with range from 1 to 50;
Y in red;
Z in yellow using shading
```

Tip If you specify a range for the expression (as in the X specification), you must also specify a range (or range width) for the desired-range-for-vertical-axis attribute.

Specifying the Graph Label

The label-to-display attribute specifies the label shown beneath the graph, which has no effect on the expression that the graph displays. If you do not enter a label-to-display, G2 displays the expression-to-display beneath the graph.

Using Grid Lines and Tickmark Labels in Graphs

When you configure a graph, the grid lines and tickmark labels may not appear exactly as you specify them. When you let G2 compute ranges automatically, it sometimes changes the layout slightly to avoid overlapping tickmark labels.

Horizontal Axis and Tickmark Labels

The horizontal axis represents time. G2 places tickmark labels at the ends of the horizontal axis, and attempts to place them next to any visible grid lines, omitting tickmark labels wherever they overlap or are too close. If the show-grid-lines? attribute is set to no, the tickmark labels remain as if the grid lines were visible.

If the desired-range-for-horizontal-axis attribute uses...	Then the...
Either a time interval or a compute automatically statement	Right-most label on the horizontal axis indicates the current time, and the other labels are relative to that label. For example, if the right-most label on the horizontal axis is 37:45 and another label reads -0:30, then the time for the latter label is 30 seconds earlier than the right-most label (37:15).
A time interval statement such as: 1 second to 20 seconds after start	Tickmark labels represent the current time.

Vertical Axis and Tickmark Labels

The vertical axis represents the values of the expression(s). G2 automatically determines where vertical tickmark labels should be, based on the size of the range specified in the desired-range-for-vertical-axis attribute. G2 creates between four and seven vertical grid lines (at the tickmark labels) on a graph. Note that if you specify that if the scroll-continuously? attribute is set to yes, the vertical grid lines move with the data and do not always align with the edges of the graph.

40

Trend Charts

An introduction to and description of trend charts and their use.

Introduction

Trend charts are graphical display items of the trend-chart class that plot time series or historical data over a designated period of time. The major features of trend chart displays are:

- Multiple Y (value) axes.
- Support of all numeric expressions, not just those that name a variable or parameter.
- An editing interface that uses subtables instead of attribute expressions.
- Fine-grained control over all aspects of trend chart labelling and layout.
- Plot marker choices, including current value and monochrome icons.
- Subsecond time support.
- Full data seeking capabilities (optional).

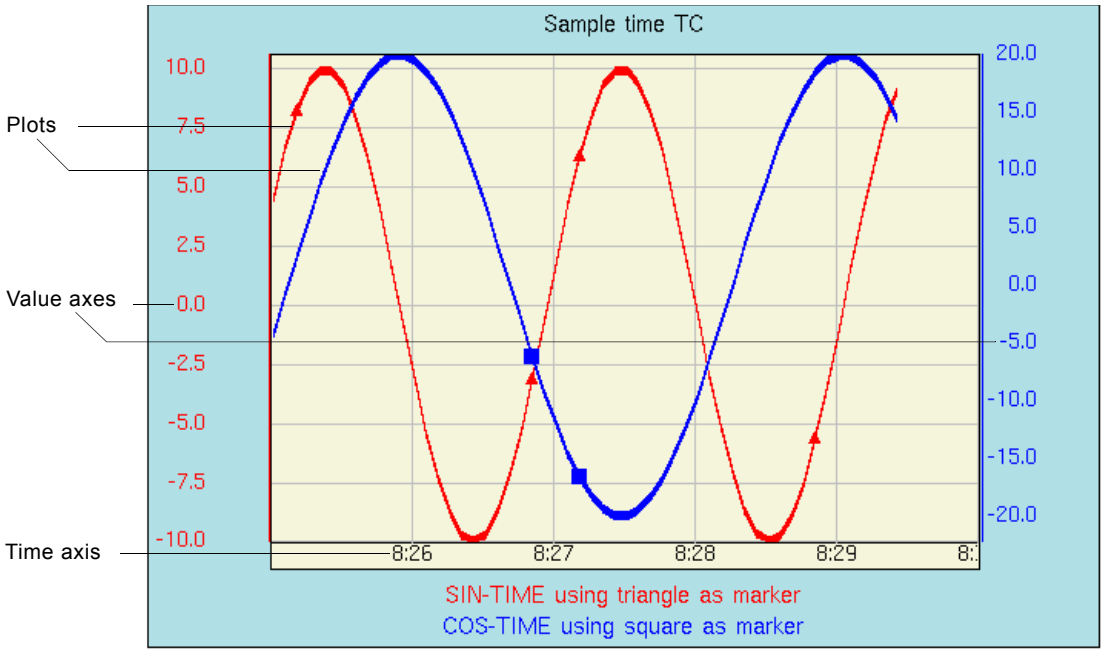
Note Do not confuse trend charts with charts, which plot numerical data in the form of one or more data series. See Charts.

About Trend Charts

A G2 trend chart consists of several **components**. Components are the building blocks of a trend chart, each customizes a particular aspect of the trend chart and its data. The trend chart components are:

- One or more plots, which plot data history values over a period of time.
- One or more value axes, which are vertical indicators of the range of values that a plot represents.
- A single time axis, indicating the time over which G2 is plotting data.
- One or more connector formats, describing whether connectors between data points appear on the chart and, if they do, how they are drawn.
- One or more point formats, describing whether markers for certain points appear on the trend chart and how and when they display.
- A single trend chart format, determining general formatting attributes for the entire trend chart.

The next figure shows a trend chart with two plots and callouts to some of the trend chart components:

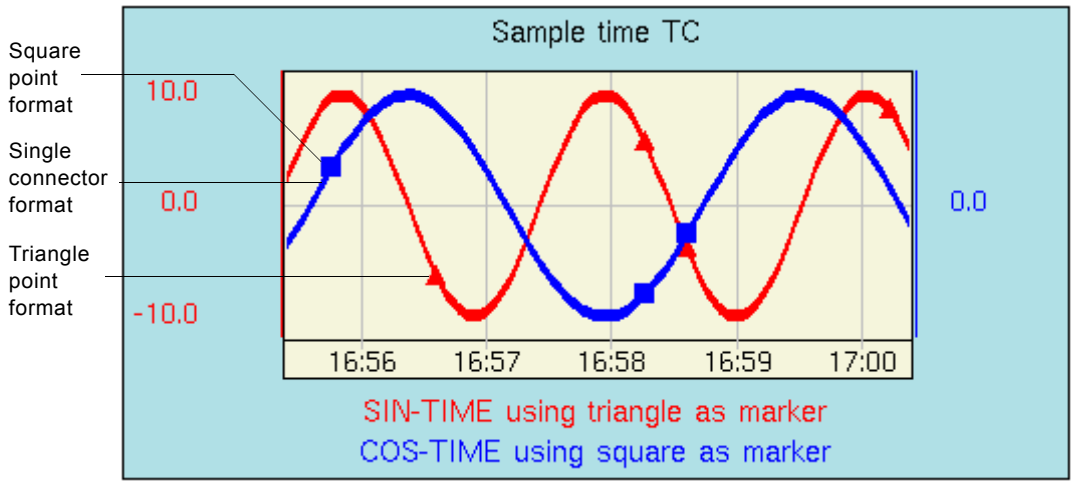


Each plot component contains an expression that plots on the trend chart. A plot expression can evaluate to a quantitative variable or a parameter, or to a quantitative value expression.

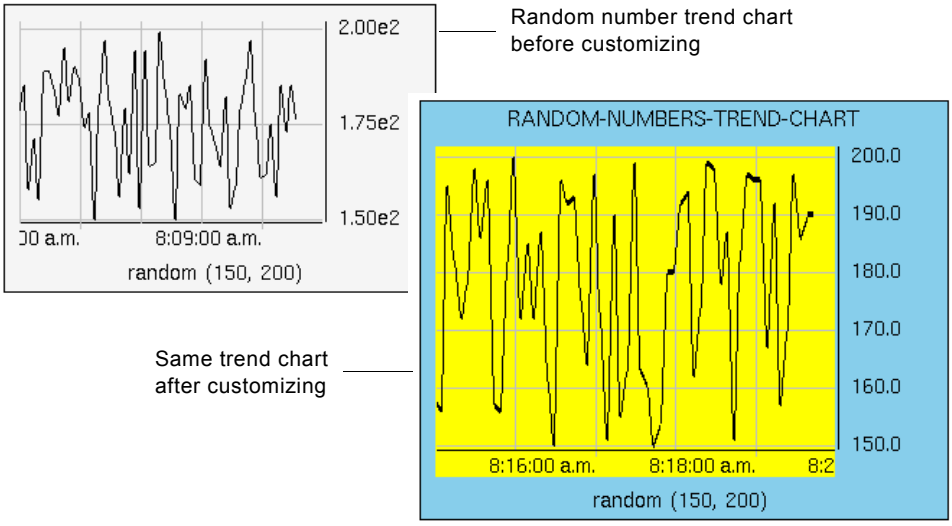
A relationship exists between each plot and its associated value axis. Every plot must have an associated value axis, and several plots may exist on a single axis. A value axis, however, does not require a plot.

Plots also have an association with point and connector formats. While plots maintain data point values, point and connector formats determine *how* the data points appear on the trend chart. Connector formats specify how the lines of a plot are drawn on the trend chart. Point formats specify what marker, if any, appears upon a drawn plot at a specified data point. Point and connector formats exist for the purpose of defining specific, named drawing styles that you can refer to in multiple plots.

The next diagram shows a single connector format, and two different point formats, triangle and square for two plots:



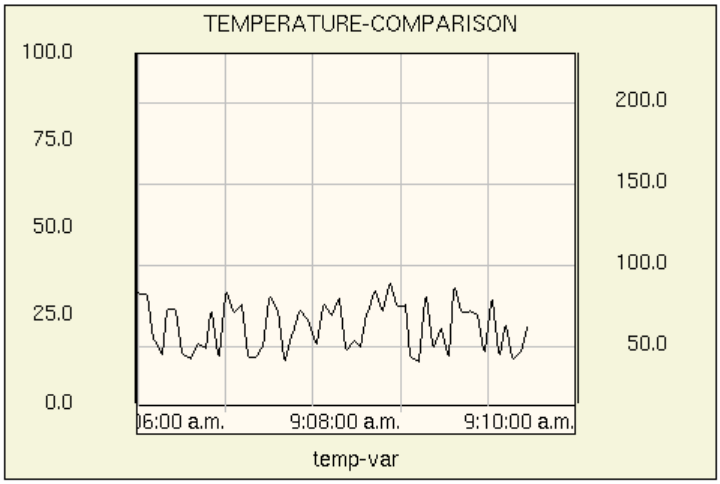
Minimally, a trend chart consists of a single time axis, one value axis, and at least one plot with an expression. Configuring a trend chart lets you fully control all aspects of the trend chart's appearance. You can configure a trend chart as much or as little as you like. As an example, the next diagram shows a single-plot trend chart plotting a random number expression before and after some minor customizing.



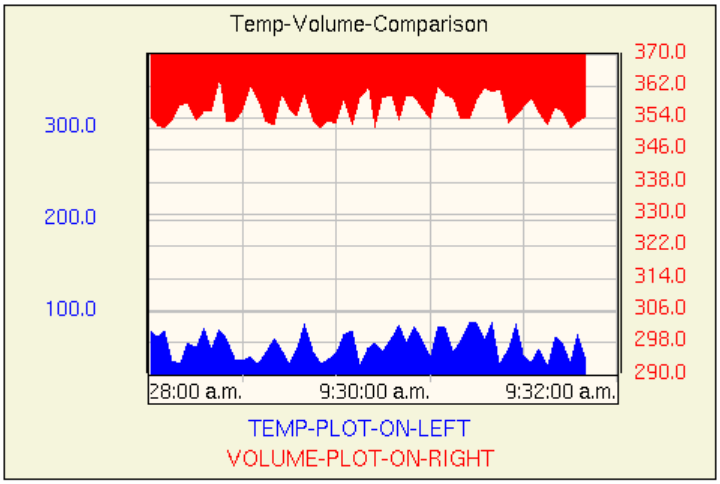
The only configuration of this trend chart was to change its size, add a title and different background colors, and change the number of significant digits on the value axis.

The ability of trend charts to support multiple value axes, sometimes called y axes, provides a flexible way to plot various types of data values. Consider the next examples.

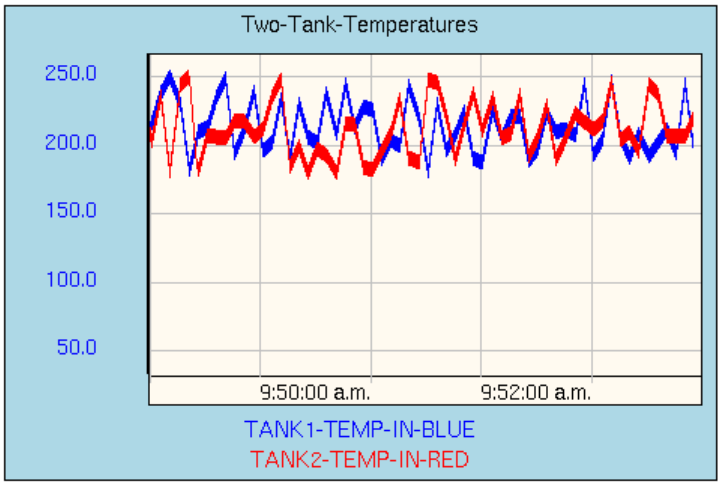
For a comparative display of one value against two measurements, such as the temperature of a tank in Fahrenheit and Celsius, use a single plot and two value axes.



To plot disparate values, such as the temperature and the volume of a tank, use two plots on two value axes.



To compare the values of two items against one another, such as the temperature of two tanks, use two plots on one value axis.



Compound Attributes

Most of the trend chart attributes are **compound attributes**. A compound attribute is composed of one or more components. The time-axis and trend-chart-format compound attributes can include only a single component. The plots, value-axes, point-formats, and connector-formats compound attributes can include multiple components.

Compound attribute components are similar to other G2 items in that they consist of attributes and corresponding values, which you can display on an attribute table. They differ from items in that you cannot see them as icons upon a workspace, or detect them as separate items through the Inspect facility or any other item reference, either interactively or programmatically.

Compound attribute values are accessible in two basic forms:

- Each component in the compound attribute has its own attribute subtable.
- The attribute/value row for the compound attribute on the trend-chart attribute table combines the values of all of the components in the compound attribute. You can select from three different views of the combination value:
  - A top-level view which simply tells you how many components are in the compound attribute.
  - A textual annotation with information on the non-default valued attributes in each component.
  - The value of the compound attribute expressed as a sequence of structures: one structure for each component.

The next four sections describe the different views and how to access them. The plot compound attribute is used in the examples.

Accessing Component Subtables

Attribute tables that represent trend chart components are called **component subtables**. A component subtable lists all the attributes of a single component and provides you with interactive access to them. Component subtables are used as examples throughout this chapter because they provide the optimal way to customize components.

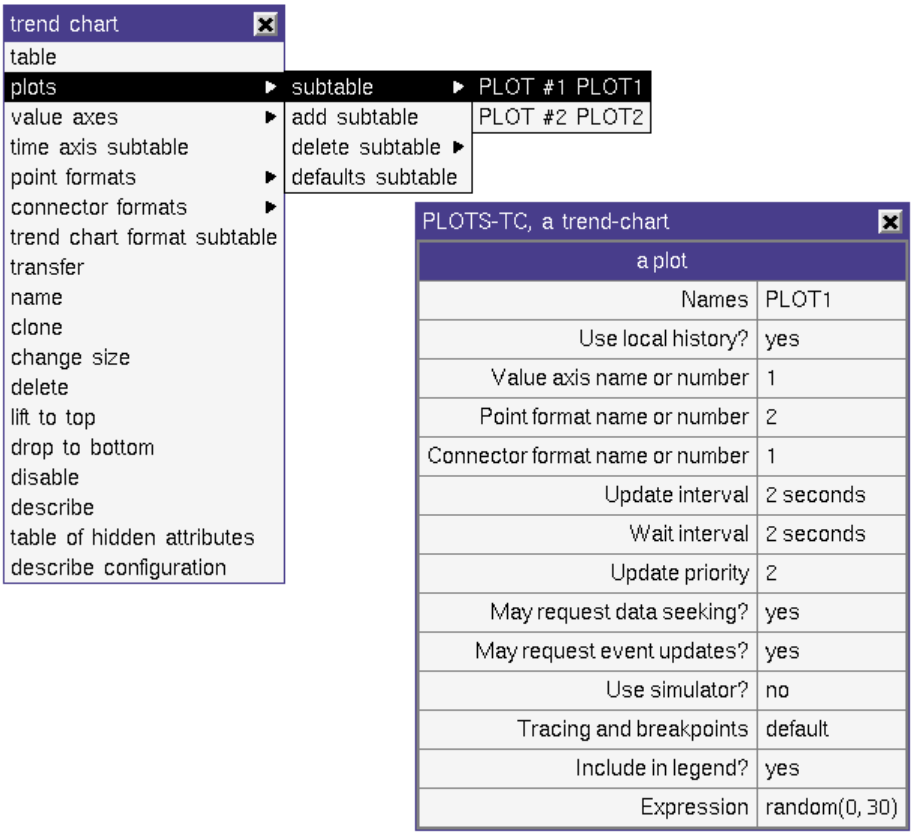
Accessing Component Subtables

For all compound attributes that can include multiple components, you can add, delete, and edit component subtables, or access the defaults subtable. The next two graphical examples show you the two ways of accessing a plot component subtable.

To access component subtables from the trend chart menu:

➔ Select *menu choice* > subtable > *component* on the trend chart.

For example: plots > subtable > *plot-component*

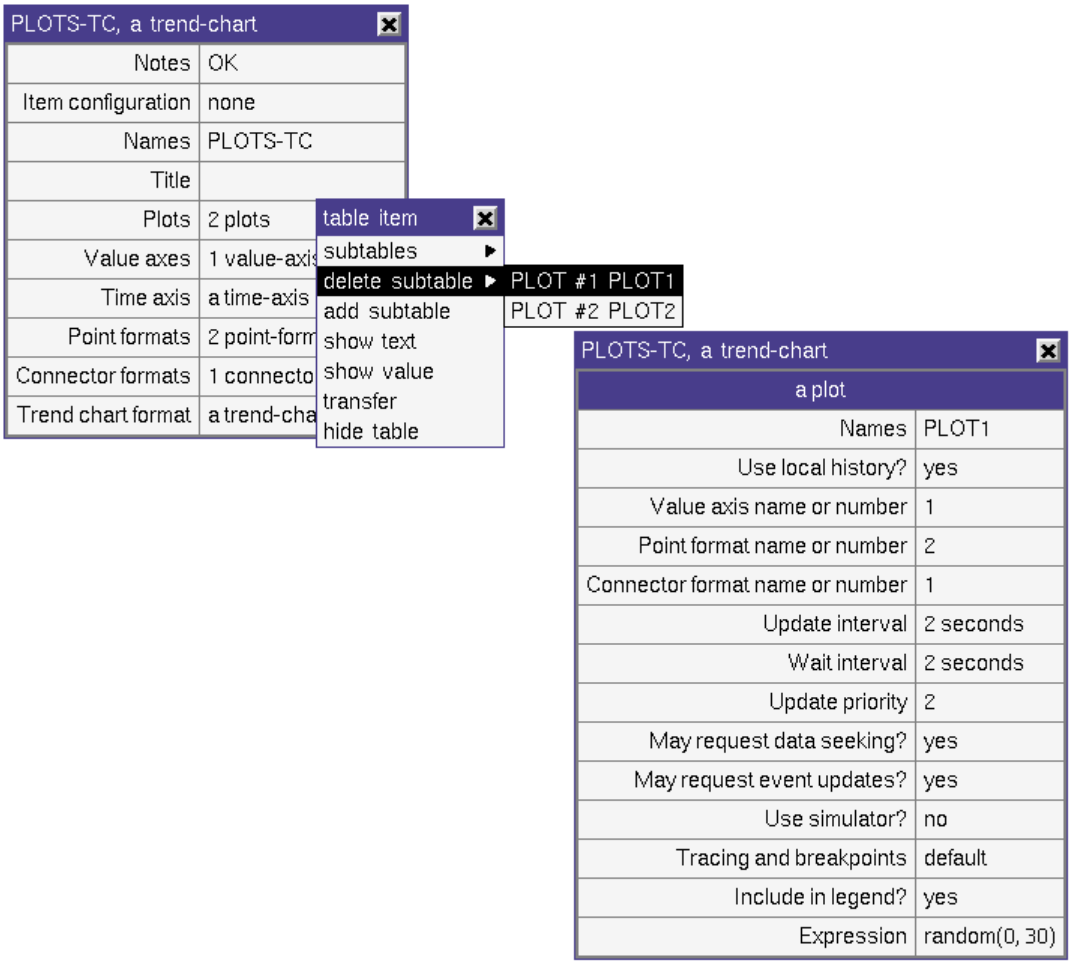


To access component subtables from the trend-chart attribute table:

➔ Select table > *compound-attribute value cell* > subtables > *component* on the trend chart.

where *compound-attribute* is the component to access, such as Plots.

For example: trend-chart > table > plots value cell > subtables > *plot-component*



Note Component subtables do not include the notes attribute. If the status of a component is anything other than ok, a message appears in the notes attribute of the trend chart attribute table. The message indicates which component is not ok.

Adding and Deleting Component Subtables Programmatically

You can add and delete component subtables programmatically by using two system procedures:

- g2-add-trend-chart-component
- g2-delete-trend-chart-component

These system procedures function almost exactly as the menu choices do. Each procedure accepts two or three arguments: the trend chart to or from which you want to add or delete something, the name of the component subtable to add or delete, and, when deleting, the specific component to delete.

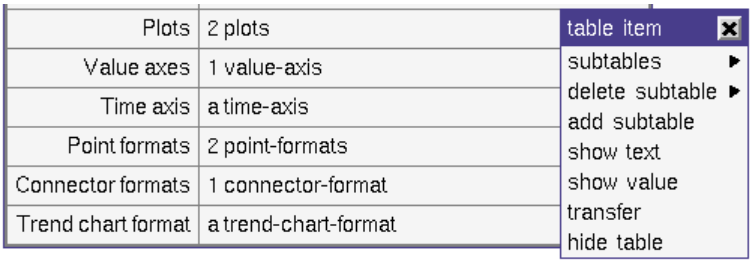
For a complete description of these and other system procedures, see the *G2 System Procedures Reference Manual*.

Selecting Compound-Attribute Value Views

The attribute/value row for a compound attribute on the trend-chart attribute table combines the values of all of the components in the compound attribute. You can select any one of three different views of a compound-attribute value.

To access a compound-attribute value view:

1 Click the value cell of the compound attribute on the trend-chart attribute table to bring up its menu.



2 Select one of these menu choices: show summary of text, show text, or show value.

The current view will not be a menu choice.

In the example above, the show summary of text menu choice is not offered because the plots value is already shown in the summary of text view. This view is the default view and simply tells you how many components the compound attribute has.

The text view is a detailed syntactical description of the non-default values of the components. For example:

Plots	2 plots: plot 1 named plot1, plot 2 named plot2; the point-format-name-or-number of the plot named PLOT1 is 2; the update-interval of the plot named PLOT1 is 2 seconds; the expression of the plot named PLOT1 is random(0, 30); the point-format-name-or-number of the plot named PLOT2 is 1; the update-interval of the plot named PLOT2 is 2 seconds; the expression of the plot named PLOT2 is random(31, 50)

The next example shows the plots attribute in value view:

Plots	sequence (structure (chart-element-uid: the symbol default, names: the symbol default, use-local-history@?: true, value-axis-name-or-number: 1, point-format-name-or-number: 1, connector-format-name-or-number: 1, update-interval: structure (update-interval: 5), wait-interval: 2, update-priority: 2, may-request-data-seeking@?: true, may-request-event-updates@?: true, use-simulator@?: false, tracing-and-breakpoints: the symbol default, include-in-legend@?: true), structure (chart-element-uid: 1, names: the symbol plot1, point-format-name-or-number: 2, update-interval: structure (update-interval: 2), expression: "random(0, 30)"), structure (chart-element-uid: 2, names: the symbol plot2, point-format-name-or-number: 1, update-interval: structure (update-interval: 2), expression: "random(31, 50)"))

The value view shows the value that is available through the attribute-access facility. In this case, the value consists of a sequence of two structures, one for each plot component.

Changing Compound Attributes

You can change a compound attribute by:

- Adding, deleting, or editing a component subtable.
- Editing the text and composite value attribute values interactively with the Text Editor.
- Using system procedures.
- Changing the compound attribute programmatically, using change the text of or conclude actions.

Regardless of how you change compound attribute, G2 reprocesses the entire attribute.

You can change a compound attribute programmatically through its composite value. This ability is available through G2's attribute access facility which is described in Attribute Access Facility.

Reprocessing the text of a compound attribute value has a number of implications, one of which is that to change even a single character of the text, you must first get all of the text. You then modify the text in its entirety to include the change.

Because of the complexity of changing a text value, we recommend that you do not use change the text of or conclude actions to do so. Instead, G2 provides two system procedures to perform this task:

- g2-get-text-of-trend-chart-component
- g2-set-text-of-trend-chart-component

The first system procedure gets the text of a textual annotation, the second lets you replace it with a change. For a complete description of these and other system procedures, see the *G2 System Procedures Reference Manual*.

Another implication of the reprocessing of textual annotations is that the entire set of components in a compound attribute is replaced by a new set of components created from the textual annotations. In general, this has little or no importance to trend chart users, with the exception of plot components that are plotting local history values. Because changing a textual annotation deletes and replaces the old set of plots, it also deletes any local history values associated with those plots. For information about using local histories, see Defining Where to Obtain History Values.

Using Component References

For compound attributes capable of multiple components, you refer to individual components by using a **component reference**. A component reference is the number or name by which you specify a particular component of a compound attribute.

By default, a component reference is a number preceded by the name of the component, such as plot #1, or connector-format #1. Each compound attribute capable of multiple components includes a single default component reference, which is 1 in all cases. You cannot delete a default component.

You add components by the methods described in Accessing Component Subtables. As you add new components to a trend chart, G2 automatically increments and assigns a component reference number for the compound attribute with each additional component. For instance, if you add two plot components to a new trend chart, in addition to the default component, their component reference numbers will be 2 and 3, respectively.

Component reference numbers are reusable. For instance, if you have three plots, 1, 2, and 3, and you delete plot #2, plots 1 and 2 remain, not 1 and 3, as you might expect.

Each component subtable includes a names attribute, through which you can name the component. Naming a component subtable does not negate its numeric reference. You can use a component name interchangeably with its corresponding reference number when referencing a component. By using a component reference name, you alleviate the need to refer to component reference numbers, which change as you add and delete components.

Note Because trend chart component reference numbers are positional, we recommend that you always provide a unique name to each trend chart component. By naming components, and then by using the appropriate name as a reference, you avoid inadvertently referencing the wrong component.

Setting Component Defaults

All compound attributes capable of multiple components also have a defaults subtable. In contrast to a component subtable that lets you format a *specific* component, a defaults subtable lets you define general format settings that can apply to all of the corresponding components.

For instance, two of the attributes on the plot defaults subtable are update-interval and wait-interval. Changing the value of these two attributes to 3 seconds changes the values on all plots that have not overridden the default value. Specific component subtables can always override any default settings for the component attributes.

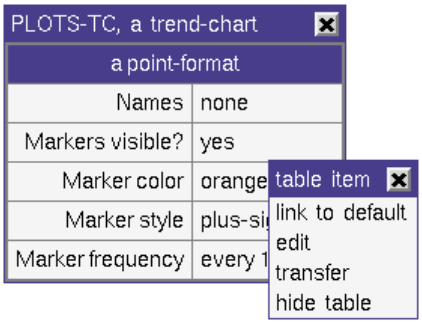
The defaults subtables themselves have a set of default attribute values. Each new component consists of these defaults before customizing. Changing the values on a defaults subtable appears in the text view of the attribute value as an any statement, such as the any point-format statement shown next in the following sample point-formats attribute:

Point formats	1 point-format; the marker-color of any point-format is dark-slate-blue

The defaults subtables of all compound attributes are accessible in the same way as other choices, described in Accessing Component Subtables.

Linking to a Default Value

When you enter a non-default value for any attribute on a component subtable that also appears on the component defaults subtable, you can revert to the default value at any time interactively by choosing the link to default choice from the table menu, as shown here.



After you change the default attribute value, the link to default menu choice is available to revert to the default value.

The defaults subtables of compound attributes include most of the attributes that appear on the corresponding components. Each of the following sections describe how to customize the trend chart components (plot, value-axis, etc.). They also describe the corresponding defaults subtables, and note what attributes do not appear on the defaults subtable.

Configuring Trend Charts

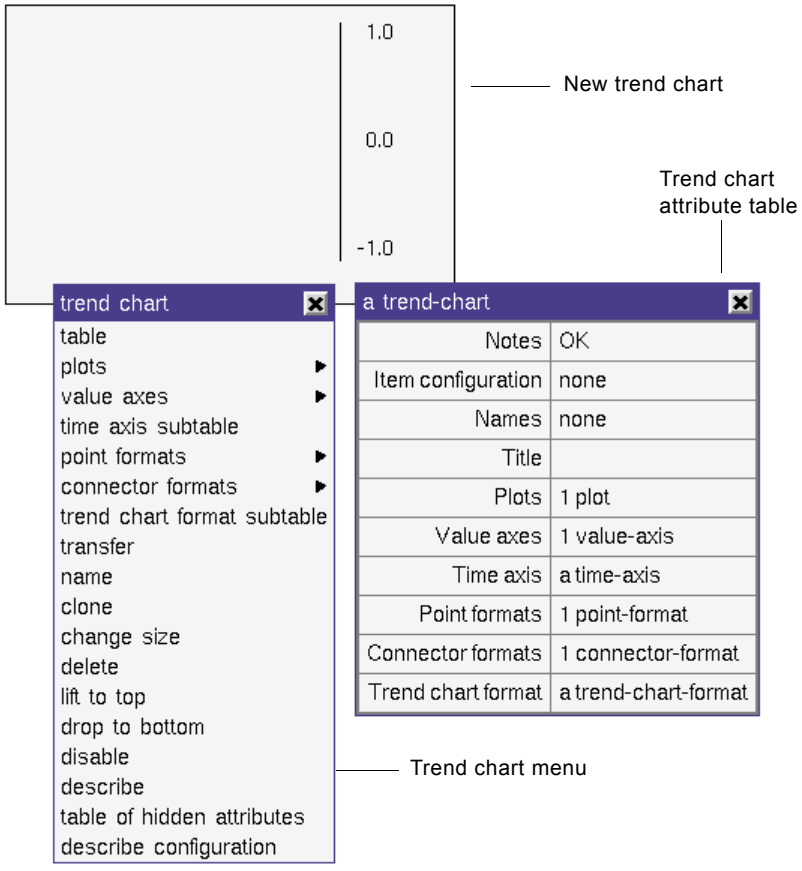
To configure a trend chart initially, you need only to provide an expression to plot.

Creating a Trend Chart

To create a trend chart:

➔ Select KB Workspace > New Display > trend-chart.

A trend chart appears upon the workspace. Click on the trend chart to display its menu. Choose table to display the attribute table of the trend chart. The next diagram shows a new trend chart, the trend chart menu, and the trend chart attribute table.



By default, a new trend chart contains the minimal set of useful components — a single value axis, ranging from -1.0 to 1.0 and a time axis. One plot is defined, but needs an expression before the trend chart can plot any data. If you start your KB

with a new trend chart you have not customized, the chart will scroll as time progresses, but no data will be displayed because the plot has no expression.

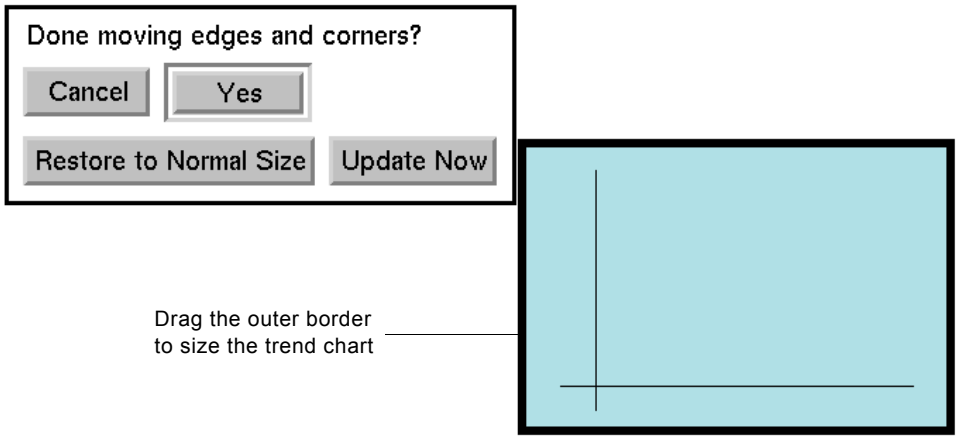
Sizing a Trend Chart

G2 limits the size of a trend chart so that it cannot become so large that it consumes all available system resources.

To change the size of a trend chart:

➔ Choose change size from the trend chart's menu.

G2 draws a size box around the trend chart and displays update dialog, shown here. Move the edges of the size box so that the trend chart is the size you want and click Update Now in the dialog.



Summarizing Trend Chart Attributes

Configuring a trend chart requires editing one or more of the compound attributes. The trend chart attributes are:

Attribute	Description
title	An optional label that is displayed on the top of the trend chart.
Allowable values:	Any symbol
Default value:	blank

Attribute	Description
plots	In summary of text view, displays the number of plots that the trend chart is plotting. In text view, it provides a description of all non-default plot attributes, including the value axis name or number, point and connector format name or number, and the expression of each plot. In value view, it gives the composite value of the plots.
Allowable values:	See Configuring Plots.
Default value:	1 plot
value-axes	In summary of text view, displays the number of value axes that the trend chart uses. In text view, provides an annotational description of each value axis and any non-default values. In value view, it gives the composite value of the value axes.
Allowable values:	See Configuring Value Axes.
Default value:	1 value-axis
time-axes	In summary of text view, displays the time axis indicator. In text view, supplies an annotational description of each of the time axis attributes and any non-default values. In value view, it gives the composite value of the time axes.
Allowable values:	See Configuring the Time Axis.
Default value:	a time-axis

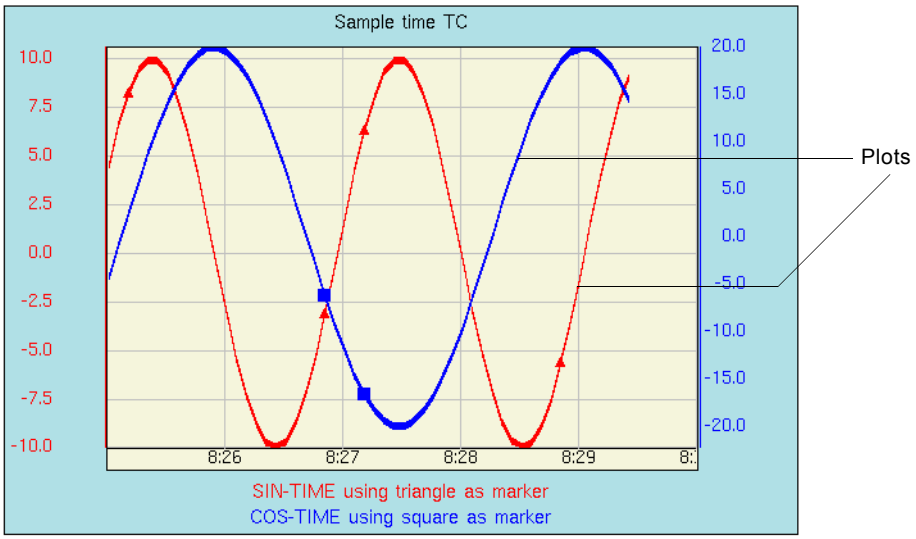
Attribute	Description
point-formats	In summary of text view, displays the total number of point formats that the trend chart uses. In text view, supplies an annotational description of each point format and any non-default values. In value view, it gives the composite value of the point formats.
Allowable values:	See Configuring Point Formats.
Default value:	1 point-format
connector-formats	In summary of text view, displays the total number of connector formats that the trend chart uses. In text view, supplies an annotational description of each connector format and any non-default values. In value view, it gives the composite value of the connector formats.
Allowable values:	See Configuring Connector Formats.
Default value:	1 connector-format
trend-chart-format	In summary of text view, displays the trend chart format indicator. In text view, supplies an annotational description of each of the trend chart format attributes containing non-default values. In value view, it gives the composite value of the trend-chart format.
Allowable values:	See Configuring the Trend Chart Format.
Default value:	a trend-chart-format

The following sections describe how to customize each trend chart component.

Configuring Plots

The plots attribute lets you define the single or multiple data plots that appear on the trend chart. A trend chart plots histories of values. Each plot specifies where the values are stored. The history values may be stored in the history of a variable or a parameter, or locally in the trend chart.

You create a plot for every data series you want to display. Every plot specifies a value-axis, a point-format, and a connector-format. Plots can be similar in appearance and color or, by using different point- and connector-formats, each can have a unique look. The next diagram indicates two plots on a trend chart:



Access plots from:

- The plots choice from the trend chart menu.
- The plots choice from the trend chart attribute table.

The attribute tables of plot defaults and plots are:

Plot defaults table

PLOTS-TC, a trend-chart	
a plot	
Names	DEFAULT
Use local history?	yes
Value axis name or number	1
Point format name or number	1
Connector format name or number	1
Update interval	5 seconds
Wait interval	2 seconds
Update priority	2
May request data seeking?	yes
May request event updates?	yes
Use simulator?	no
Tracing and breakpoints	default
Include in legend?	yes
Expression	

Plot table

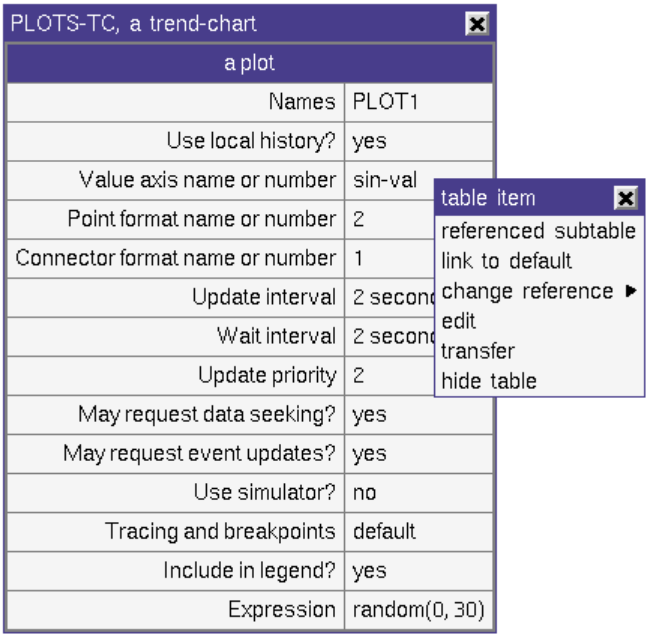
PLOTS-TC, a trend-chart	
a plot	
Names	PLOT1
Use local history?	yes
Value axis name or number	1
Point format name or number	2
Connector format name or number	1
Update interval	2 seconds
Wait interval	2 seconds
Update priority	2
May request data seeking?	yes
May request event updates?	yes
Use simulator?	no
Tracing and breakpoints	default
Include in legend?	yes
Expression	random(0, 30)

Plot subtables include three attributes that refer to other components, namely the value-axis, point-format, and connector-format components.

By accessing the table choice (clicking at the side of the value) of each of these plot subtable attributes, you can:

- Access the referenced component subtable directly.
- Link to the default subtable for that component (only if you have changed a default value).
- Change the current component reference by choosing another component reference.

The following diagram shows the value-axis-name-or-number attribute table choice:



Defining Where to Obtain History Values

The use-local-history? attribute defines whether the plot is plotting local history values, stored within the trend chart, or external history values, stored within a variable or a parameter.

Setting this attribute to yes (the default) directs G2 to maintain a history of values locally. G2 evaluates local history values from the plot expression in the plot's expression attribute, regardless of whether that expression refers to a variable or parameter that itself maintains a history of values.

Setting this attribute to no plots the history values of a variable or a parameter and implies that the plot expression *must* evaluate to a variable or a parameter, specifically, one that maintains history. If you set the use-local-history? attribute to no and the variable or parameter is not saving history values, the trend chart does not display anything for this plot.

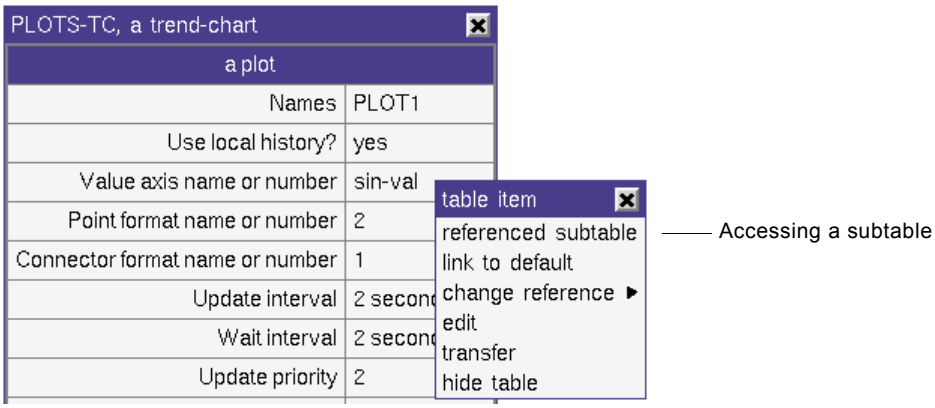
For information about history keeping specifications, see Specifying Whether to Keep a History of Values.

Specifying the Value Axis for the Plot

The value-axis-name-or-number attribute indicates the component reference name or number of the value axis for the current plot. The default number 1 indicates that the plot will use value axis #1, which is the default.

If you create other value axes and provide names for them, or provide a name to value axis #1, you can enter either the name or the number of the value axis.

You can access the value axis subtable that this plot references by choosing referenced subtable from the attribute's table choice, or change the current reference to a new one by choosing change reference, shown next.



Specifying the Point Format

The point-format-name-or-number attribute specifies the component reference name or number of the point format for the current plot. The default number 1 indicates that the plot will use point-format #1, which is the default.

If you create other point formats and provide names for them, or provide a name to point-format #1, you can enter either the name or the number of the point format component reference.

You can access the point format subtable that this plot references by choosing referenced subtable, or change the current reference to a new one by choosing change reference.

Specifying the Connector Format

The connector-format-name-or-number attribute specifies the component reference name or number of the connector format this plot will use. The default number 1 indicates that the plot will use connector-format #1, which is the default.

If you create other connector-formats and provide names for them, or provide a name to connector-format #1, you can enter either the name or the number of the connector format component reference.

You can access the connector format subtable that this plot references by choosing referenced subtable, or change the current reference to a new one by choosing change reference.

Defining the Update Interval

The update-interval attribute determines the interval at which G2 evaluates the expression contained in the expression attribute. This interval takes effect *following* the first time G2 evaluates the expression after activation. Conversely, the wait-interval attribute determines the interval G2 waits before evaluating the plot expression for the first time after activation.

The update-interval attribute determines when the expression is evaluated regardless of the value of the use-local-history? attribute. To recap what the section Defining Where to Obtain History Values describes, that attribute determines whether a plot is using historical values maintained in the trend chart, or the history values of a variable or a parameter. If use-local-history? is set to no, indicating that a variable or parameter history is being plotted, the expression is still evaluated at the interval that this attribute specifies.

When use-local-history? is set to yes, and to correspond with G2's subsecond interval capabilities, the update-interval attribute can optionally include the minimum interval between data points statement, as:

```
2 seconds, with minimum interval between data points = .5 seconds
```

The minimum interval between data points statement lets you specify the granularity between data points, that is, the amount of time between when G2 saves one data point and the next.

This statement is identical to one you could specify for the history keeping specification of variables and parameters, and is described in Specifying a Minimum Interval between History Data Points.

Specifying the Activation Interval

The wait-interval attribute determines the interval G2 waits before evaluating the expression in the plot's expression attribute after activation.

Specifying the Update Priority Level

The update-priority attribute sets the priority level within the scheduler task queue for the trend chart expression. Priority levels range from 1 – 10, with 1 being the highest priority. The default for this attribute is 2. The number you enter affects how soon G2 evaluates a given expression when its task reaches the top of the task queue.

Specifying Data Seeking Capabilities

The may-request-data-seeking? attribute specifies whether the trend chart can data seek for an unknown value. When the use-local-history? attribute is yes, and the plot expression includes a variable or a parameter such as:

```
95 + X
```

where X is a variable, if the value of X has expired and this attribute is set to yes, the trend chart causes G2 to data seek for a value for X.

Using Simulated History Values

The use-simulator? attribute should always be set to no, because the G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices.

Specifying Event Updates

The may-request-event-updates? attribute determines whether the trend chart receives event updates. Forward chaining is one example of an event update. For instance, if a trend chart expression relies on a value that changes, setting this attribute to yes indicates that when the value changes, the expression receives the new value.

If this attribute is set to no, changing values may not forward chain to the trend chart.

Defining the Debugging Level

The tracing-and-breakpoints attribute lets you set tracing and breakpoint levels. Setting this attribute overrides the tracing-message-level and breakpoint-level attributes in the Debugging Parameters system table.

Note The tracing-and-breakpoints-enabled? attribute of the Debugging Parameters system table must be set to yes for tracing and breakpoints to occur.

Entering an Expression

The expression attribute specifies what value this plot represents. This attribute is not on the plot defaults subtable.

This attribute works on conjunction with the use-local-history? attribute, described in Defining Where to Obtain History Values, as follows:

If use-local-history? is yes then...	If use-local-history? is no then...
The expression must evaluate to a quantity data type.	The expression must evaluate to a variable or a parameter of the quantity, float, or integer data type. Further, for the trend chart to plot data, the variable or parameter must be keeping history.

Note The value of the plot expression cannot exceed ± 1.7978e304, and will be truncated to this limit.

Summarizing Plot Attributes

The plot component attributes are:

Attribute	Description
names	The optional name of the plot. We recommend that you name each trend chart plot. This attribute is not on the plot defaults subtable.
Allowable values:	Any symbol
Default value:	none
use-local-history?	Indicates whether to use a history of values maintained within a trend chart from the data points it plots (yes) or to plot the history values of a variable or a parameter (no).
Allowable values:	{yes | no}
Default value:	yes

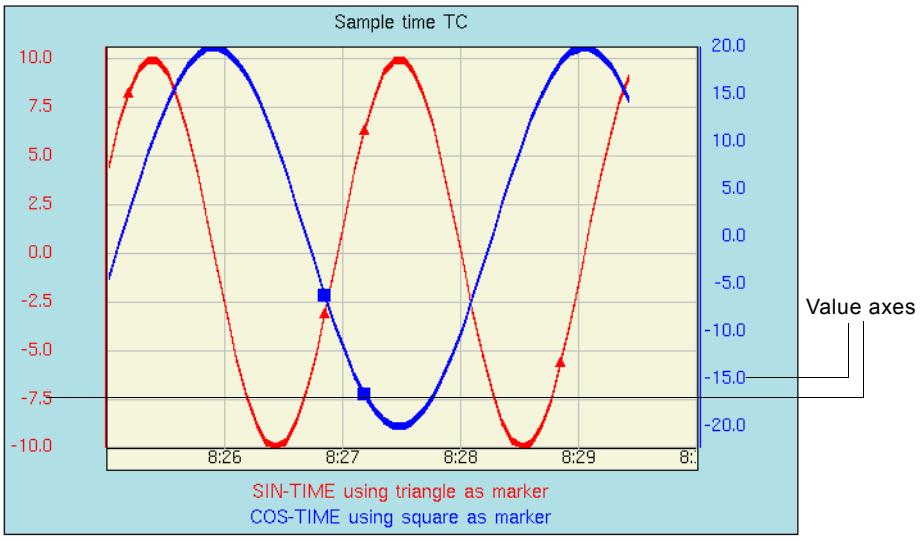
Attribute	Description
value-axis-name-or-number	The component reference name or number of the value axis for this plot.
Allowable values:	See Configuring Value Axes.
Default value:	1
point-format-name-or-number	The component reference name or number of the plot's point format.
Allowable values:	See Configuring Point Formats.
Default value:	1
connector-format-name-or-number	The component reference name or number of the plot's connector format.
Allowable values:	See Configuring Connector Formats.
Default value:	1
update-interval	The interval at which G2 evaluates the expression in the expression attribute.
Allowable values:	{*time-interval* [minimum interval between data points] }
Default value:	5 seconds
wait-interval	The interval G2 waits before evaluating the expression in the expression attribute after the trend chart is activated.
Allowable values:	*time-interval*
Default value:	2 seconds

Attribute	Description
update-priority	The task priority level at which G2 evaluates the plot expression.
Allowable values:	{1 – 8}
Default value:	2
may-request-data-seeking?	Determines whether the trend chart can data seek for an unknown value.
Allowable values:	{yes | no}
Default value:	yes
may-request-event-updates?	Determines whether the trend chart receives event updates.
Allowable values:	{yes | no}
Default value:	yes
use-simulator?	Determines whether the plot uses simulated history values of a variable.
Allowable values:	{yes | no}
Default value:	no
Notes:	The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices.
tracing-and-breakpoints	The level of tracing and breakpoints.
Allowable values:	{default | warning message level (0, 1, 2, or 3) | tracing message level (0, 1, 2, or 3) | breakpoint level (0, 1, 2, or 3)}
Default value:	default

Attribute	Description
include-in-legend?	Specifies whether information about this plot is included in the trend chart legend.
Allowable values:	{yes | no}
Default value:	yes
expression	The expression whose value this plot represents.
Allowable values:	*value-expression*
Default value:	No default value

Configuring Value Axes

The value-axes attribute of a trend chart determines how the vertical axes of a chart appear and behave. The value axes of a trend chart are:



You can access value axes from:

- The value axes choice from the trend chart item menu.
- The subtables choice of the value-axes attribute on the trend chart attribute table.
- The table of the value-axis-name-or-number attribute on a plot subtable.

The attribute tables of the value axis defaults and the value axis subtable are:

Value axis defaults table

VALUE-AXES-TC-1, a trend-chart	
a value-axis	
Names	DEFAULT
Value axis visible?	yes
Value axis position	right
Range mode	autoranging on data window
Range bounds	none
Range slack percentage	10
Labels visible?	yes
Label color	black
Label frequency	automatic
Show labels as percent?	no
Significant digits for labels	4
Grid lines visible?	yes
Grid line color	gray
Grid lines per label	1
Extra grid lines	none
Baseline visible?	yes
Baseline color	black

Value axis table

VALUE-AXES-TC-1, a trend-chart	
a value-axis	
Names	SIN-VAL
Value axis visible?	yes
Value axis position	right
Range mode	autoranging on data window
Range bounds	none
Range slack percentage	10
Labels visible?	yes
Label color	red
Label frequency	automatic
Show labels as percent?	no
Significant digits for labels	4
Grid lines visible?	yes
Grid line color	gray
Grid lines per label	1
Extra grid lines	none
Baseline visible?	yes
Baseline color	black

Displaying the Value Axis

The value-axis-visible? attribute determines whether the value axis is displayed on the trend chart. This attribute is not on the value axis defaults subtable.

Regardless of whether a value axis is visible, all of its functionality, such as its range-mode and range-bounds, is still in effect.

Specifying the Value Range

The range-mode attribute specifies how G2 adjusts the value axis in response to plotting values outside of the current range of this value axis. The range of the value axis increases by the amount specified in the range-slack-percentage attribute.

This attribute has three modes:

- fixed
- autoranging on data window (the default)
- autoranging on all past values

The term autoranging indicates that G2 controls the range of values for this value axis according to the criteria you specify as:

When range-mode is...	Then G2...
autoranging on data window	Adjusts the value axis bounds to the minimum and maximum values of all data plotted in the data window.
autoranging on all past values	Adjusts the value axis bounds to include the minimum and maximum values, not only of the data plotted on the data window, but of all values in the trend chart history since G2 started.
fixed	Includes only those values specified in the next attribute, range-bounds. G2 does not plot points outside of this range. For example, if you want the range to be from 0 – 100, set the range-mode attribute to fixed, and specify the range-bounds attribute as 0 to 100.

Specifying fixed in this attribute and *not* setting the range-bounds attribute accordingly causes G2 to use the attribute default, autoranging on data window.

All values that the trend chart displays will be truncated to lie within the range of ± 1.7978e304, even when the value axis is autoranging.

Specifying Range Limits

The range-bounds attribute specifies the numerical bounds for a value axis as a from-to value, for instance 0 to 100. These bounds are used in conjunction with the range-mode attribute as follows.

If range-mode is fixed, the trend chart uses the range-bounds exactly as you specify them.

If range-mode is set to either of the autoranging modes, G2 uses the range-bounds that you specify as a starting point for the value axis bounds. If the minimum or maximum plot data values exceed the specified range-bounds, the bounds are adjusted automatically to accommodate the values.

Note The value bounds cannot exceed ± 1.7978e304, and will be truncated to this limit.

Defining the Range Slack Percentage

The range-slack-percentage attribute defines the percentage G2 increases the range-mode value of the value axis in response to plotting a point that is not currently in range. This attribute is applicable only when the range-mode attribute is set to either of the autoranging modes, described in Specifying the Value Range.

Specifying the Label Frequency

The label-frequency attribute specifies the number of labels that are displayed on the value axis and where those labels will appear.

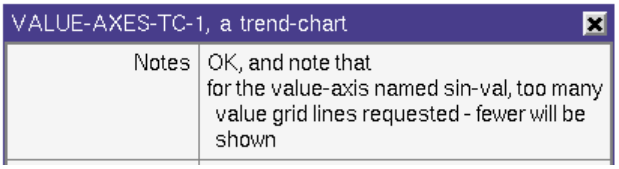
Choosing automatic directs G2 to provide an appropriate number of value labels based on the space available on the trend chart. G2 automatically computes which values to label, based on the plot values.

You can also specify a set number of value labels (such as 10 labels), or a label at specific intervals (every 10), optionally starting at a particular number (every 10 beginning at 20). If no beginning is specified, 0 is assumed.

When the grid-lines-visible? attribute for the value axis is set to yes, each label is aligned on the grid line. If the value range is very large, it is possible for there to be too many grid lines to accommodate labels.

For instance, if you specify the label-frequency as every 10 and the range-bounds are from 0 to 10,000, the range exceeds the number of labels that can be displayed legibly.

In such a case, G2 would change the label frequency that you had requested and display an appropriate message in the trend chart notes attribute, as shown in this diagram:



Displaying Labels as Percentages

The show-labels-as-percent? attribute changes the value axis labels to show the values as a percentage of the entire range of the value axis range.

Before a plot expression is specified, and while the significant-digits-for-labels attribute is at its default value of 4, the default value axis labels are 4.5%, 50.0%, and 95.5%.

Specifying the Significant Digits for Labels

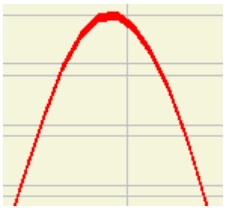
The significant-digits-for-labels attribute specifies the number of significant digits. The default 4 accommodates numbers up to 99.99 and numbers down to, and including, 0.01.

If the label value is outside of the range that can be accommodated, G2 displays the label as an exponential expression, 1.0e2 for 100 when 4 is the value for this attribute.

Showing Grid Lines

The grid-lines-visible? attribute determines whether grid lines are displayed for the value axis.

In most cases, for the sake of clarity, we recommend that only one value axis has a set of visible gridlines.



For instance, in a trend chart that has two value axes, each representing different plot values, especially when the range-mode is set to autoranging on data window, two grid lines can make the trend chart difficult to read. The diagram here illustrates a portion of the sample trend chart used throughout this chapter with grid-lines-visible? set to yes for two value axes.

Adding Extra Grid Lines

The extra-grid-lines attribute specifies:

- That extra grid lines should be displayed on the trend chart.
- Where on the vertical axis to place the extra lines.
- The color of the extra lines.

For example, to add extra grid lines at several different positions in a particular color specify:

```
at 100, at 200, at 300 in blue
```

If you do not specify a color for the extra grid lines, G2 uses the current grid line color. Note that you can see extra grid lines even if the grid-lines-visible? attribute is set to no.

Displaying a Baseline

The baseline-visible? attribute determines whether the baseline for the value axis is displayed. Set the attribute to yes to display the baseline, which is a vertical line that displays to the left or the right of the value axis, depending on which side of the trend chart the axis is displaying.

Specifying the Baseline Color

The baseline-color attribute determines the color of the value axis baseline. The value axis baseline must be set to yes for this attribute to have any effect.

Summarizing Value Axis Attributes

The value axis attributes are:

Attribute	Description
names	The optional value axis name. This attribute is not on the defaults subtable. You can refer to a value axis either by name (if you provide one) or by reference number in the trend chart attribute table or on the plots table.
Allowable values:	Any symbol
Default value:	none
value-axis-visible?	Determines whether the value axis is displayed on the trend chart. This attribute is not on the value axes defaults subtable. Regardless of whether a value axis is visible, the value axis functionality, such as its range-mode and range-bounds, are still in effect.
Allowable values:	{yes | no}
Default value:	yes

Attribute	Description
value-axis-position	Determines on which side of the trend chart the value axis is displayed. Changing the side moves both the value axis base line and any labels.
Allowable values:	{left | right}
Default value:	right
range-mode	Determines how G2 adjusts the value axis in response to plotting values outside of the current range of this value axis.
Allowable values:	{fixed | autoranging on all past values | autoranging on data window}
Default value:	autoranging on data window
range-bounds	Specifies the numerical bounds for a value axis.
Allowable values:	{none | from *integer* to *integer*}
Default value:	none
range-slack-percentage	The amount G2 increases the range mode value of the value axis in response to plotting a point that is not currently in range.
Allowable values:	{none | 1 – 100}
Default value:	10
labels-visible?	Determines whether the value labels are visible.
Allowable values:	{yes | no}
Default value:	yes

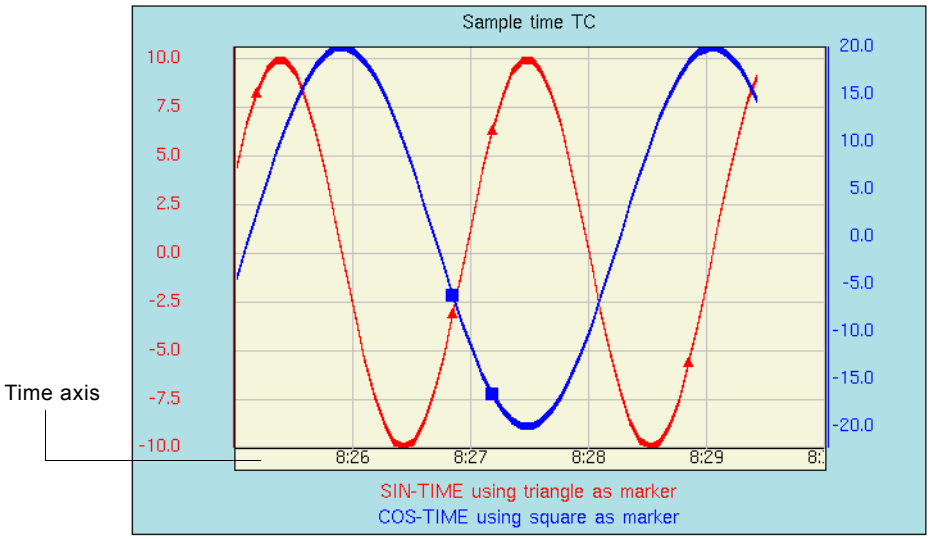
Attribute	Description
label-color	Determines the color of the labels for this value axis.
Allowable values:	{*color-name* | match plot color}
Default value:	black
label-frequency	Determines the number of labels that are displayed and where those labels appear on the value axis.
Allowable values:	{automatic | *integer* labels {every *integer* | every *integer* beginning at *integer*} }
Default value:	automatic
show-labels-as-percent?	Changes the labels to show values as a percentage of the value axis range.
Allowable values:	{yes | no}
Default value:	no
significant-digits-for-labels	Determines the number of significant digits. The default 4 accommodates numbers up to 99.99 and numbers down to, and including, 0.01.
Allowable values:	*integer*
Default value:	4
grid-lines-visible?	Determines whether grid lines for the value axis are visible.
Allowable values:	{yes | no}
Default value:	yes

Attribute	Description
grid-line-color	Determines the color of the grid lines for this value axis.
Allowable values:	*color-name*
Default value:	gray
grid-lines-per-label	Lets you determine how many grid lines appear for each value axis label you are formatting.
Allowable values:	{1 – 50}
Default value:	1
extra-grid-lines	Determines the position and color of extra grid lines.
Allowable values:	{none | at *number* [, ...] [in *color-name*] }
Default value:	none
baseline-visible?	Determines whether the baseline for the value axis is visible.
Allowable values:	{yes | no}
Default value:	yes
baseline-color	Determines the color of the value axis base line.
Allowable values:	*color-name*
Default value:	black

Configuring the Time Axis

Trend charts plot data values over a period of time. The time axis component determines the period of time over which values are plotted, how that time period moves and grows, and how the time period is labelled. There is one time axis per trend chart, which you must define.

By customizing the time axis, you can determine how scrolling occurs—the way in which plots are displayed on the trend chart, along with what data the trend chart displays. The time axis of a trend chart is at the bottom of the data window:



Access the time axis component from:

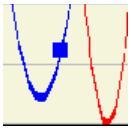
- The time axis subtable choice from the trend chart item menu.
- The time-axis attribute of the trend chart attribute table.

Defining the Data Window Time Span

The data-window-time-span attribute determines the time span that the data window represents. You can base the time span on a specific time interval (fixed), or on the history of values.

Entering a specific time interval means that the time span of the data window is fixed, and moves according to the values of the end-time, jump-scroll?, and jump-scroll-interval attributes.

Depending on what plots are active, changing this value can affect the way in which plots appear on the trend chart. As an example, the figure above captures a sample portion of the sine curve from the trend chart shown throughout this chapter when the data-window-time-span has a value of 5 minutes.

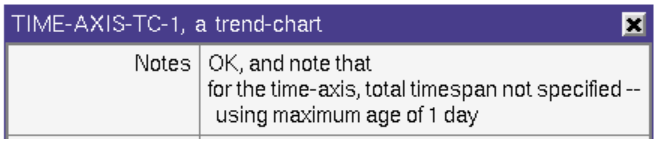


In contrast, this figure captures a similar portion of the trend chart after changing the data-window-time-span to 10 minutes. When the time span is longer, G2 has to display more data.

In this example, since more of the plot is visible within the data window, G2 adjusts the trend chart display to accommodate as much data as possible, having the effect of compressing the sine curve.

Entering show all history means that G2 ignores the end-time, jump-scroll? and jump-scroll-interval attributes. The time of the trend chart then reflects the time span of all the values in the histories of all of its plots.

If the data-window-time-span attribute is set to show all history and the total-time-span attribute is set to same-as-data-window, G2 uses a maximum time span of one day and includes that in the notes attribute as:



Specifying How Long to Maintain Local History

The total-time-span attribute specifies how long history is maintained in a trend chart. This attribute relates to and is used *only* for plots whose use-local-history? attribute is set to yes. It is useful when you need to work with the end-time attribute to view data displayed earlier.

If use-local-history? is set to no for a plot, the total time span for that plot is limited by the history keeping specification of the variable or parameter being plotted.

Always set this attribute to a value greater than that of the data-window-time-span attribute. If you set the value to be *less* than the data window time span, G2 adjusts the value to be at least equal to the data-window-time-span value.

Setting the total-time-span attribute to same-as-data-window when the data-window-time-span attribute has a show-all-history value defaults the total time span to one day as noted in the previous attribute description.

Specifying the Last Plot Value

The end-time attribute determines the last plot value as:

If End-time is...	Then...
last-sample	The trend chart ends at the most recent value in any of its plot's histories.
current-time	The current time, rather than the most recent historical value, marks the last plot value.
time-interval	Prevents the trend chart from scrolling, and directs G2 to ignore any settings for the jump-scroll?, and jump-scroll-interval attributes. A *time-interval* refers to an interval of time since G2 started, *not* an absolute time specification.
A time earlier than the current time, or what is currently displaying	G2 redisplays the values that the trend chart previously displayed at the time designated by end-time.

The end time of the chart gets rounded by the jump-scroll-interval to align with the next interval that aligns with the number and time. For example, if the last history plotted was received at 6:59.9, the end-time attribute rounds to 7:00 so that all labels are legible.

Because of rounding the end time of the trend chart, early data points may not be shown when data-window-time-span is show-all-history, total-time-span is same-as-window, and jump-scroll-interval is automatic.

Updating the Trend Chart Data

The display-update-interval attribute defines how frequently G2 updates the data on the trend chart. This value is independent of the update interval associated with, for instance, the variable whose value is being displayed on the trend chart.

Specifying How Data Scrolls

The jump-scroll? attribute specifies how data scrolls in the data window.

A trend chart with a fixed data-window-time-span attribute and an end-time attribute specified as last-sample or current-time needs to scroll data out of the

data window to accommodate new data. Other trend charts do not scroll. Two ways to scroll data in the data window are:

If jump-scroll? is...	Then the data window scrolls...
yes	At the rate specified in the jump-scroll-interval attribute.
no	The smallest amount possible to accommodate any new data, giving the appearance of smooth scrolling. If no new data is arriving, and end-time is set to last-sample, the data window does not scroll. If end-time is set to current time, the data window will scroll with the passing of time, regardless of whether data is arriving.

Shifting the Data Window

In correlation with the data-window-time-span value, the jump-scroll-interval attribute defines when to shift the data window to the left, based on whether jump-scroll? is set to yes or no.

When jump-scroll-interval is set to automatic, the data window shifts to the left at an interval that G2 determines.

Displaying Current Real-Time Clock Labels

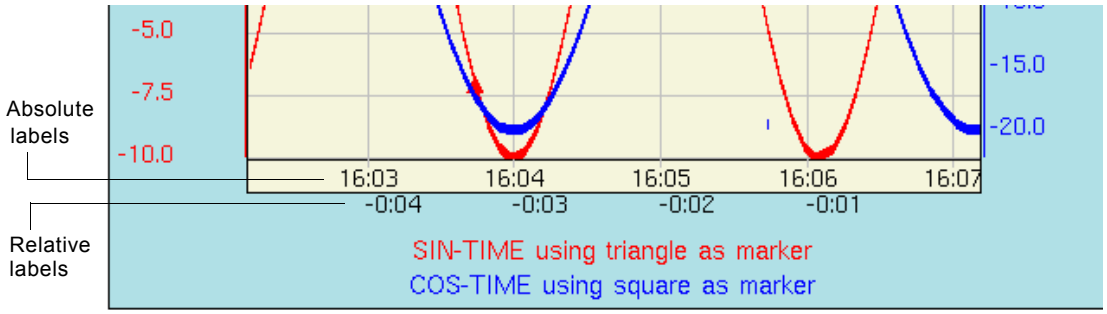
The absolute-labels-visible? attribute specifies where current real-time clock labels are displayed in the data window. The labels scroll with the data.

Displaying Negative Offset Labels

The relative-labels-visible? attribute does two things:

- Specifies that you want relative labels to be displayed on the trend chart.
- Marks negative offsets from the end time of the trend chart.

Relative labels do not scroll with the data. You can display both absolute and relative labels:



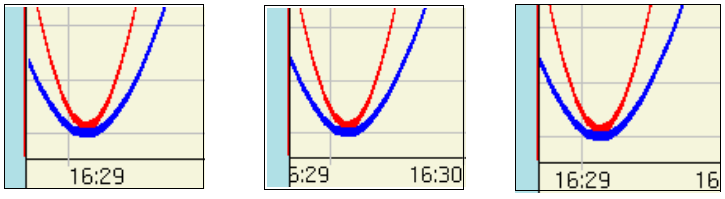
Defining the Label Frequency

The label-frequency attribute determines the interval between time axis labels.

If Label-frequency is...	Then...
none	Time labels are not displayed.
automatic	G2 determines how many labels are appropriate for the time period, and which time labels make sense.
every *time-interval*	G2 places a label at the time interval you specify, unless too many labels would display, in which case G2 adjusts the total number of labels appropriately to fit the time axis.

Specifying the Label Alignment

The label-alignment attribute determines where G2 places the time label relative to the time that it labels on the data window:



Left Right Center

When the grid line is visible, it visually depicts how G2 aligns the time label—the vertical grid line appears in the center or to the left or right of the time label as the diagrams illustrate.

Absence of a grid line does not change the relative placement of the time labels; it is just more difficult to see how the time labels align.

Summarizing Time Axis Attributes

The time axis attributes are:

Attribute	Description
data-window-time-span	Determines the time span that the data window represents. You can base the time span on a specific time interval (fixed), or on the history of values.
Allowable values:	{*time-interval* | show all history}
Default value:	5 minutes
total-time-span	Determines how long a history is maintained in a trend chart. This attribute relates to and is only used for plots whose use-local-history? attribute is set to yes.
Allowable values:	{*time-interval* | same as data window}
Default value:	same as data window
end-time	Determines the last plot value.
Allowable values:	{*time-interval* | last sample | current time}
Default value:	last sample
display-update-interval	Determines how frequently the data on the trend chart is updated. This value is independent of the update interval associated with the variable whose value is being displayed on the trend chart.
Allowable values:	*time-interval*
Default value:	2 seconds

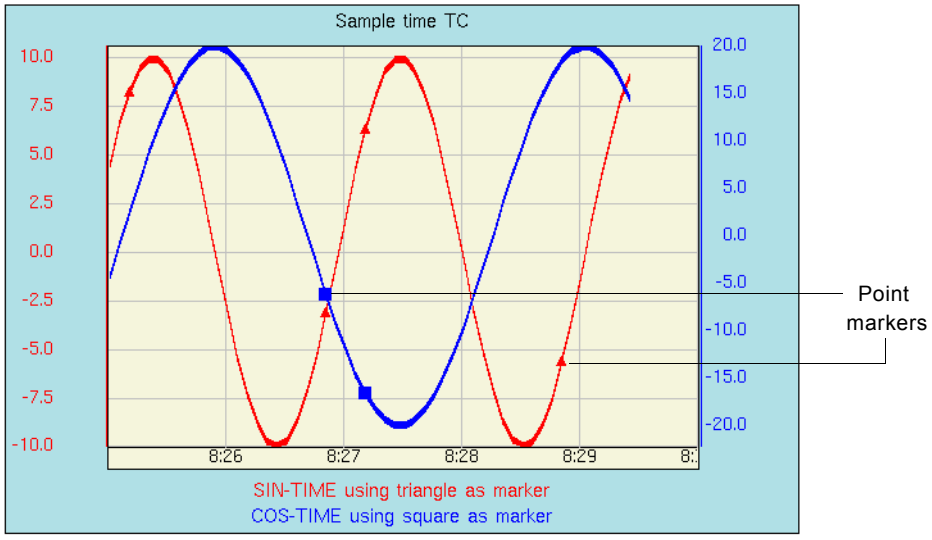
Attribute	Description
jump-scroll?	Determines whether or not the trend chart will jump-scroll (shift the data window from right to left at a certain interval).
Allowable values:	{yes | no}
Default value:	yes
jump-scroll-interval	Determines when to shift the data-window to the left, based on whether jump-scroll is set to yes or no, and then in correlation with the data-window-time-span value.
Allowable values:	{*time-interval* | automatic}
Default value:	automatic
absolute-labels-visible?	Determines whether real-time clock labels are displayed in the data window.
Allowable values:	{yes | no}
Default value:	yes
relative-labels-visible?	Determines whether negative offset labels from the end time of the trend chart are displayed in the data window.
Allowable values:	{yes | no}
Default value:	no
label-frequency	Determines the frequency of the time axis labels.
Allowable values:	{automatic | none | every *time-interval*}
Default value:	automatic

Attribute	Description
label-alignment	Determines whether the time label is placed relative to the time that it labels on the data window.
Allowable values:	{center | right | left}
Default value:	right
time-format	Determines how the time expression is displayed as the time axis label. Note that the dd-hh-mm as an interval format will display the times as negative offsets from the current time. Note that the hh.hh as an interval format may be used to display times relative to G2 start time as hours and decimal fractions of hours. For example, 90 minutes after G2 start time is rendered 1:30 in the "hh.mm as an interval" format; and as 1:50 in the "hh.hh as an interval" format.
Allowable values:	{mmm-yyyy | mmm-dd-yyyy | dd-mmm-yyyy | dd-mm-yy | yy-mm-dd | mm-dd-yy | mm-yy | mm-dd-hh-mm | mmm-dd-hh-mm | mm-ss | hh-mm-ss-am-pm | hh-mm-ss | hh-mm-am-pm | hh-mm | dd-hh-mm as an interval | hh.hh as an interval}
Default value:	hh-mm-ss-am-pm
label-color	Specifies the color of the time label.
Allowable values:	*color-name*
Default value:	black
grid-lines-visible?	Determines whether the time axis vertical grid lines are visible.
Allowable values:	{yes | no}
Default value:	yes

Attribute	Description
grid-line-color	Specifies the color of the vertical grid lines that are displayed for the time axis. The value for this attribute is only applicable if grid lines are visible.
Allowable values:	*color-name*
Default value:	gray
grid-lines-per-label	Determines the number of vertical grid lines that are displayed on the trend chart.
Allowable values:	{0 – 50}
Default value:	1
baseline-visible?	Determines whether the time axis baseline, which is the horizontal line displayed just above the time axis, is visible.
Allowable values:	{yes | no}
Default value:	yes
baseline-color	Determines the color of the time axis baseline. Setting a specific color for this baseline has no effect if the baseline is not visible (previous attribute is set to no).
Allowable values:	*color-name*
Default value:	black

Configuring Point Formats

Point format components specify whether a point marker appears at certain places on a plot. If point markers are visible, the point format attribute determines the style and frequency of markers. The next diagram indicates point markers on a trend chart:



Note Once point markers are visible, the trend chart legend includes a description of which marker styles are in use.

Access point formats from:

- The point formats choice from the trend chart item menu.
- The point-formats attribute on the trend chart attribute table.
- The point-format-name-or-number attribute on a plot subtable.

The attribute tables of the defaults subtable and a point-format subtable are:

point-format defaults table

POINT-FORMATS-TC, a tren...	
a point-format	
Names	DEFAULT
Markers visible?	no
Marker color	black
Marker style	triangle
Marker frequency	every 1 points

point-format table

POINT-FORMATS-TC, a tren...	
a point-format	
Names	none
Markers visible?	yes
Marker color	black
Marker style	triangle
Marker frequency	every 1 points

Displaying Markers

The markers-visible? attribute specifies whether point markers are displayed upon the trend chart. The marker-color attribute defines the point marker color.

Specifying the Marker Style

The marker-style attribute specifies what type of point marker to use. You can choose among several marker styles (square, plus-sign, or triangle) or the current value of the plot. Alternatively, you can use an object icon as a marker, as long as the icon consists of a single layer or is monochrome.

Defining the Marker Frequency

The marker-frequency attribute defines when, in relation to a number of data points, point markers are displayed on the trend chart. You specify that frequency as:

every *integer* points

The Effect of Markers on Trend Chart Drawing

When markers are not displayed, G2 draws to a trend chart only when the result would change the appearance of the chart. This optimization does not occur when markers are displayed: G2 then draws every data point, even if a point already exists at the same location.

Summarizing Point Format Attributes

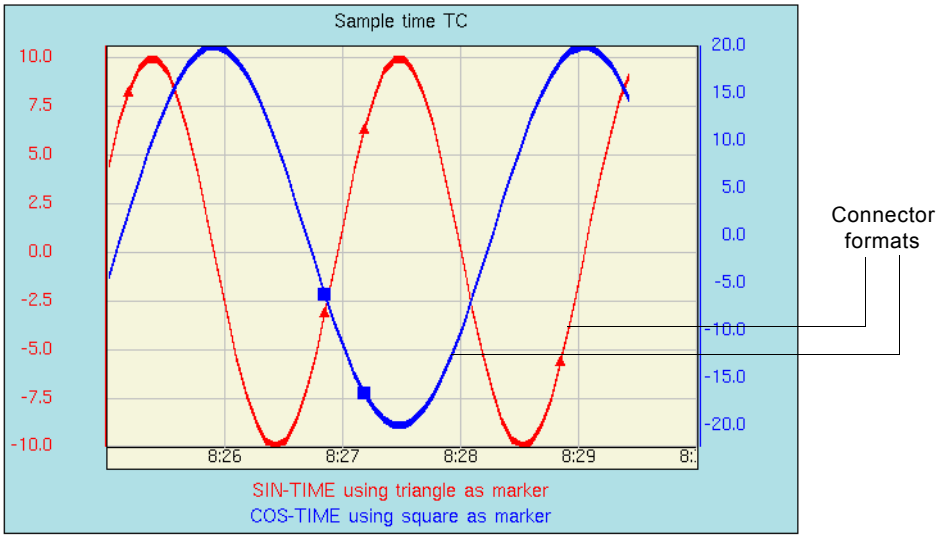
The point format attributes are:

Attribute	Description
names	Provides an optional name for the point format. This attribute is not on the defaults subtable.
Allowable values:	*symbol*
Default value:	none

Attribute	Description
markers-visible?	Determines whether the trend chart displays point markers.
Allowable values:	{yes | no}
Default value:	no
marker-color	Determines the point marker color.
Allowable values:	*color-name*
Default value:	black
marker-style	Specifies what type of point marker to use.
Allowable values:	{square | plus-sign | triangle | icon of *object-class* | current value}
Default value:	triangle
marker-frequency	Determines when to display a point marker.
Allowable values:	every *integer* points
Default value:	every 1 points

Configuring Connector Formats

Connector format components effectively provide a visual interpretation of plot values (data points), by determining how plots are drawn on the trend chart. A connector format determines the way in which G2 interpolates the plot values as they become available. The next diagram illustrates the connector formats in use on a trend chart:



Access connector formats from:

- The connector formats choice from the trend chart item menu.
- The connector-formats attribute on the trend chart attribute table.
- The connector-format-name-or-number attribute on a plot subtable.

The attribute tables of the connector format default menu and of a connector format table are:

connector-format defaults table

CONNECTOR-FORMATS-TC, a t...

a connector-format	
Names	DEFAULT
Connectors visible?	yes
Connector line color	black
Connector interpolation	linear
Connector line width	1
Connector shading target	none

connector-format-table

CONNECTOR-FORMATS-TC, a t...

a connector-format	
Names	none
Connectors visible?	yes
Connector line color	black
Connector interpolation	linear
Connector line width	1
Connector shading target	none

Displaying Connectors

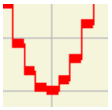
The connectors-visible? attribute specifies whether connectors are drawn between plots on the trend chart. If point markers are not in use and this attribute is set to no, a plot is effectively hidden on the trend chart.

The connector-line-color attribute specifies the line color when connectors are visible.

Specifying How Connectors are Drawn

The connector-interpolation attribute specifies the way in which G2 draws the connection between two successive points on a plot.

The default value, linear, indicates that G2 draws a continuous connector between two points when two values are available. The first sample connector here shows a linear connector. (The connector line width in these examples is set to 7.)

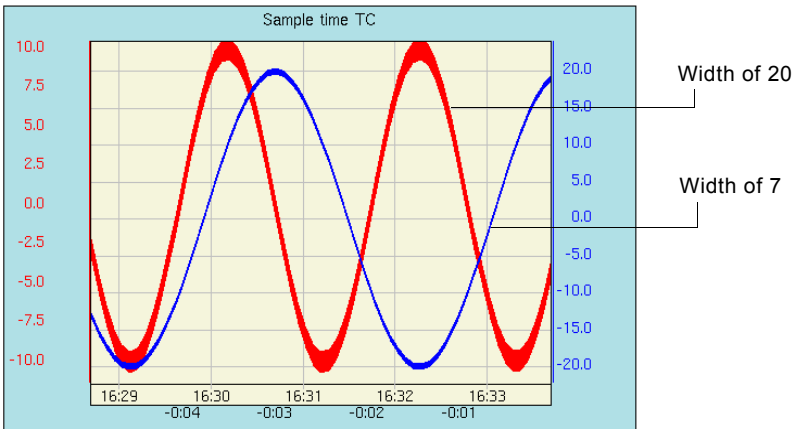


Specifying sample-and-hold interpolation directs G2 to draw a connector line when a single value is available, and to draw a riser and a second connector when the second value is available. Sample and hold implies that G2 draws a riser between the two connectors. The second sample connector shows the same plot as the previous diagram, but with a sample-and-hold connector.

Specifying sample and hold, without risers draws the connector lines only at the time a value is received. The third sample

Specifying the Connector Line Width

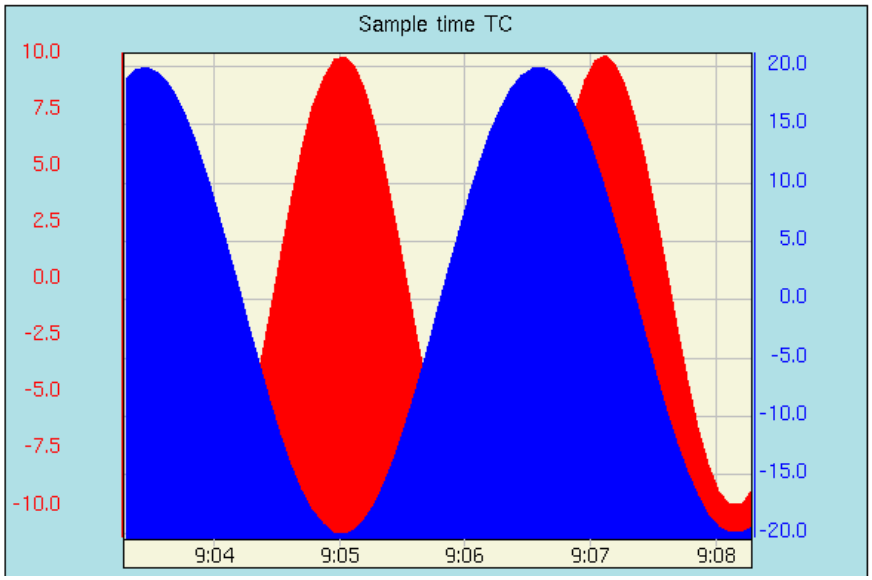
The connector-line-width attribute specifies the width of the connector line in workspace units. The width is measured vertically. The next diagram shows part of the sample trend chart with two plots and connectors of different widths.



Displaying Block Shading

The connector-shading-target attribute specifies whether connectors have block shading. The default is none. Changing the value to bottom or top, directs G2 to shade from wherever the connector point exists to either the top or bottom of the data window, rather than connecting the data points with lines.

The diagram here shows a portion of the trend chart used in the other examples with the connector-shading-target attribute of two plots set to bottom.



When using connector-shading-target, plots appear to be layered, one on top of the other. The bottom layer of a trend chart consists of the grid lines. Plots are drawn from back to front in the order they appear on the menu — plot #1 is the first layer after the grid lines, plot #2 is next, and so on. You cannot change the layering order of plots.

Summarizing Connector Format Attributes

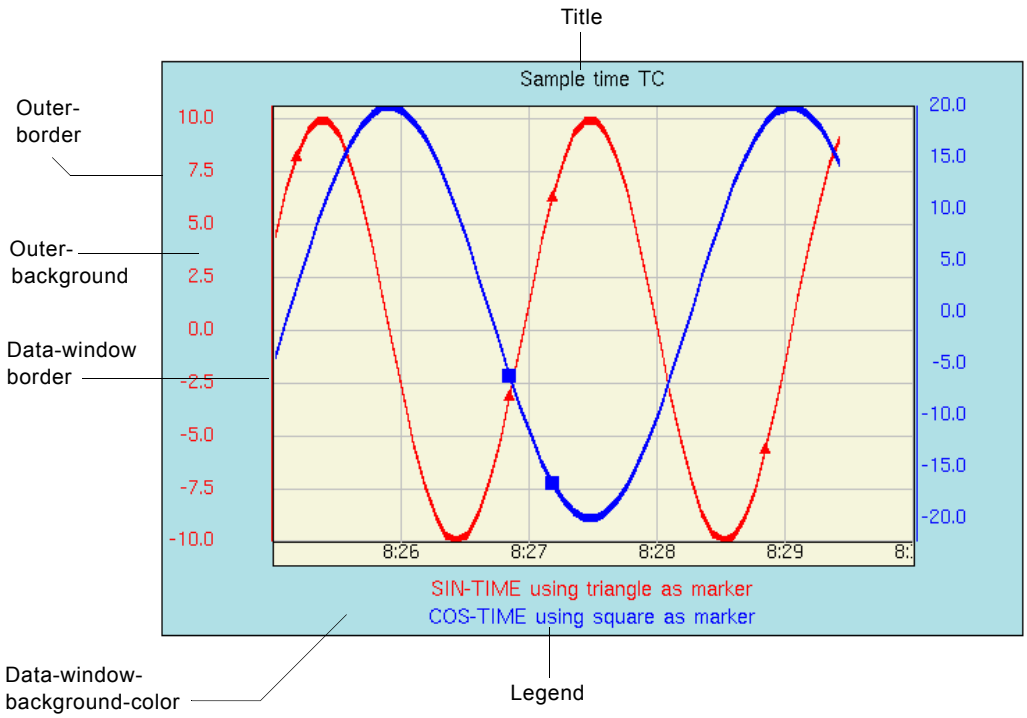
The connector format attributes are:

Attribute	Description
names	Indicates the optional name of the connector format. This attribute is not on the defaults subtable.
Allowable values:	*symbol*
Default value:	none
connectors-visible?	Determines whether connectors are drawn between plots on the trend chart.
Allowable values:	{yes | no}
Default value:	yes
connector-line-color	Determines the color of the plot as it is drawn on the trend chart. Setting this attribute has no effect if the connectors are not displayed.
Allowable values:	*color-name*
Default value:	black
connector-interpolation	Determines the way in which G2 draws the connection between two successive points on a plot.
Allowable values:	{linear | sample and hold [,without risers] }
Default value:	linear

Attribute	Description
connector-line-width	Determines the width of the connector line in workspace units. The width is measured vertically. When a workspace is at full size, there are approximately 100 workspace units per inch.
Allowable values:	*integer*
Default value:	1
connector-shading-target	Determines whether connections are drawn with block shading. Setting this attribute has no effect if connectors are not visible.
Allowable values:	{none | bottom | top}
Default value:	none

Configuring the Trend Chart Format

The trend chart format compound attribute lets you define options for the entire trend chart that you are creating. The next diagram labels the various trend chart components that you can change with the trend chart format subtable.



Access the trend chart format subtable from:

- The trend chart format subtable choice from the trend chart item menu.
- The trend-chart-format attribute from the trend chart attribute table.

Displaying an Outer Border

The outer-border-visible? attribute specifies whether the border is displayed. If it is, you can set its color in the outer-border-color attribute.

The outer-background-color determines the color of the border surrounding the data window.

Displaying a Data Window Border

The data-window-border-visible? attribute determines whether G2 draws a border around the data window. If a border is displayed, the data-window-border-color attribute sets its color.

The data-window-background-color attribute specifies the background color within which all plots are displayed.

Adding a Trend Chart Legend

The legend-visible? attribute defines whether a legend is displayed. If it is, you can set its color with the legend-color attribute, and its position with the legend-position attribute.

Providing a Trend Chart Title

The title-visible? attribute specifies whether the trend chart displays a title. The text of the title is the value of the trend chart's names attribute. If a title is displayed, you can set its color with the title-color attribute and its position, above or below the trend chart, with the title-position attribute.

Summarizing Trend Chart Format Attributes

The trend chart format attributes are:

Attribute	Description
outer-border-visible?	Determines whether the outer border of the trend chart is visible. The outer border is the outline drawn on the outer-most edges of a trend chart.
Allowable values:	{yes | no}
Default value:	yes
outer-border-color	Specifies the color of the trend chart outer border. Setting this attribute has no effect if the outer border is not visible.
Allowable values:	*color-name*
Default value:	black

Attribute	Description
outer-background-color	Specifies the outer background color of a trend chart, which is the area surrounding the data window.
Allowable values:	*color-name*
Default value:	smoke
data-window-border-visible?	Determines whether the data window border is visible. The data window border is the line surrounding the outer-most edge of the data window.
Allowable values:	{yes | no}
Default value:	no
data-window-border-color	Specifies the color of the data window border. Changing this value has no effect if the data-window-border-visible? attribute is set to no.
Allowable values:	*color-name*
Default value:	black
data-window-background-color	Specifies the background color of the trend chart data window.
Allowable values:	*color-name*
Default value:	smoke
legend-visible?	Determines whether to display a legend for the trend chart.
Allowable values:	{yes | no}
Default value:	yes

Attribute	Description
legend-color	Determines the color of the legend. You can display the entire legend in one color, or have the legend for each plot match the plot color (determined in the plot's connector-format).
Allowable values:	{*color-name* | match plot colors}
Default value:	match plot colors
legend-position	Determines the position of the legend, if it is visible.
Allowable values:	{below | above}
Default value:	below
title-visible?	Determines whether to display the title of the trend chart (entered as a symbol on the trend chart's attribute table).
Allowable values:	{yes | no}
Default value:	yes
title-color	Specifies the color of the trend chart title, if it is visible.
Allowable values:	*color-name*
Default value:	black
title-position	Determines the position of the trend chart title if it is visible.
Allowable values:	{below | above}
Default value:	above

Working with Trend Charts

Trend charts provide a dynamic and flexible means to represent data values visually. Here are some tips about working with trend charts.

Updating Trend Charts

Like other display items, G2 updates trend chart displays only when they are visible in the current window. An item is visible in the current window if the workspace upon which it resides is not hidden, even though it may be positioned below another.

If the workspace upon which a trend chart resides is hidden, G2 continues to update the history values, but not the display.

How Plots are Drawn

Plots are drawn from back to front in the order that they appear on the component menus. The bottom layer of a trend chart consists of the grid lines followed by plot #1, plot #2, and so on. You cannot change the order of the trend chart plots.

Causes of Redrawing and Reformatting

Changing certain component attributes of a trend chart causes G2 to either redraw or reformat the trend chart. Differentiating between redrawing and reformatting is important, since each causes different effects.

Repainting a trend chart redisplays the same history values while incorporating some visual change. For example, changing the size of the trend chart or the color of a connector-format component causes G2 to redraw the entire trend chart. If a trend chart has a large number of values to plot, redrawing may not be immediate, depending on other system settings and the KB's current processing load.

Reformatting recomputes certain aspects of the trend chart, such as the value axis and the layout of the trend chart's components. When a trend chart is plotting local history values, changing a plot expression or the value of the use-local-history? attribute causes all history values to be lost and the trend chart to be reformatted. After reformatting, G2 redraws the trend chart to reflect all changes.

System Procedures for Trend Charts

The system procedures specifically for use with trend charts are:

To do this...	Use this system procedure...
Add a new trend chart component.	g2-add-trend-chart-component
Delete a new trend chart component.	g2-delete-trend-chart-component
Get the complete annotation text of a trend chart component.	g2-get-text-of-trend-chart-component
Change the complete annotation text of a trend chart component.	g2-set-text-of-trend-chart-component

For a complete description of these and other G2 system procedures, see the *G2 System Procedures Reference Manual*.

Trend Chart Attributes Reference

Because there are so many formatting attributes associated with trend charts, you may not remember on which component subtable a particular attribute exists. The following figures present each of the compound attribute subtables.

Note In expressions, when referring to trend chart attributes that include question marks, precede the question mark with an @ sign. For example:
use-local-history@?

See these sections for detailed descriptions:

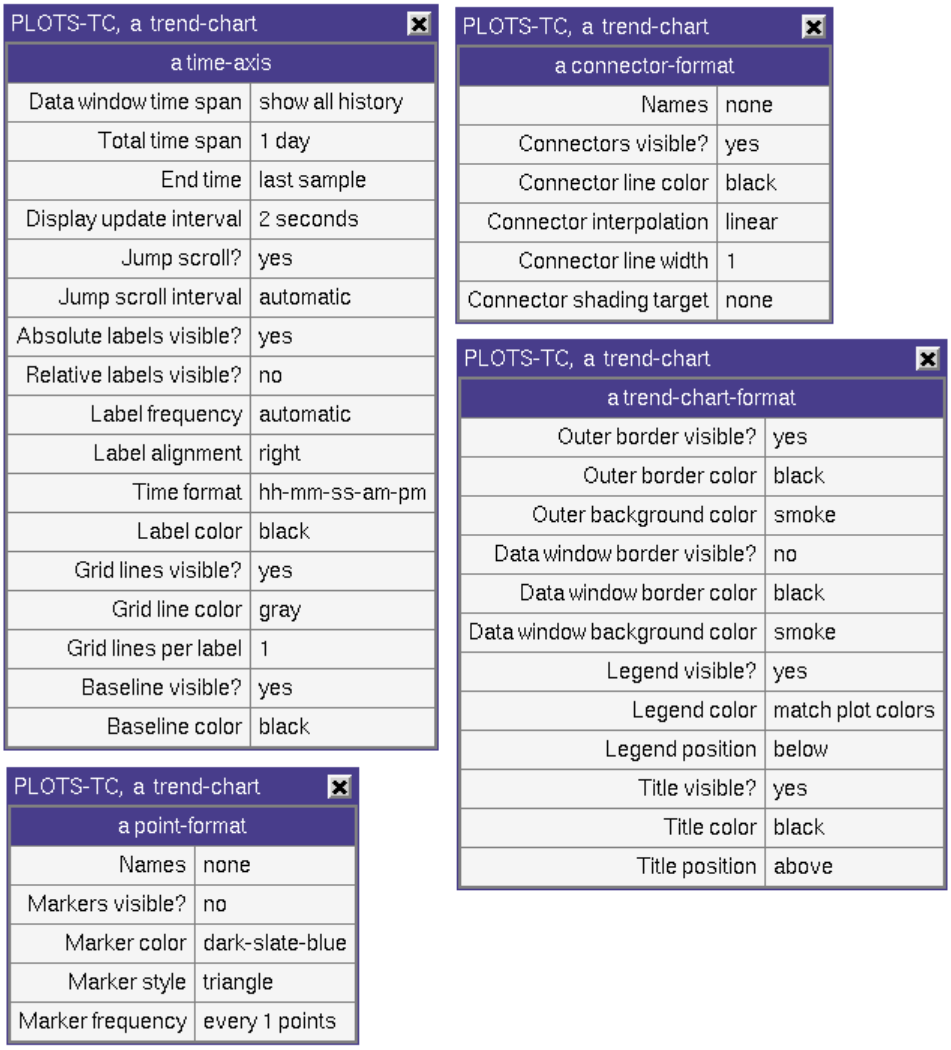
- Configuring Plots.
- Configuring Value Axes.

PLOTS-TC, a trend-chart	
a plot	
Names	none
Use local history?	yes
Value axis name or number	2
Point format name or number	1
Connector format name or number	1
Update interval	5 seconds
Wait interval	2 seconds
Update priority	2
May request data seeking?	yes
May request event updates?	yes
Use simulator?	no
Tracing and breakpoints	default
Include in legend?	yes
Expression	

PLOTS-TC, a trend-chart	
a value-axis	
Names	none
Value axis visible?	yes
Value axis position	right
Range mode	autoranging on data window
Range bounds	none
Range slack percentage	10
Labels visible?	yes
Label color	black
Label frequency	automatic
Show labels as percent?	no
Significant digits for labels	4
Grid lines visible?	yes
Grid line color	gray
Grid lines per label	1
Extra grid lines	none
Baseline visible?	yes
Baseline color	black

See these sections for detailed descriptions:

- Configuring Plots
- Configuring Value Axes



PLOTS-TC, a trend-chart

a time-axis	
Data window time span	show all history
Total time span	1 day
End time	last sample
Display update interval	2 seconds
Jump scroll?	yes
Jump scroll interval	automatic
Absolute labels visible?	yes
Relative labels visible?	no
Label frequency	automatic
Label alignment	right
Time format	hh-mm-ss-am-pm
Label color	black
Grid lines visible?	yes
Grid line color	gray
Grid lines per label	1
Baseline visible?	yes
Baseline color	black

PLOTS-TC, a trend-chart

a connector-format	
Names	none
Connectors visible?	yes
Connector line color	black
Connector interpolation	linear
Connector line width	1
Connector shading target	none

PLOTS-TC, a trend-chart

a trend-chart-format	
Outer border visible?	yes
Outer border color	black
Outer background color	smoke
Data window border visible?	no
Data window border color	black
Data window background color	smoke
Legend visible?	yes
Legend color	match plot colors
Legend position	below
Title visible?	yes
Title color	black
Title position	above

PLOTS-TC, a trend-chart

a point-format	
Names	none
Markers visible?	no
Marker color	dark-slate-blue
Marker style	triangle
Marker frequency	every 1 points

See these sections for detailed descriptions:

- Configuring the Time Axis
- Configuring Point Formats
- Configuring Connector Formats
- Configuring the Trend Chart Format

41

Windows Menus

Provides examples of how to create Windows menus in Telewindows by using one of two techniques: rendering native GMS menus in Telewindows and using the Native Menu System.

Introduction

G2 provides tools for creating standard end-user interfaces for G2 applications when viewed through Telewindows. The **Native Menu System (NMS)** allows you to create custom pulldown menus and popup menus by:

- Rendering menus created using the G2 Menu System (GMS) as standard Windows menus, when viewed through Telewindows.
- Providing an API for creating and manipulating Windows menus, using G2 system procedures.

To create custom menus, the G2 developer can choose to use GMS, which provides a graphical interface, or G2 system procedures, which provides a

programmatic interface. Both techniques support standard menu features such as menu bars, submenus, and popup menus.

While you can use the NMS API to implement almost everything that you can implement using GMS, GMS provides a more intuitive, graphical environment for defining menus. In addition, it provides built-in tools that the NMS API requires specific programming to accomplish. Which approach you use depends on your preference.

For detailed information on how to create GMS menus, see the *G2 Menu System User's Guide*.

For a detailed description of the Native Menu System API, see Native Menu System (NMS) API in the *G2 System Procedures Reference Manual*.

By default, GMS menus render as standard menus when viewed through Telewindows. You can also choose to display GMS menus in Telewindows, using their classic G2 interface.

Note The total number of native (NMS) menus and menu items in existence at one time is limited by G2 to around 30,000 per Telewindows connection. This limit comes from a Windows limitation. The exact limits depend on various factors, including which version of Windows you are running, the amount of physical memory installed, and registry settings. The limits are in the neighborhood of 10,000 and includes native windows, menus (but not menu items), and bitmaps, over all processes on the machine. The integer handles used by the native user interface routines in G2 are limited by the largest integer, around 500 million.

Comparison between Native GMS, Classic GMS, and NMS Menus

The menus that you create using GMS or the NMS API are identical; both render as standard menus when viewed through Telewindows. When you view classic GMS menus through Telewindows, they appear just as they did in classic G2 or Telewindows.

This chart summarizes the features that each menu type supports:

Feature	Native GMS Menus	Classic GMS Menus	NMS Menus
Available on Windows and UNIX		✓	
Render as native Windows menus	✓		✓
Render as classic KB workspaces		✓	
Menu bars	✓	✓	✓

Feature	Native GMS Menus	Classic GMS Menus	NMS Menus
Alternate menu bars	✓	✓	✓
Submenus	✓	✓	✓
Popup menus	✓	✓	✓
Localized menu choice labels	✓	✓	✓
Menu choice keys	✓	✓	✓
Callbacks for individual menu choices	✓	✓	✓
Callbacks for menu hierarchies	✓	✓	✓
Enabling/disabling menu choices	✓	✓	✓
Checking/unchecking menu choices	✓	✓	✓
Choosing one from a set of menu choices (radio choices)	✓	✓	✓
Help text		✓	✓
Color	✓	✓	✓
Character underlining	✓	✓	✓
Right justification	✓	✓	✓
Separators	✓	✓	✓
Groupings	✓	✓	✓
Icons	✓	✓	✓
Dynamic pulldown menus and popup menus	✓	✓	✓
Mouse tracking		✓	

Using Native G2 Menu System (GMS) Menus

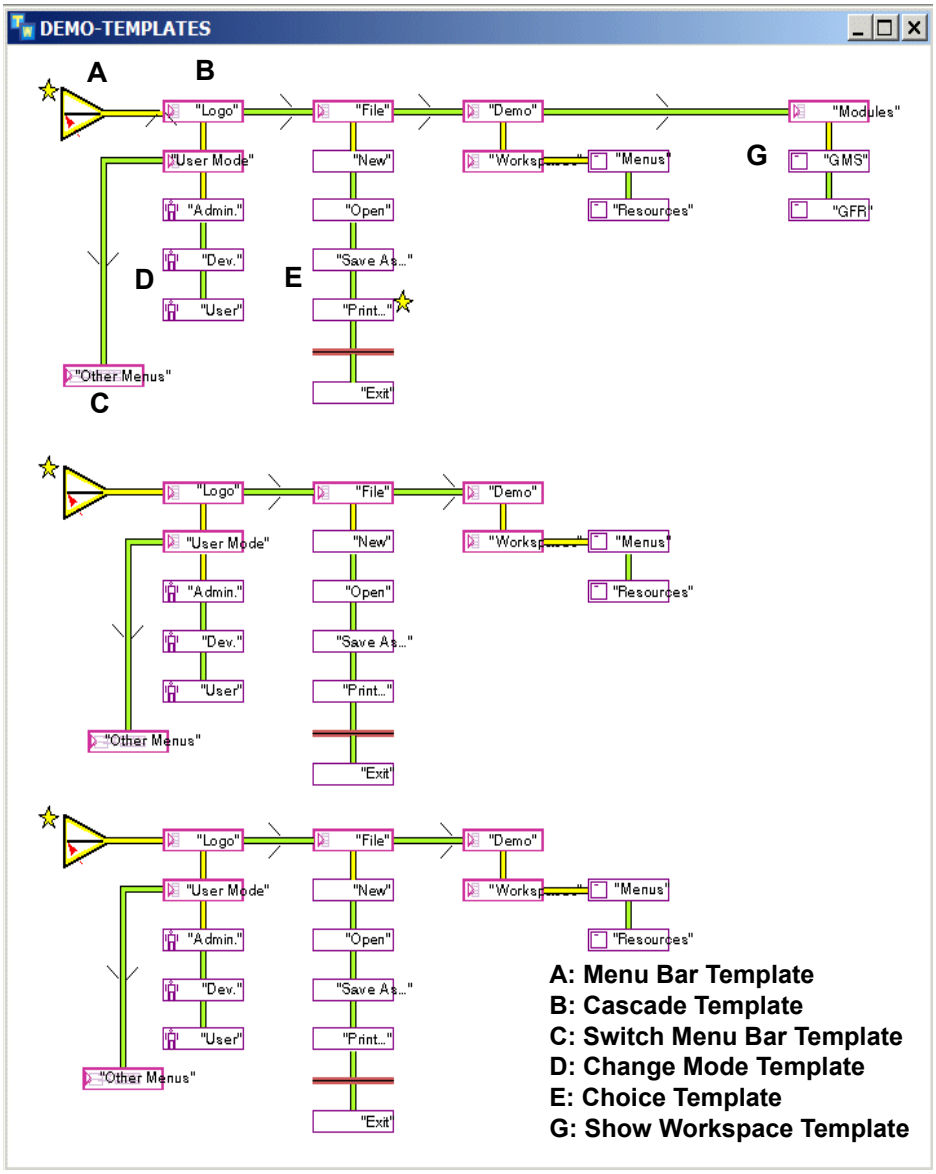
To use native GMS menus, you simply load a KB that defines GMS menus; the menus automatically render as standard menus when viewed through Telewindows.

Following are examples of GMS menu specifications and the resulting menus when viewed through Telewindows. These examples do not describe how to create the menus, using GMS; they merely show examples of various features of GMS and how they render in Telewindows.

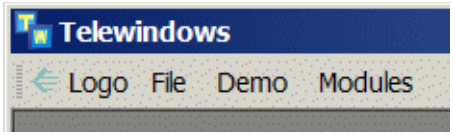
For detailed information on how to create GMS menu specifications, see the *G2 Menu System User's Guide*.

Example: Alternate GMS Menu Bar

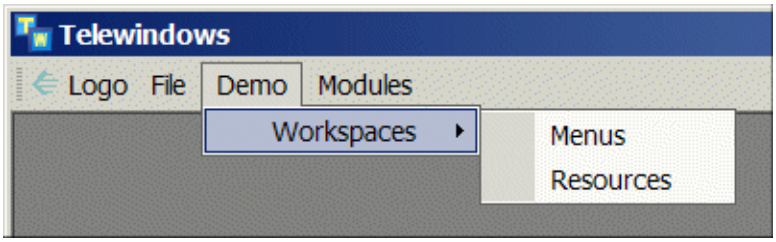
Below is a GMS menu specification that provides three alternate menus. The specification uses GMS menu bar templates, switch menu bar templates, cascade templates, change mode templates, show workspace templates, choice templates, and separators, all of which are labeled and described below.



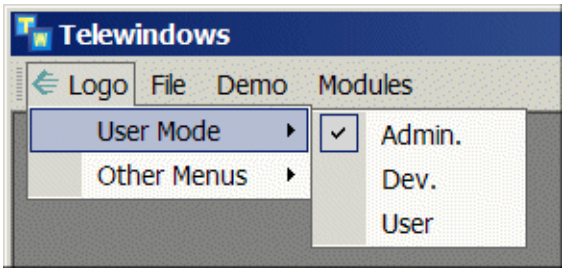
Here is the menu bar that appears when you load the KB and start G2:



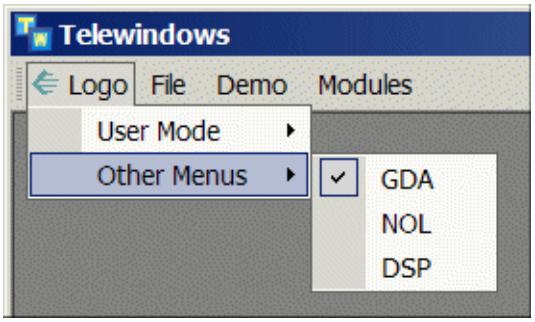
Here is an example of a cascading menu:



This figure shows a cascading menu that sets the user mode of the KB by choosing one from among several choices:

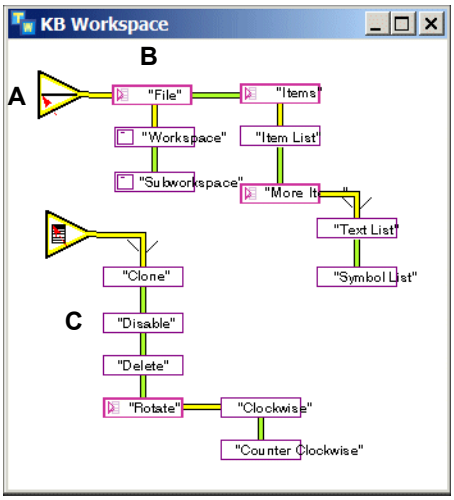


This figure shows a menu choice that lets you choose an alternate menu bar:



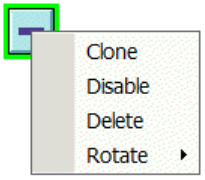
Example: GMS Popup Menu

This figure shows a GMS popup menu specification, which uses a GMS popup menu template, cascade menu template, and choice template:



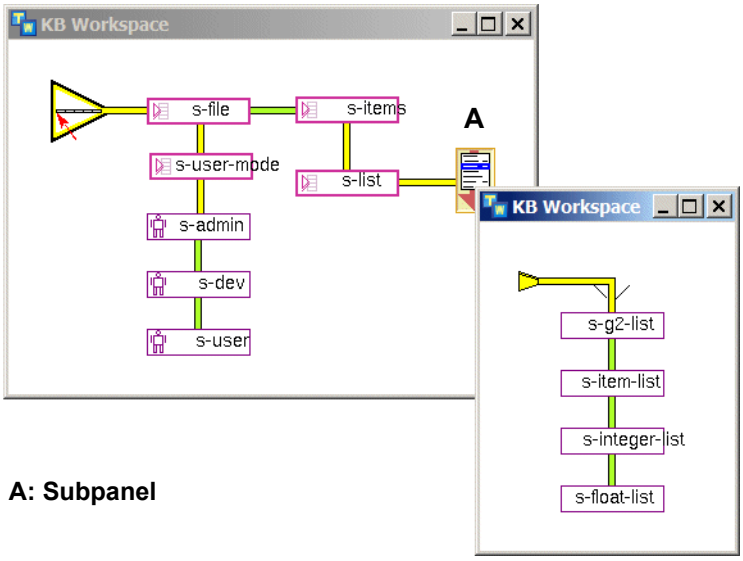
A: Popup Menu Template
B: Cascade Template
C: Choice Template

Here is the popup menu that appears when you click an item:



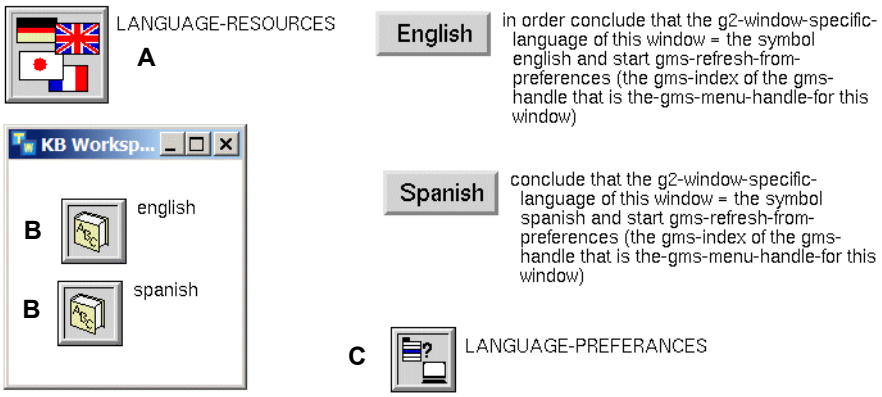
Example: GMS Localization

This figure shows a GMS menu specification that uses local text resources to localize menu labels. The menu specification uses a GMS menu bar template, cascade templates, and change mode templates, as in the other examples. In addition, it uses a GMS subpanel, whose subworkspace contains choice templates. Notice that the labels for each template item represent keys to look up in a local text resource, for example, s-file.



A: Subpanel

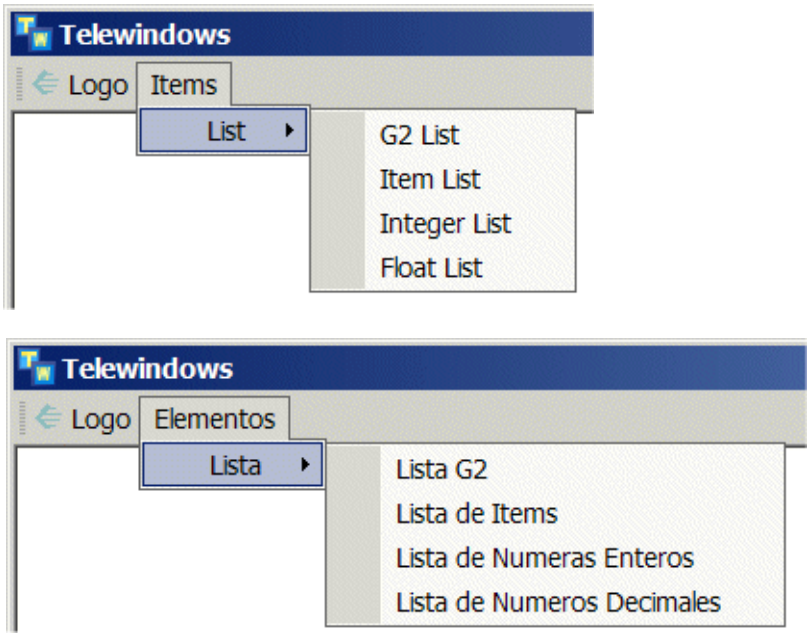
This figure shows a GMS text resource group named language-resources with its associated local text resources on its subworkspace. One text resource uses English as the language and the other uses Spanish. The buttons labeled English and Spanish switch the language by switching the g2-window-specific-language of the GMS preference object, then refreshing the preference.



A: Text Resource Group
B: Local Text Resources
C: GMS Preference

This figure shows the portion of the lookup table that specifies Spanish text strings for keys in the Item menu:

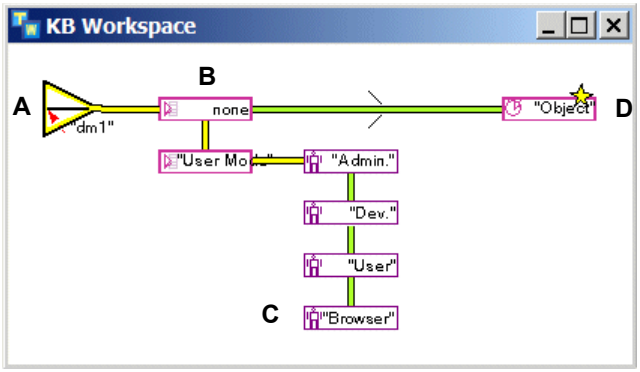
Here are the resulting menu bars in English and in Spanish:



Example: GMS Dynamic Menus

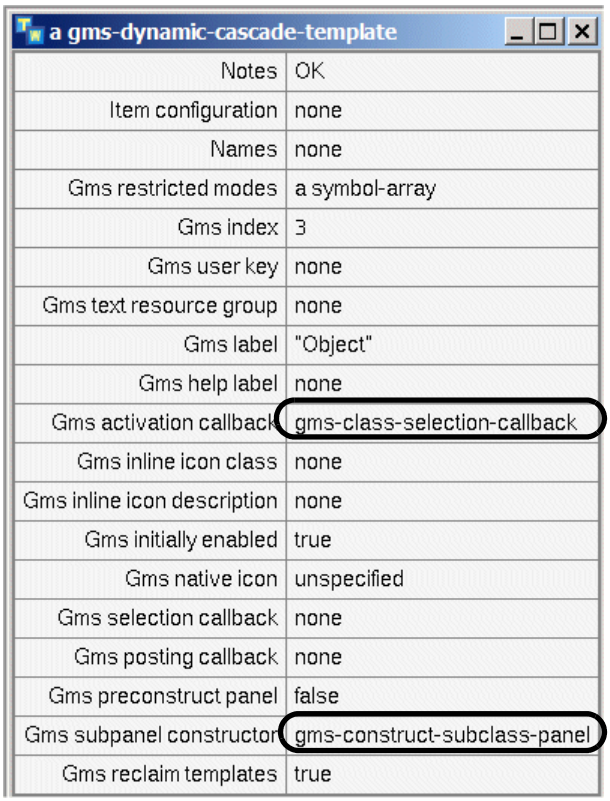
You can use dynamic cascade templates to create GMS menus whose choices update dynamically, based on a callback procedure. When viewed in the standard user interface through Telewindows, these menus render as native Windows menus and update dynamically, just as they do in G2. You can create both dynamic menus and dynamic popup menus.

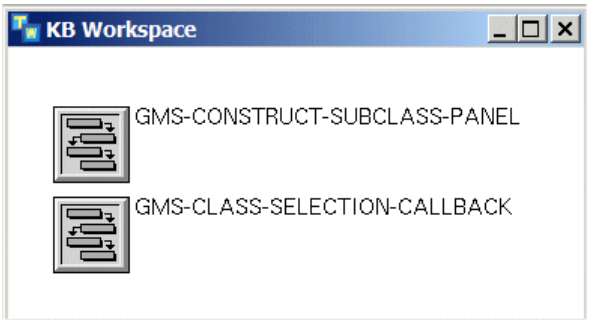
Here is a GMS menu specification, which uses a GMS dynamic cascade template to create a menu whose contents updates, based on a callback:



A: Menu Bar Template
B: Cascade Template
C: Change Mode Template
D: Dynamic Cascade Template

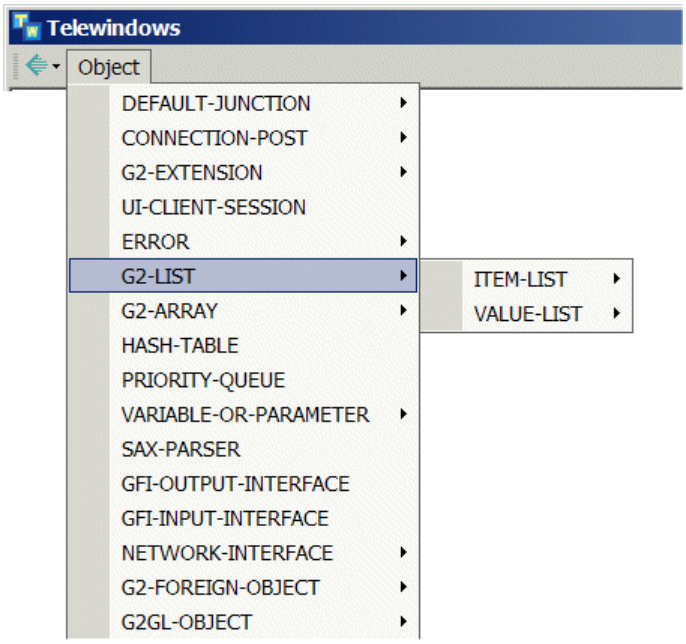
The subpanel constructor collects all the subclasses of the object class, then creates dynamic cascade templates for each subclass that itself has subclasses and choice templates for each leaf node in the class hierarchy. The activation callback creates an instance of the class and attaches it to the mouse. Here is the dynamic cascade template table and the two callbacks:





To see the text of the procedures, load `g2\kbs\gmsdemo.kb`.

Here is the dynamic submenu that appears when you choose the Object menu:

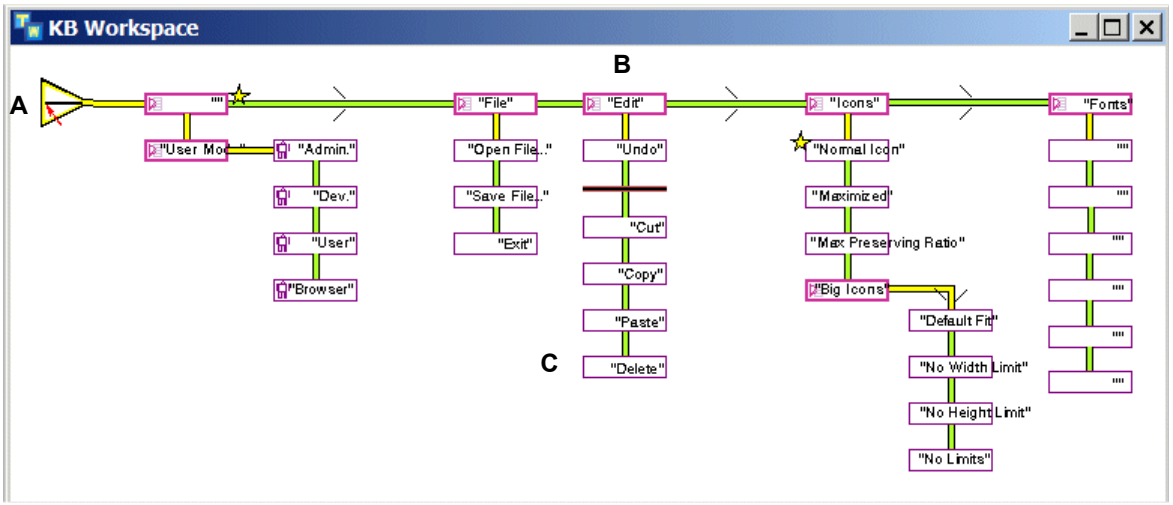


Example: GMS Menu Icons

You can use GMS choice templates to create GMS menu choices that include icons. When viewed in the standard user interface through Telewindows, these menus render as native Windows menus and icons.

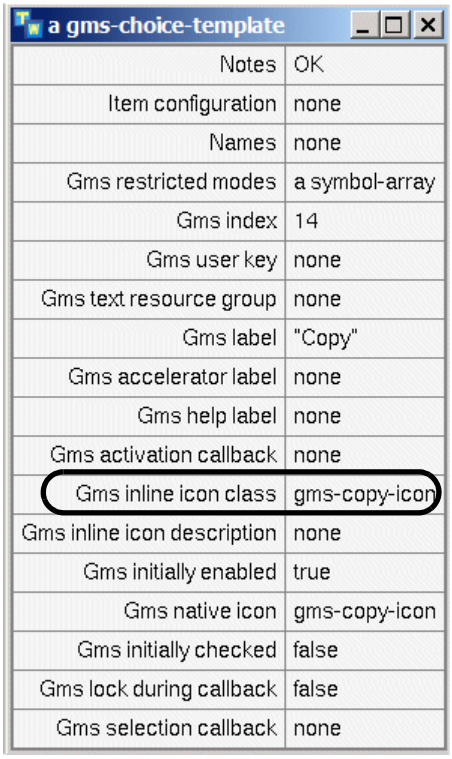
By default, native GMS uses the same icon as standard GMS. To optimize the appearance of the icon in standard GMS, you can specify different icons for native and standard GMS.

Here is a GMS menu specification, which uses a GMS choice template to create a menu choice that includes native Windows icons:

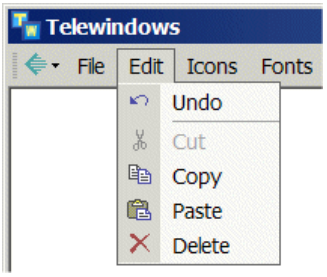


A: Menu Bar Template
B: Cascade Template
C: Choice Template

Here is the table for the choice template that defines the Copy menu choice, which specifies the icon named gms-copy-icon:



Here is the Edit menu, which includes icons for its menu choices:



If the gms-native-icon is the symbol unspecified, then the gms-inline-icon-class is used for both G2 and Telewindows.

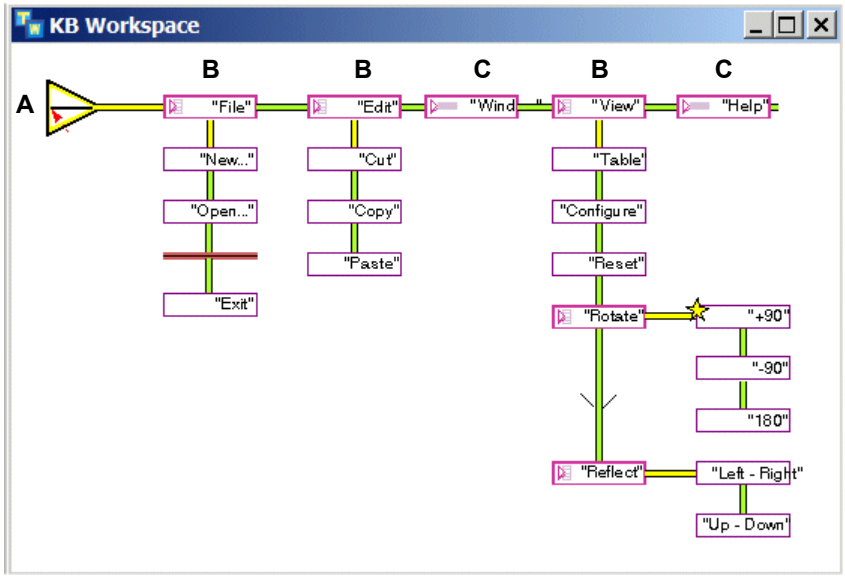
For a list of icons, see Native Menu System (NMS) API in the *G2 System Procedures Reference Manual*.

Example: Built-in G2 Menu

You can use the gms-builtin-template to create one of the built-in G2 menus in Telewindows. The built-in menu only works in Telewindows Next Generation and standard Telewindows; the menu choice is grayed out in classic G2 and Telewindows.

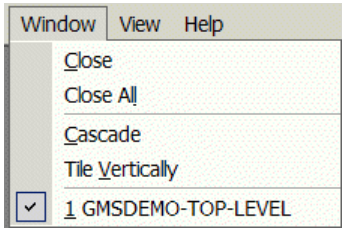
To specify which built-in menu to create, configure the gms-label attribute to be one of these text values: file, edit, view, toolbars, run, tools, window, run-options, or package-preparation, or help.

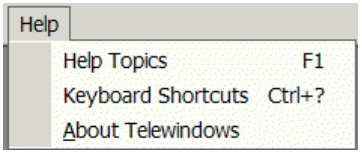
Here is a menu specification that includes two built-in G2 menu templates, the Window menu and the Help menu:



A: Menu Bar Template
B: Cascade Template
C: Built-in Template

Here are the two built-in G2 menus:





See `g2\kbs\gmsdemo.kb` for the example.

Using the Native Menu System API

The Native Menu System (NMS) API provides a low-level set of system procedures for creating and manipulating menus and dialogs in Telewindows. Typically, you use this API to create and manipulate existing G2 objects that you want to render as native Windows menus. For example, you might have built your own menu system in G2, using workspaces. You could use this API to render those "menus" as native Windows menus when viewed through Telewindows. In fact, to implement native GMS menus, we used this API to create a thin layer on top of GMS.

If you have not already created a menu system in G2, using either GMS or workspaces, you can also use this API to create native Windows menus "from scratch."

Note Gensym recommends that you use either native GMS menus or native NMS menus. We do not recommend that you combine native GMS menus with NMS menus that you create either from scratch or as a layer on top of your existing menu objects.

If your G2 application does not define either GMS menus or NMS menus, Telewindows displays the developer menu bar. You can also display the developer menu bar programmatically, using the API.

The `sys-mod.kb` module contains a category of system procedures called G2 Native Menu API, which let you:

- Create and delete menu bars, menus, and submenus.
- Show, hide, and reset menu bars.
- Manage popup menus and menu choices.
- Get and set various information related to menus.
- Get information about the current menus in a given window.

Each API procedure call takes a g2-window as its last argument; thus, you create menu choices for each connected window. Menus and menu choices are represented by "handles," which are positive integers that are unique over a given window.

NMS menus behave like transient items in that clearing or resetting the KB deletes all of them. NMS menus are only available while G2 is running; when G2 is stopped or paused, the menu bar automatically reverts to the developer menu bar. When G2 is resumed, the user-defined menu bar, if any, is restored.

For detailed information on the NMS API, see Native Menu System (NMS) API in the *G2 System Procedures Reference Manual*.

Using the NMS API to Create Menus and Toolbars

You use the NMS API to create two basic types of menus:

- Menu bars — Consist of menus, which are often called "submenus" or "pulldown menus." A menu, in turn, consists of menu choices and/or submenus. A menu that consists of one or more submenus is often called a "cascading menu." Menus and menu choices implement callbacks, which implement the behavior of each menu choice.
- Popup menus — Consist of menu choices and/or submenus.

Both submenus and popup menus are considered menus.

You can also use the NMS API to create dockable toolbars, as well as edit boxes and combo boxes, which you can add to any NMS menu or toolbar. Toolbars are considered a special sort of menu, and edit boxes and combo boxes are considered a special sort of menu choice. You can use the extended callback procedure to access edit and combo box text.

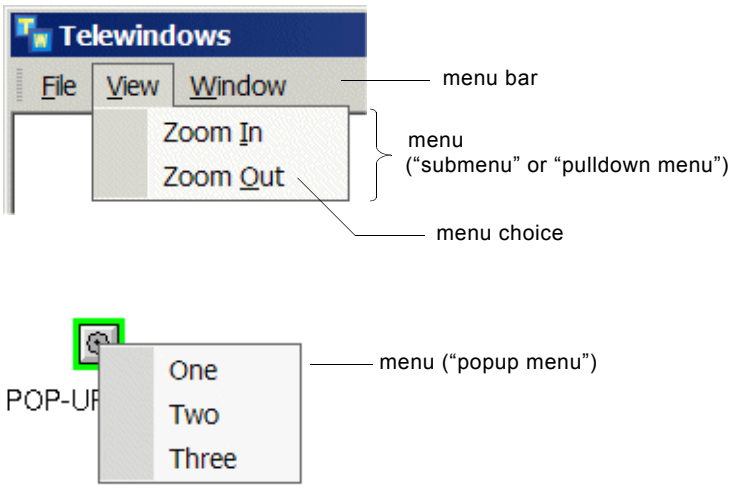
Note Toolbars are only available when using Telewindows Next Generation

(`twng.exe`).

Menu bars, menus, and menu choices exists on a per-window basis, that is, for a single Telewindows connection only. Thus, all of the API calls take a g2-window as the last argument.

Examples of Menu Hierarchies

This figure shows examples of the types of menu hierarchies you can create. The terms in parentheses are the common names for these types of menus, whereas the other terms are the object types that NMS actually creates.



Menu Bars and Popup Menus

To display a menu bar, you create a G2 procedure that performs these sequential steps:

1. Create the menu bar.
2. Create menus and submenus.
3. Add menu choices and submenus to the menus.
4. Add menus to the menu bar.
5. Set the current menu bar.

To display a popup menu in the window, you create a procedure that performs these steps:

1. Creates the popup menu, which is just a menu.
2. Create submenus.
3. Add menu choices and submenus to the popup menu.
4. Display the popup menu.

Callbacks

The Native Menu System supports these types of callback procedures:

- Basic callbacks.

  The basic callback is called when the user clicks a choice in an NMS menu or when the user dismisses an NMS menu.

- Extended callbacks.

  The extended callback is also called when the user clicks a choice or dismisses an NMS menu; however, the extended callback provides the text string from edit and combo boxes.

- Selection callbacks.

  Selection callbacks are called when the user highlights a menu choice by dragging the mouse over the choice or by pressing an arrow key, or when the user unhighlights a highlighted menu choice.

You specify the basic callback procedure:

- When you create a menu bar or menu, using g2-nms-create-menu-bar or g2-nms-create-menu.
- By using the g2-nms-set-callback procedure.

You specify the selection callback procedure by using the g2-nms-set-selection-callback procedure.

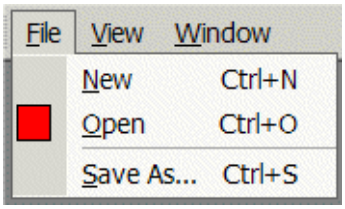
The procedure argument can be a procedure object, a symbol naming a procedure, or the symbol INHERITED, which uses the callback from the parent menu, if any. NMS signals an error if it cannot find the callback procedure when it tries to call it.

Callbacks are also supported in native GMS.

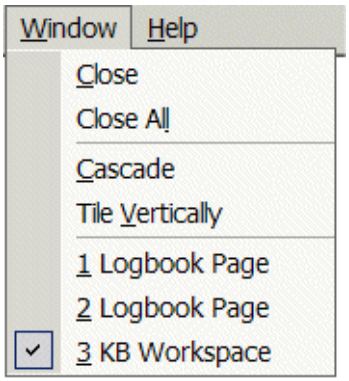
Menu Choices

A menu choice consists of a label and an optional user-defined key. The label can include tabs for right-justifying shortcut key labels such as CTRL+S. The label can also include access keys, such as File, whereby pressing the ALT key plus the access key executes the menu choice, for example, ALT+F. You use the user-defined key to identify the menu choice for further processing. You can get and set various features of the menu choice, including its color, help text, and callback. You can check and uncheck menu choices, select one of some number of menu choices, and enable and disable menu choices.

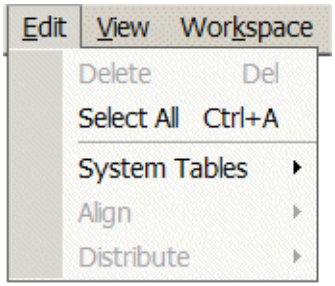
Here are some examples of menu choices:



Menu choices include access keys and right-justified shortcut key labels.



Menu choice is checked.



Menu choices are disabled.

Note By default, Windows always displays menu choices with access keys visible. You can configure Windows to hide these keys until you press the ALT key. To do this, display the Display Properties dialog from the Windows Control Panel and click the Effects tab.

Additional Features

The procedures you write to create NMS menus must check whether the window supports NMS menus by using an API call. If the window does not support NMS menus, your procedure cannot use NMS menus.

The API provides procedures for:

- Creating dynamically updating menus.
- Adding icons to menu choices.
- Adding separators, breaks, and right-justification to a menu.
- Getting and setting menu bars, menus, and menu choices.
- Testing whether an item is a menu bar, menu, or menu choice.
- Testing whether a menu choice is checked.

The NMS API represents menus and menu choices as positive integer handles. Zero is not a valid handle and has a special meaning in some procedures. Handles are unique over a given window, where there is a limit of approximately 32000 handles, per window.

While all menus are represented as integers, Windows makes a distinction between a menu bar and a popup menu. As such, the API provides different procedures for initializing menu bars and popup menus.

NMS menus have certain behaviors with respect to the G2 run state and KB. Starting or restarting G2 initializes all NMS menus on all connected windows. Similarly, clearing the KB deletes all NMS menus on all connected windows. Because callbacks cannot run when G2 is paused, G2 temporarily replaces the user-defined menu bar with the developer menu bar when G2 is paused. When G2 is resumed, the user-defined menu bar is restored.

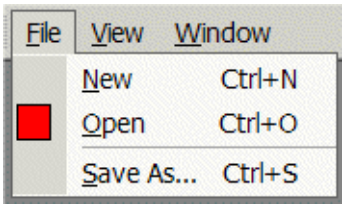
Examples

Following are a number of simple examples that show how to use the NMS API to create and manipulate menus and toolbars.

Note You must delete NMS menus when they are no longer needed, using g2-nms-delete-menu or nms-reset. The following examples omit this important step for simplicity.

Example: Simple Menu Bar

This example builds a simple menu bar. Each menu choice uses nms-demo-callback as its callback. Here is the resulting menu bar:



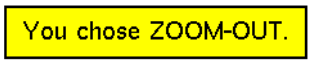
In the example, when using g2-nms-add-choice, a tab character precedes all keyboard shortcuts, such as CTRL+N. You enter a tab character in G2 by entering ALT+I, then pressing the TAB key.

```
simple-menu-bar(win: class g2-window)
menu-bar, file-menu, view-menu, window-menu: integer;
i: integer;
begin
    menu-bar = call g2-nms-create-menu-bar(the symbol
        NMS-DEMO-CALLBACK, win);
    file-menu = call g2-nms-create-submenu(win);
    call g2-nms-add-choice(file-menu, "&New     Ctrl+N", the symbol NEW, win);
    i = call g2-nms-add-choice(file-menu, "&Open     Ctrl+O", the symbol OPEN, win);
    call g2-nms-set-help(i, "Help for the Open command", win);
    call g2-nms-set-colors(i, the symbol white, the symbol red, win);
    call g2-nms-add-separator(file-menu, win);
    call g2-nms-add-choice(file-menu, "&Save As@.@.@.     Ctrl+S",
        the symbol SAVE, win);
    view-menu = call g2-nms-create-submenu(win);
```

```
    call g2-nms-add-choice(view-menu, "Zoom &In", the symbol ZOOM-IN, win);
    call g2-nms-add-choice(view-menu, "Zoom &Out", the symbol ZOOM-OUT, win);
    window-menu = call g2-nms-create-submenu(win);
    call g2-nms-add-choice(window-menu, "&My Window",
        the symbol MY-WINDOW, win);
    call g2-nms-add-submenu(menu-bar, "&File", file-menu, win);
    call g2-nms-add-submenu(menu-bar, "&View", view-menu, win);
    call g2-nms-add-submenu(menu-bar, "&Window", window-menu, win);
    call g2-nms-set-menu-bar(menu-bar, win);
end
```

Example: Simple Callback

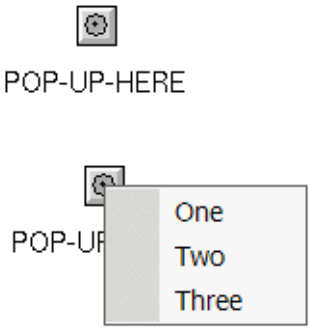
This example provides a callback that simply updates a free text named menu-choice to indicate which menu choice was selected. Here is the text when the Zoom Out menu choice is selected:




```
nms-demo-callback(window: class g2-window, menu: integer, choice: integer,
    path: sequence)
user-data: item-or-value;
begin
    if (choice = 0) then
        change the text of menu-choice to "You dismissed the menu."
    else
        begin
            user-data = call g2-nms-get-key(choice, window);
            change the text of menu-choice to "You chose [user-data].";
        end
end
```

Example: Simple Popup

This example shows how to create a simple popup menu. Each menu choice uses nms-demo-callback as its callback. Here is the item before and after displaying the popup menu:



In the example, when using g2-nms-add-choice, a tab character precedes all keyboard shortcuts, such as CTRL+N. You enter a tab character in G2 by entering ALT+I, then pressing the TAB key.

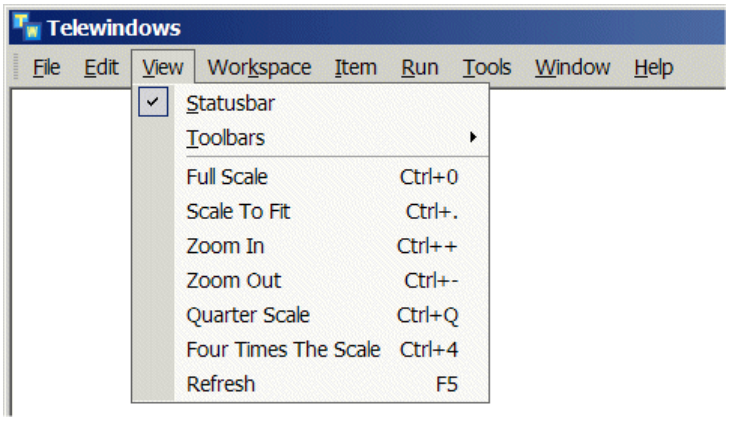
```
simple-popup-menu(win: class g2-window, item: class item)
menu: integer;
x,y: integer;
begin
    menu = call g2-nms-create-menu(the symbol NMS-DEMO-CALLBACK,
        win);
    call g2-nms-add-choice(menu, "One", the symbol ONE, win);
    call g2-nms-add-choice(menu, "Two", the symbol TWO, win);
    call g2-nms-add-choice(menu, "Three", the symbol THREE, win);

    x = call g2-x-in-window(the item-x-position of item, the workspace of item,
        win);
    y = call g2-y-in-window(the item-y-position of item, the workspace of item,
        win);
    call g2-nms-manage-popup-menu(menu, x, y, win);
end
```

Example: Menu Bar with Built-in G2 Menus

This example builds a simple menu bar that includes the built-in G2 View menu, and adds menu choices after the built-in menu choices:

```
simple-menu-bar(win: class g2-window)
menu-bar, file-menu, view-menu, window-menu: integer;
i: integer;
begin
    menu-bar = call g2-nms-create-menu-bar(the symbol NMS-DEMO-CALLBACK,
        win);

    file-menu = call g2-nms-create-submenu(win);
    {Contents of file menu goes here}

    view-menu = call g2-nms-get-built-in-menu(the symbol VIEW, win);
    call g2-nms-add-separator(view-menu, win);
    call g2-nms-add-choice(view-menu, "Zoom &In", the symbol ZOOM-IN, win);
    call g2-nms-add-choice(view-menu, "Zoom &Out", the symbol ZOOM-OUT, win);

    window-menu = call g2-nms-create-submenu(win);
    {Contents of window menu goes here}

    call g2-nms-set-menu-bar(menu-bar, win);
end
```

Here is the resulting View menu:

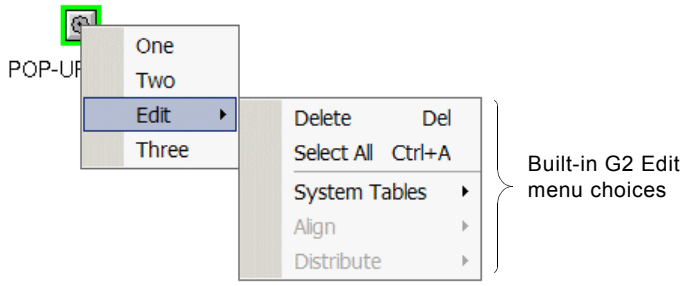


Example: Popup Menu with Built-in G2 Menus

This example builds a popup menu that includes the built-in G2 Edit menu as a submenu, along with several other menu choices:

```
simple-popup-menu(win: class g2-window, item: class item)
menu: integer;
x,y: integer;
begin

    menu = call g2-nms-create-menu(the symbol NMS-DEMO-CALLBACK, win);
    call g2-nms-add-choice(menu, "One", the symbol ONE, win);
    call g2-nms-add-choice(menu, "Two", the symbol TWO, win);

    x = call g2-nms-get-built-in-menu (the symbol EDIT, win);
    call g2-nms-add-submenu(menu, "Edit", x, win);

    call g2-nms-add-choice(menu, "Three", the symbol THREE, win);

    x = call g2-x-in-window(the item-x-position of item, the workspace of item, win);
    y = call g2-y-in-window(the item-y-position of item, the workspace of item, win);
    call g2-nms-manage-popup-menu(menu, x, y, win);
end
```

Here is the resulting popup menu:



Example: Toolbar with Edit Box and Combo Box

The following example is located in `g2-80r0-doc-examples.kb` located in `g2\kbs\demos` (Windows) or `g2/kbs/demos` (UNIX).

This procedure creates a toolbar, then populates it. The toolbar is docked at the default location, at the top of the window below the menu bar.

```
demo-toolbar (win: class g2-window)
tb: integer;
begin
    tb = call g2-nms-create-toolbar ("My Toolbar", the symbol toolbar-callback,
        structure(), win);
    call populate(tb, win);
end
```

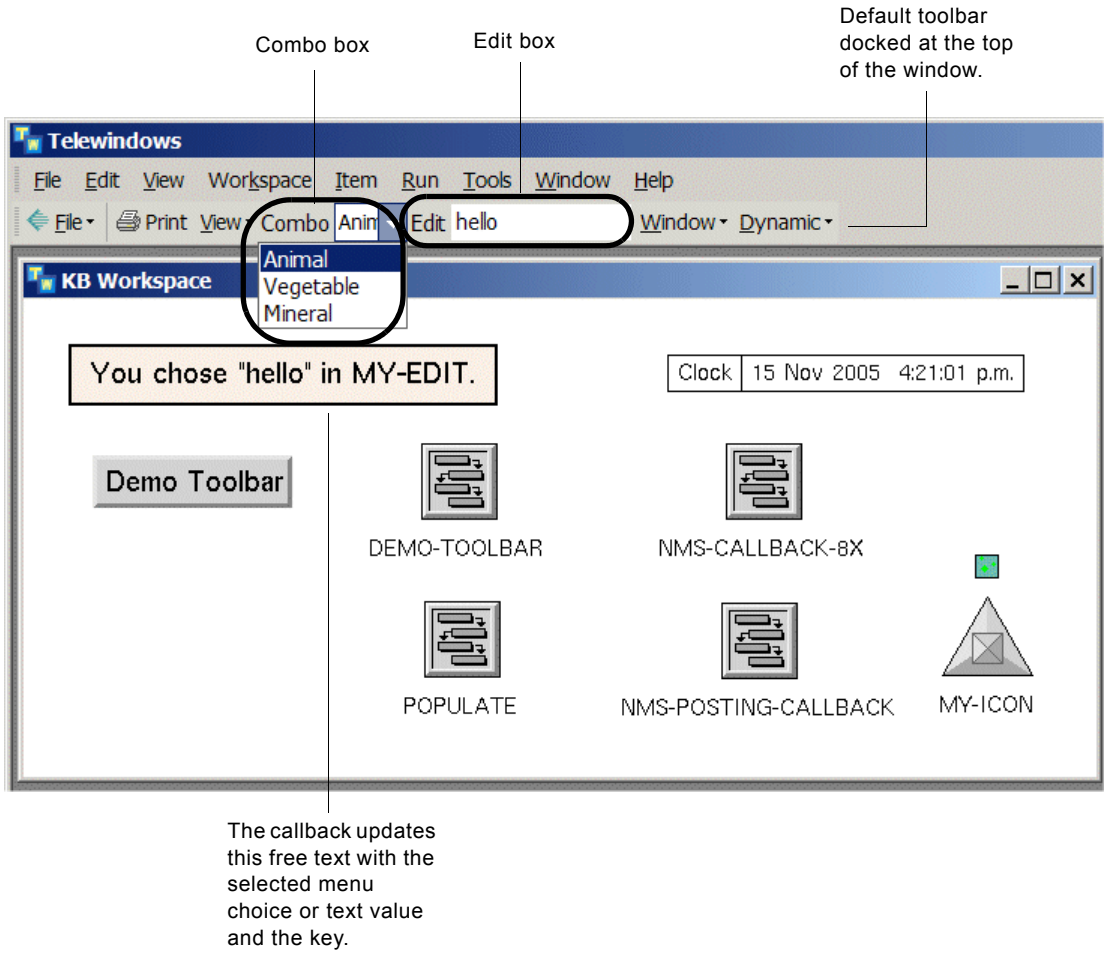
Here is part of the populate procedure, which creates File, View, Window, and dynamic submenus, and which creates and adds a combo box and edit box, each with a key.

```
populate(bar: integer, win: class g2-window)
file-menu, view-menu, window-menu, dynamic-menu: integer;
i, j: integer;
version: float;
begin
    file-menu = call g2-nms-create-submenu(win);
    {contents of file-menu goes here}

    view-menu = call g2-nms-create-submenu(win);
    {contents of view-menu goes here}

    i = call g2-nms-create-combo-box ("Combo", sequence ("Animal", "Vegetable",
        "Mineral"), structure (key: the symbol MY-COMBO, width: 100), win);
     call g2-nms-add-control (bar, i, win);


    i = call g2-nms-create-edit-box ("Edit", structure (key: the symbol MY-EDIT,
        width: 150), win);
    call g2-nms-add-control (bar, i, win);


    window-menu = call g2-nms-create-submenu(win);
    {contents of window-menu goes here}

    dynamic-menu = call g2-nms-create-submenu(win);
    {contents of dynamic-menu goes here}

end
```

Here is the callback procedure that gets called when a menu choice is selected or when the user enters a value in the combo box or edit box. The callback simply updates the free text named menu-choice with the selected choice or text, based on the specified key. If the selected choice has an associated string in the properties structure argument to the callback, the free text displays the text of the edit box or combo box.

```
toolbar-callback (window: class g2-window, menu: integer, choice: integer,
    plist: structure)
userdata: item-or-value;
begin
    if (choice = 0) then
        change the text of MENU-CHOICE to "You dismissed the menu."
    else
    begin
        userdata = call g2-nms-get-key (choice, window);
        if (the string of plist exists) then
        change the text of MENU-CHOICE to "You chose @"[the string of plist]@" in
            [userdata]."
    else
        change the text of MENU-CHOICE to "You chose [userdata].";
    end
end
```

This figure shows the result of clicking the Demo Toolbar button, which executes the demo-toolbar procedure. The toolbar is docked at the top of the window and includes a combo box and edit box. The nms-callback procedure updates the free text with the selected menu choice or text value, in this case, the edit box text "hello".



Displaying Classic GMS Menus in Telewindows

You might want to display GMS menus in Telewindows, using their classic interface. Reasons for doing this include:

- You have already documented your end user interface, using classic GMS menus.
- You require dynamic menus, which only classic GMS supports.
- You want to test how your GMS menu system will look when displayed on UNIX platforms.

The active GMS preference object controls how GMS menus appear in Telewindows.

To display classic GMS menus in Telewindows:

1 Find the active gms-preference and display its table.

You create and configure a GMS preference object when you create your GMS menus. Here is a gms-preference object:

If you have more than one GMS preference, you must determine the active preference programmatically by calling gms-get-current-preference, which takes an integer handle argument and returns the current preference.

For information on how to create a GMS preference and how to get the active preference, see the *G2 Menu System User's Guide*.

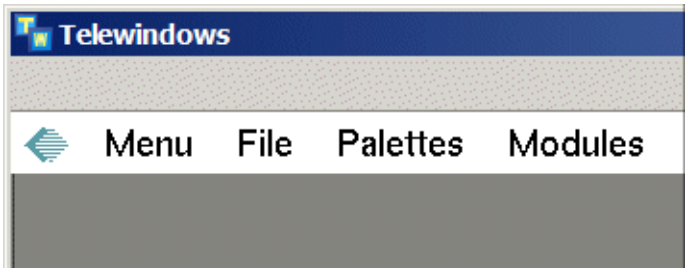
By default, the gms-use-native-menus attribute of the preference is set to true.

2 Set gms-use-native-menus to false.

3 Restart G2.

The GMS menus now appear in Telewindows as they appear in classic G2 or Telewindows.

Here is an example of a classic GMS menu bar displayed in Telewindows:



GMS and NMS Menus and the G2 Run State

A number of differences exist between how native GMS menus, classic GMS menus, NMS menus, and the G2 developer menu bar behave in Telewindows with respect to the G2 run state. The following table compares the behavior of each of these menus under different G2 run states. Native G2 menus are also included for comparison. In the table, menus refer to both menu bars and popup menus.

	When G2 is...			
	Paused...	**Resumed...**	**Started or Restarted...**	**Reset...**
Native GMS Menus	Menu bar is replaced with the developer menu bar	Menu choices are available and functional	Menu bar resets to the value of gms-initial-menu-bar	Menu bar is replaced with the developer menu bar
Classic GMS Menus	Menu choices are unavailable	Menu choices are available and functional	Menu bar resets to the value of gms-initial-menu-bar	Menu bar is unavailable
NMS Menus	Menu bar is replaced with the developer menu bar	Menu choices are available and functional	Menu choices are available and functional	Menu bar is replaced with the developer menu bar

	When G2 is...			
	Paused...	**Resumed...**	**Started or Restarted...**	**Reset...**
Developer Menu Bar	Menu choices are available and functional	Menu choices are available and functional	Menu choices are available and functional	Menu choices are available and functional
Classic G2 Menus	Menu choices are available and functional	Menu choices are available and functional	Menu choices are available and functional	Menu choices are available and functional

In the table above:

- **Native GMS Menus** are menus created using GMS and displayed in Telewindows running with a standard user interface.
- **Classic GMS Menus** are menus created using GMS and displayed in Telewindows running with a classic user interface.
- **NMS Menus** are menus created using the NMS API, except for the menu bar you get when you use g2-nms-set-menu-bar with an argument of 0, which displays the developer menu bar.
- **Developer Menu Bar** is the menu bar you get by default in Telewindows when no other menus are defined. This is also the menu bar you get when you use g2-nms-set-menu-bar with an argument of 0.
- **Classic G2 Menus** are the G2 Main Menu, the KB Workspace menu, and item popup menus.

Demos

G2 provides a number of demos that illustrate how GMS menus render as native Windows menus, as well as how to use the NMS API to create and manipulate menus.

You can access these demos in this subdirectory of your G2 product installation directory:

```
\kbs\demos\
```

You can also load `\kbs\gmsdemo.kb` to test other features of GMS in Telewindows, such as dynamic and built-in menus.

gms-native-multiple-menubar-demo.kb

This demo shows various features of GMS menus rendered as native Windows menus. The Multiple Menubar Demo WS workspace provides the GMS menu

templates and settings for three alternate menu bars. The menu bars are implemented as GMS menu bar templates, and the submenus are implemented as GMS cascade templates. G2 must be running to interact with the menus.

To switch between menus bars, choose Logo > Other Menus and choose a menu. The menu bar switches to the alternate menu bar and a workspace with a blue title bar also appears. Now choose a different menu. The menu bar switches to the alternate menu, and another workspace appears. The title bar of the new workspace is blue, and the title bar of the previous workspace switches to grey. Switching between menu bars is implemented by using a GMS switch menu bar cascade template. Displaying the workspace and coloring its title bar is implemented by specifying a callback procedure in the gms-posting-callback of the GMS menu bar template, which activates when the menu bar is posted.

To change the user mode, choose Logo > User Mode and choose a user mode. Changing the user mode is implemented by using a GMS change mode template.

With the GDA menu active, choose the Modules menu and choose a module. The top-level workspace of the module appears, and a message is posted to the Message Board. Displaying a workspace is implemented by using a GMS show workspace template. Posting to the Message Board is implemented by specifying a callback procedure in the gms-activation-callback of the GMS menu bar template, which activates when any menu choice in the menu bar is chosen.

See Example: Alternate GMS Menu Bar.

gms-native-large-menu-demo.kb

This demo provides another GMS menu bar template, which renders as a native Windows menu bar. The Large Menu Demo WS provides the GMS menu template and settings. G2 must be running to interact with the menus.

The unique part of this demo is the Item menu, which creates instances of items. Choose Items, then choose a submenu until you display a submenu of items. Choosing an item from the last submenu attaches the item to the mouse. Click the workspace where you want to place the item.

The Item menu and its submenus are implemented as GMS cascade templates. The submenus are defined on the subworkspace of GMS subpanel templates. Creating instances of items is implemented by specifying a callback procedure in the gms-activation-callback of the Item cascade template.

See Example: Alternate GMS Menu Bar.

gms-native-popup-demo.kb

This demo shows how GMS popup menus render as native Windows popups when viewed through Telewindows. The Popup Demo WS provides the GMS menu templates and settings. G2 must be running to interact with the popups.

To display the popups:

- Right-click the test-umc object and choose Post Item Menu to display a popup, using a user menu choice.
- Right-click the test-track object to display a popup, using mouse tracking.
- Right-click the test-directed-item object to display a popup, then choose Rotate and either Clockwise or Counterclockwise.

The popup menu is implemented as a GMS popup template. The class definitions for each item and the callbacks are located on the Settings subworkspace. The callbacks for all the menu choices in the popup simply post messages to the Message Board. The callbacks for the menu choices in the Rotate submenu actually rotate the item.

See Example: GMS Popup Menu.

gms-native-language-demo.kb

This demo shows how localized GMS menus render as native Windows menus when viewed through Telewindows. The Language Demo WS provides the GMS menu templates and settings. G2 must be running to interact with the menus.

Click the English button to display the List menu in English. Click the Spanish button to display the List menu in Spanish. This demo also shows how to implement a callback that creates an item and attaches it to the mouse. First, create a new workspace, then choose an item from the List submenu and click on the workspace to create the item.

The Item menu is implemented as a GMS cascade template, whose label is a key, s-items, into a local text resource. Similarly, the List menu is a GMS cascade template, whose label is s-list. The menu choices in the List menu appear on the subworkspace of a GMS subpanel template and are GMS choice templates.

The menu bar is implemented as a GMS menu bar template, whose gms-text-resource-group is language-resources, a GMS text resource group, which appears on the Language Demo WS.

To view the local text resources for each menu, display the subworkspace of the language-resources text resource group. The subworkspace contains two local text resources, one for English and one for Spanish, each of whose gfr-resource-group is the language-resources resource.

To view the specification of each menu, first, get the GXL top-level workspace and enable the Array and List Editing option. Once this option is enabled, you can choose edit resource on each local text resource to view a GXL spreadsheet that specifies key-value pairs for each menu choice.

See Example: GMS Localization.

nmsdemo.kb

This demo provides examples of how to use the NMS API to create and manipulate native Windows menus. The NMSDemo workspace contains several G2 action buttons, which start procedures that call various NMS API procedures.

The Simple Menu Bar procedure shows how to create a menu bar, create a submenu, add menu choices with keys, set help text and color, add separators, and, finally, set the menu bar in the window.

The Simple Popup Menu procedure shows how to create a popup menu and add menu choices with keys, then set the popup menu in the window at a given x, y location relative to the item.

The demo callback procedure simply gets the specified key for the selected menu choice and uses it to update the text of a free text called menu-choice.

See Examples.

42

Windows Dialogs

Describes the complete specification for creating custom Windows dialogs, including a description of the various dialog controls.

Introduction

This chapter shows examples of Windows dialogs in Telewindows. These examples are available in `dialogs-demo.kb` located in `g2\kbs\demos` (Windows) or `g2/kbs/demos` (UNIX).

You must be running Telewindows on a Windows machine to post custom dialogs. Posting dialogs in G2 or in Telewindows running on UNIX generates an error. Custom Windows dialogs are only supported in Telewindows Next Generation (`twng.exe`). You can create these types of basic Windows dialogs:

- Basic
- Query
- Notification
- Delay notification
- File
- Print
- Custom

Custom dialogs can contain a variety of standard Windows controls, including text, list, color, time and date, progress bar, tabular, grouping, image, and workspace controls.

You specify the custom dialog as a structure, which describes the format and layout of the controls within the dialog. You post dialogs on a specific G2 window, which returns a unique dialog ID. When the user updates or dismisses a dialog, the G2 application receives callbacks. You can modify dialogs dynamically after they have been posted, for example, to enable and disable controls.

You specify basic and custom dialogs as part of your G2 application, using the dialog API, which is fully described in:

- Dialog Views in User Interface Operations in the *G2 System Procedures Reference Manual*.
- Custom Windows Dialogs.

Running the Dialogs Demo

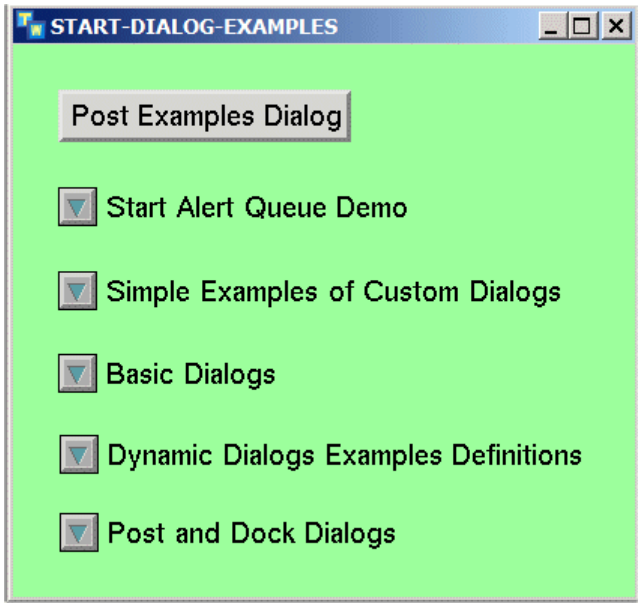
This section describes how to run the dialogs demo and shows examples of some of the basic and custom dialogs, as well as a sample alert queue.

To run the dialogs demo:

1 Load `dialogs-demo.kb` located in `g2\kbs\demos` (Windows) or `g2/kbs/demos` (UNIX).

2 Connect to the demo using Telewindows.

The start-dialog-examples workspace appears:

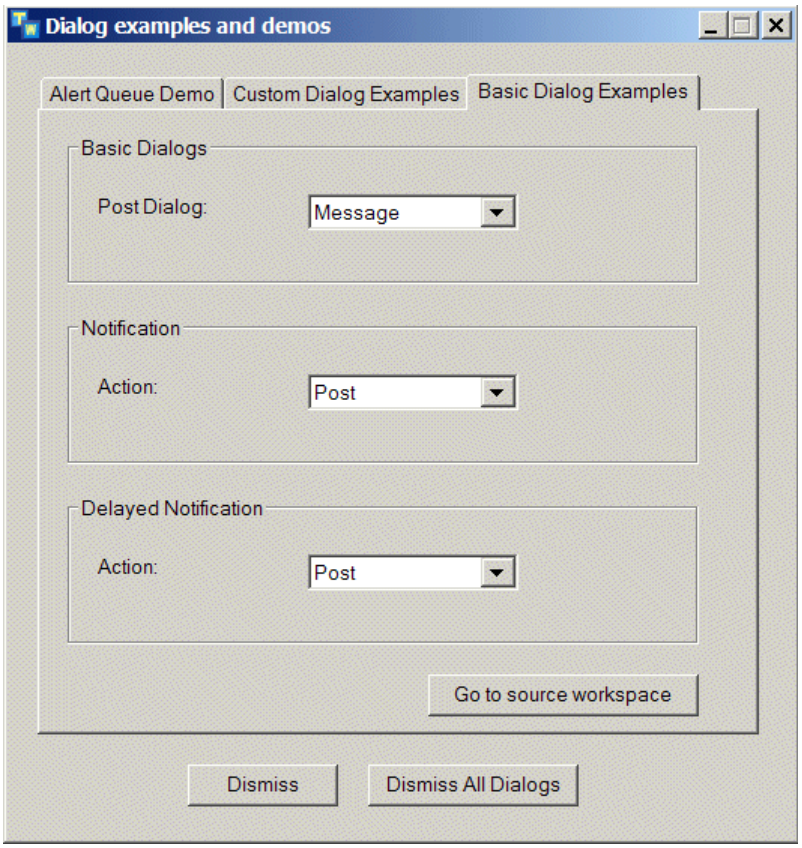


3 Click the Post Example Dialogs button.

A custom Windows dialog appears from which you can post and modify various types of dialogs.

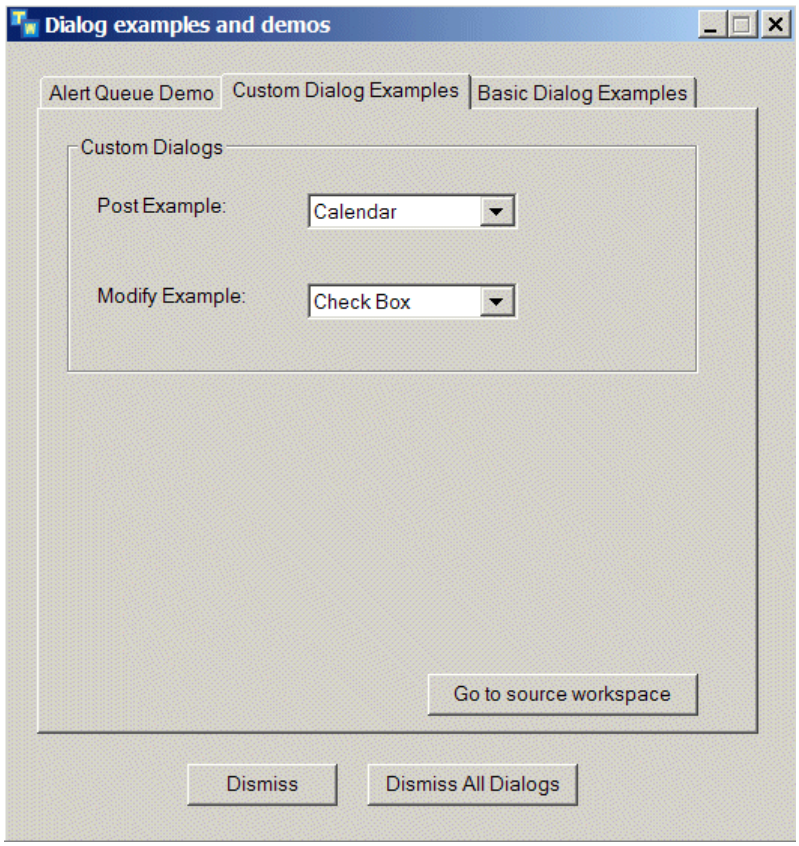
4 Click the Basic Dialog Examples tab.

From this tab, you can post basic, notification, and delay notification dialogs, as well as perform various actions on the notification and delay notification dialogs:



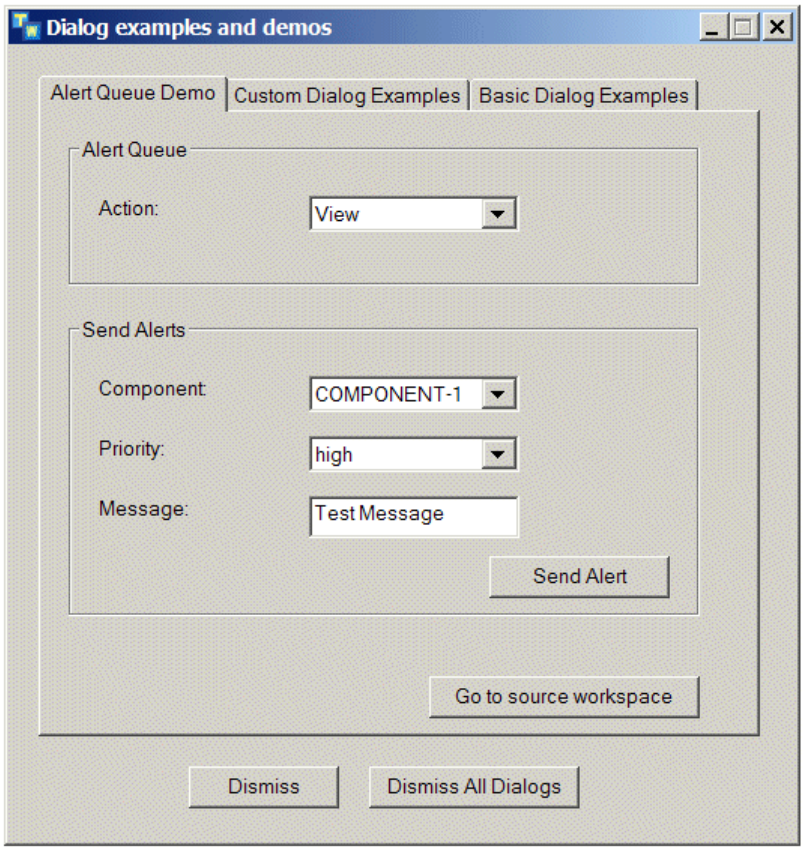
5 Click the Custom Dialog Examples tab.

From this dialog tab, you can launch custom dialogs with examples of each type of Windows control:



6 Click the Alert Queue Demo tab:

From this dialog tab, you can launch a sample alarm queue and send alarm messages to components:



7 To dismiss the current dialog, click Dismiss.

8 To dismiss all dialogs, including the current dialog, click the Dismiss All Dialogs button.

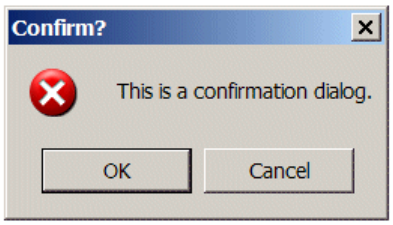
Posting Basic Dialogs

A basic dialog contains a message, a set of buttons for the specified type of dialog, an icon, and a caption. You can create message, confirmation, yes-no, yes-no-cancel, and retry-cancel dialogs. The procedure that posts the basic dialog returns the name of the button chosen.

To post a basic dialog:

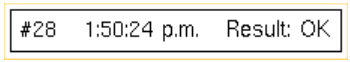
1 On the Basic Dialog Examples tab, choose Message, Confirmation, Yes/No, Yes/No/Cancel, Retry/Cancel from the Post Dialog dropdown list.

For example, here is a confirmation dialog, which consists of OK and Cancel buttons:



2 Click the OK button.

The example dialog posts the name of the clicked button to the Message Board:

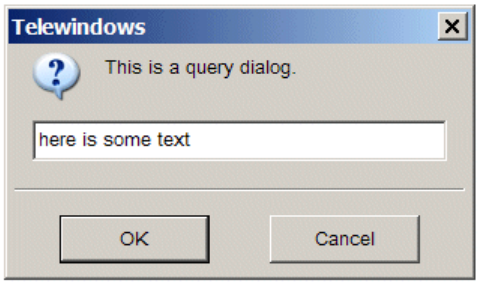


Posting Query Dialogs

A query dialog consists of a message, a text field for the user to enter a value, a caption, and an icon. The dialog provides OK and Cancel buttons. The procedure that posts the query dialog returns the entered text and the name of the button chosen.

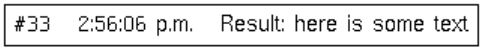
To post a query dialog:

1 On the Basic Dialog Examples tab, choose Query from the Post Dialog dropdown list and enter some text:



2 Click the OK button:

The example dialog posts the entered text to the Message Board:



Posting Notification Dialogs

A notification dialog consists of a message, a caption, and an icon. You can configure the font size of the message text. You can post, update, and remove notification dialogs. Typically, you post a notification dialog, wait for some event to occur, update the dialog, then remove the dialog when some other event occurs.

To post a notification dialog:

1 On the Basic Dialog Examples tab, choose Post from the Action dropdown list in the Notification group to post this dialog:

2 Choose Update from the Action dropdown list to update the dialog text:

3 Choose Remove from the Action dropdown list to dismiss the dialog.

Posting Delay Notification Dialogs

A delay notification dialog consists of a message, a caption, and an animation object. You can configure the font size of the message text. You can post, update, and remove notification dialogs. Typically, you post a notification dialog, wait for some event to occur, update the dialog, then remove the dialog when some other event occurs. You can also start and stop the animation, and step through the animation one frame at a time.

To post a notification dialog:

1 On the Basic Dialog Examples tab, choose Post from the Action dropdown list in the Delay Notification group to post this dialog, whose icon animates:

2 Choose Update from the Action dropdown list to update the dialog text:

3 Choose Step several times from the Action dropdown list to step through the animation one step at a time.

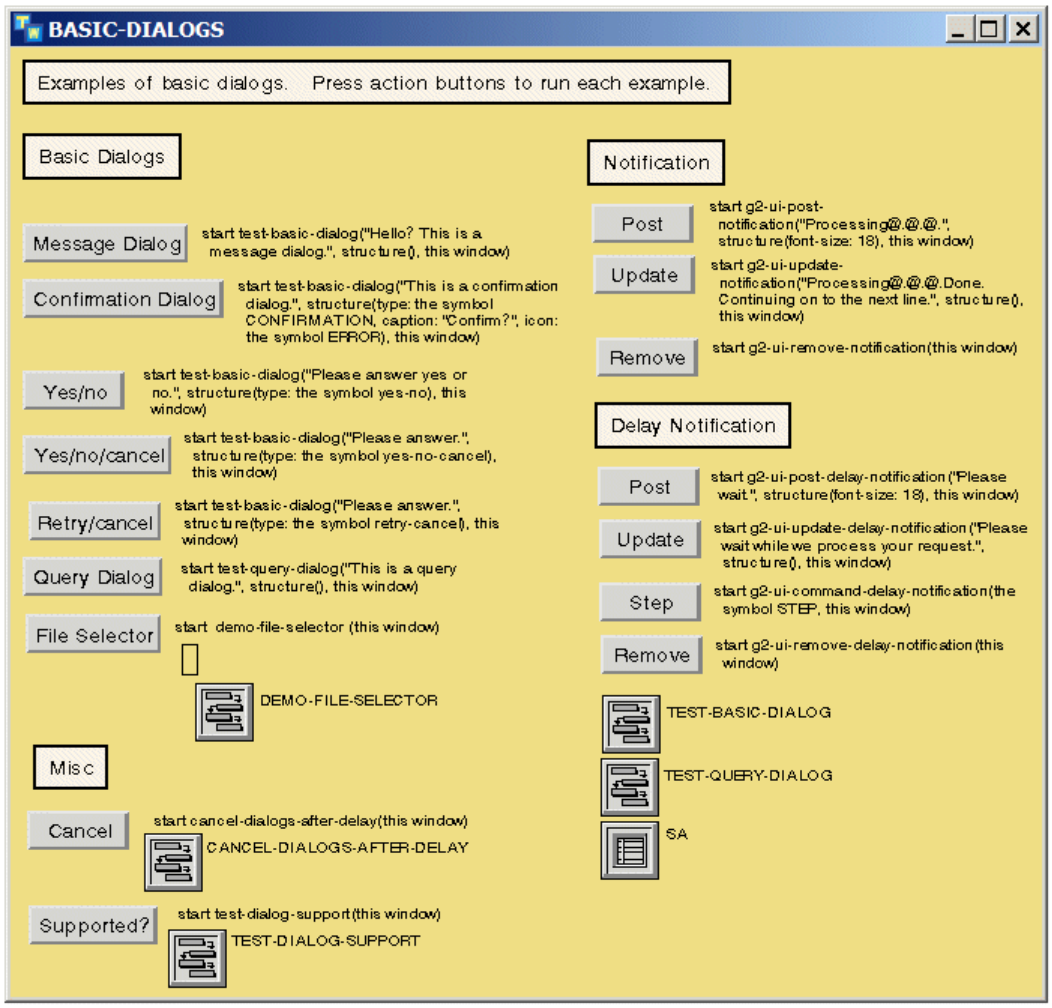
4 Choose Remove from the Action dropdown list to dismiss the dialog.

Viewing the Source Workspace for Basic Dialogs

To view the source workspace for basic dialogs:

➔ Click the Go To Source Workspace button on the Basic Dialogs Examples tab.

The workspace contains action buttons that call procedures that post basic, query, and notification dialogs and display the results to the message board:



Posting Custom Dialogs

A custom dialog consists of any number of these standard Windows controls: labels, text boxes, push buttons, radio buttons, spinners, check boxes, combo boxes, list boxes, groups, tab frames, and tabular views.

You specify the custom dialog as a structure, which describes the format and layout of the controls within the dialog. You post dialogs on a specific G2 window, which returns a unique dialog ID. When the user updates or dismisses a dialog, the G2 application receives callbacks, depending on the response action you specify for each control. For example, when the user clicks the OK button, it can trigger the dialog updated callback, and when the user clicks the Cancel button, it can trigger the dialog dismissed callback.

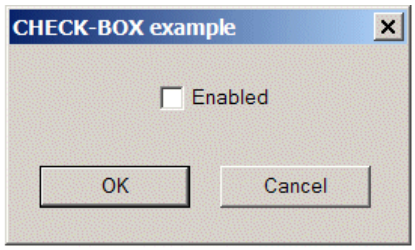
You can post and cancel custom dialogs. You can modify dialogs dynamically after they have been posted, for example, to enable and disable, show and hide, check and uncheck, and add elements to various types of controls. You can also query custom dialogs to obtain information back from the dialog.

To post a custom dialog:

1 Click the Custom Dialog Examples tab.

2 Choose a control from the Post Example dropdown list to display a dialog that contains examples of the control.

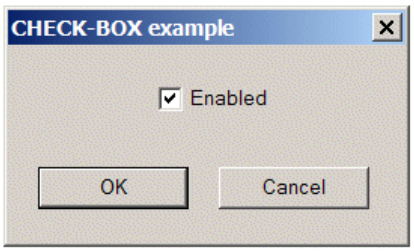
For example, here is the result of choosing Check Box from the dropdown list, which displays a custom dialog with a check box and two push buttons:



3 Check and uncheck the control manually.

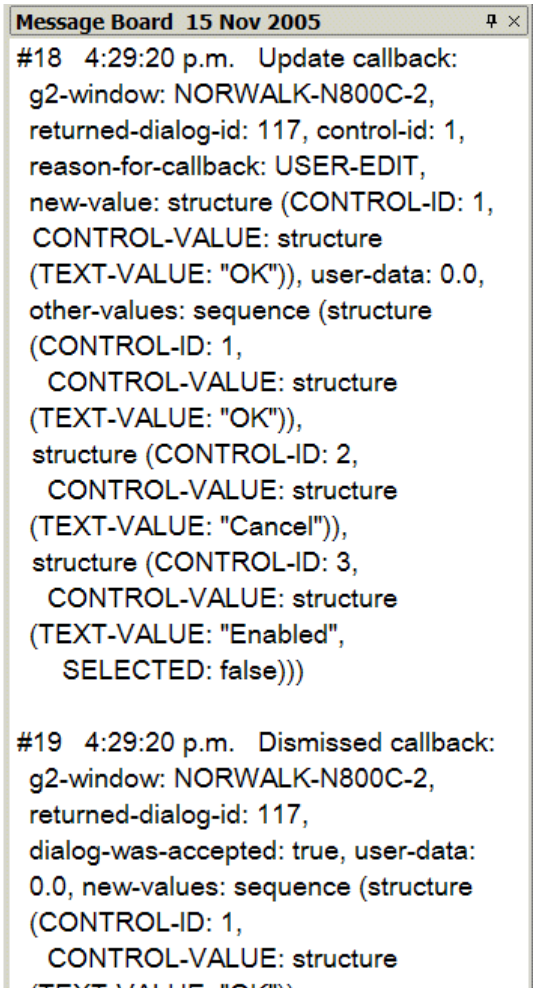
4 To modify the control programmatically, choose Check Box in the Modify Example dropdown list.

For example, here is the result of modifying the Check Box control, which enables the check box:



5 Click the OK button.

The custom dialog executes the dialog updated and dialog dismissed callback, which posts this information to the Message Board:

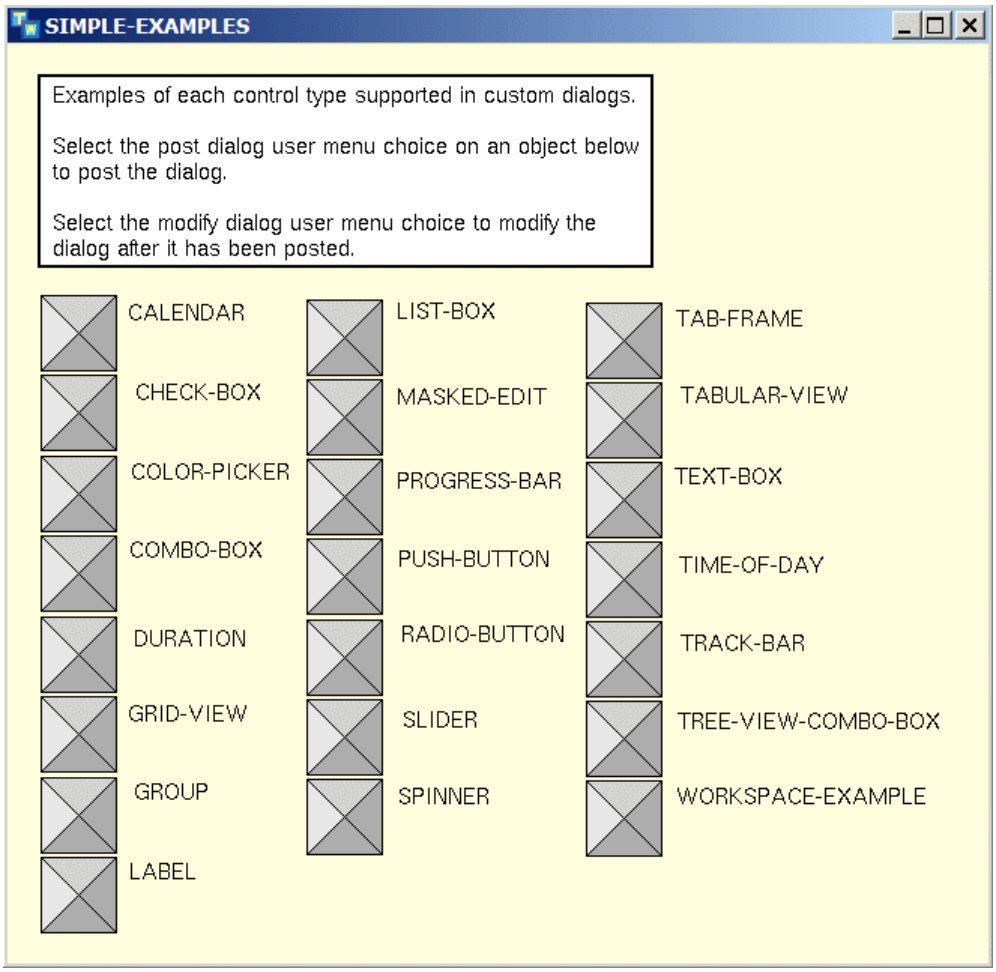


Viewing the Source Workspace for Custom Dialogs

To view the source workspace for custom dialogs:

1 Click the Go To Source Workspace button on the Custom Dialogs Examples tab.

The workspace contains example-dialog-class instances, which define the post dialog menu choice for posting the dialog:



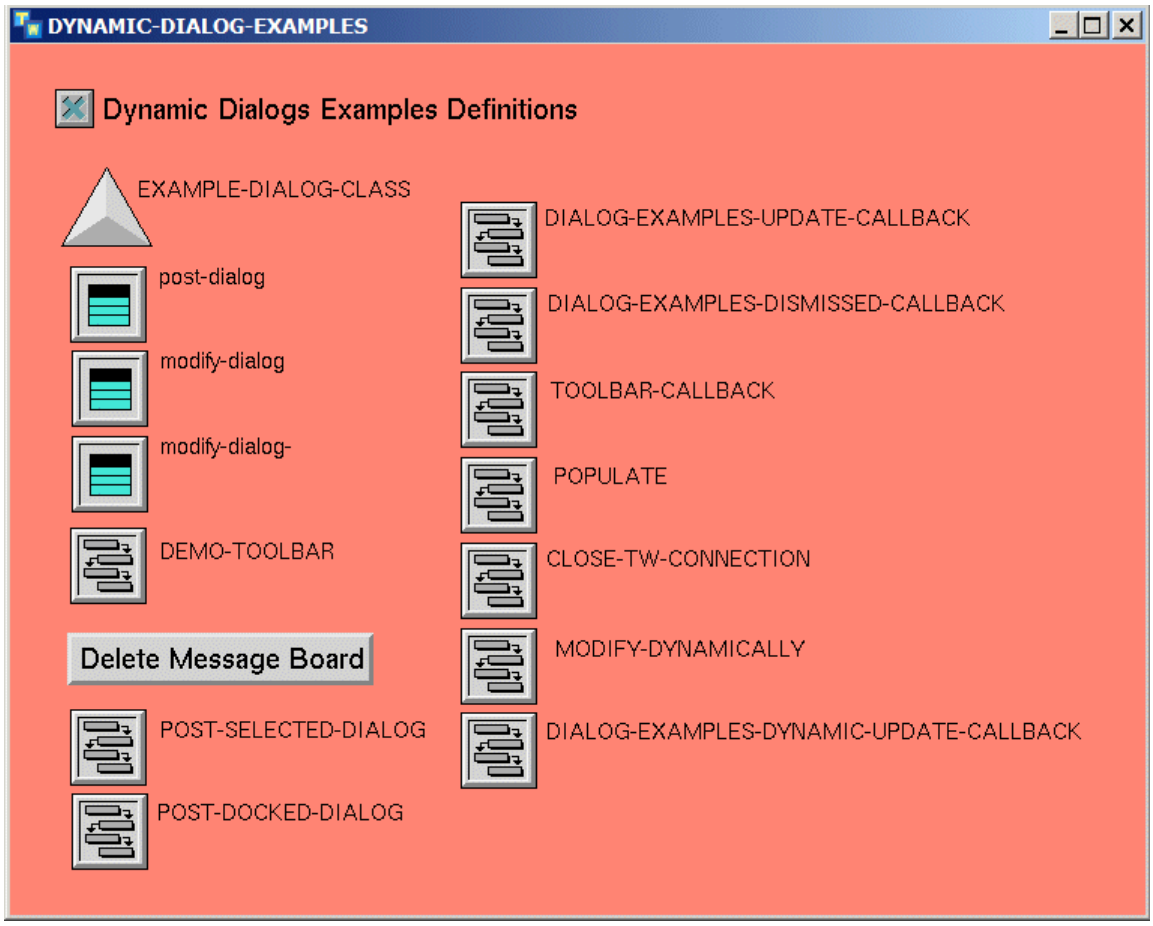
2 Choose table on one of the example-dialog-class instances to see the dialog and dialog update specifications.

Here is the table for the check-box example dialog, which specifies dialog-update and dialog-specification.

CHECK-BOX, an example-dialog-class

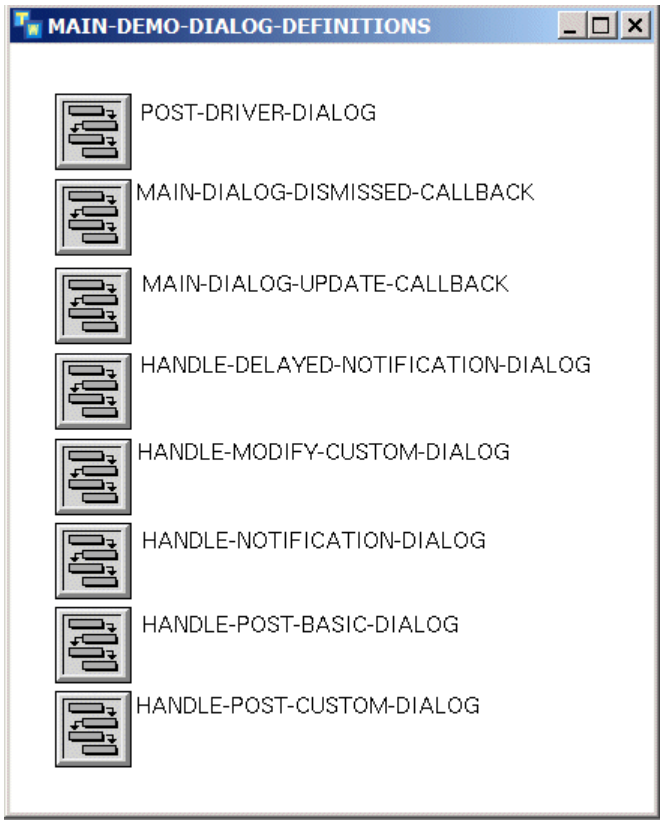
Notes	OK
Authors	msn (6 May 2005 3:26 p.m.)
Change log	0 entries
Names	CHECK-BOX
Dialog title	"CHECK-BOX example"
Dialog width	135
Dialog height	80
Dialog x position	left
Dialog y position	top
Dialog type	modeless
Update callback	dialog-examples-update-callback
Dismissed callback	dialog-examples-dismissed-callback
Returned dialog id	117
Dialog user data	0.0
Dynamic modify	false
Dialog update	sequence (structure (control-action: the symbol check, control-id: 3))
Dialog components	sequence (structure (control-type: the symbol push-button, control-id: 1, height: 14, width: 50, left: 10, top: 40, response-action: the symbol ok, control-value: structure (text-value: "OK")), structure (control-type: the symbol push-button, control-id: 2, height: 14, width: 50, left: 70, top: 40, response-action: the symbol cancel, control-value: structure (text-value: "Cancel")), structure (control-type: the symbol check-box, control-id: 3, height: 15, width: 100, top: 10,

3 Go to the dynamic-dialogs-examples workspace to see the definition of the example-dialog-class and its associated procedures and user menu choices:



The post-selected-dialog procedure is called when you choose post dialog, and the dialog-examples-update-callback and dialog-examples-dismissed-callback are called when the dialog is updated and dismissed, respectively.

4 Go to the main-demo-dialog-definitions workspace to see how these procedures get called when dialogs are posted and updated:



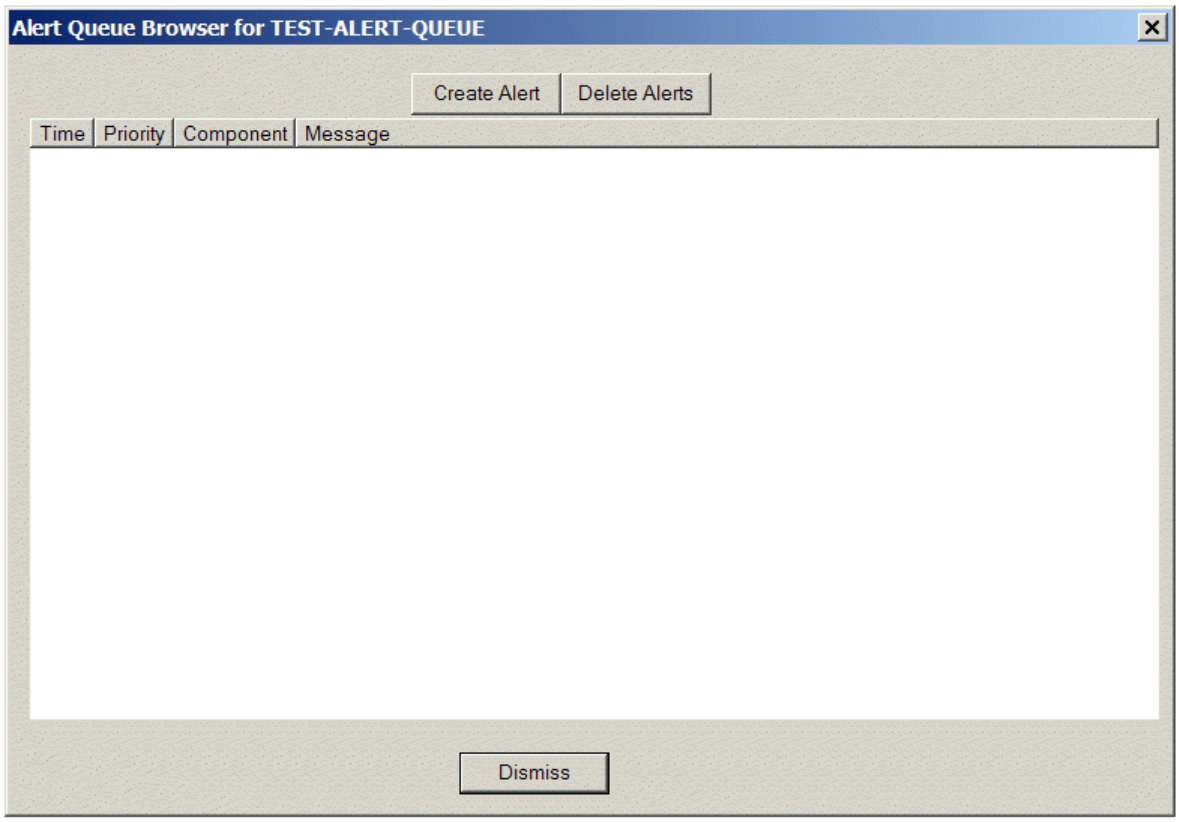
Posting Messages to an Alert Queue

You can create an alert queue and post messages to the queue, by creating a custom dialog that uses the tabular view control. The dialogs demo KB has a complete example of creating an alert queue, including creating and deleting alerts, and clearing the queue.

To display the example alert queue:

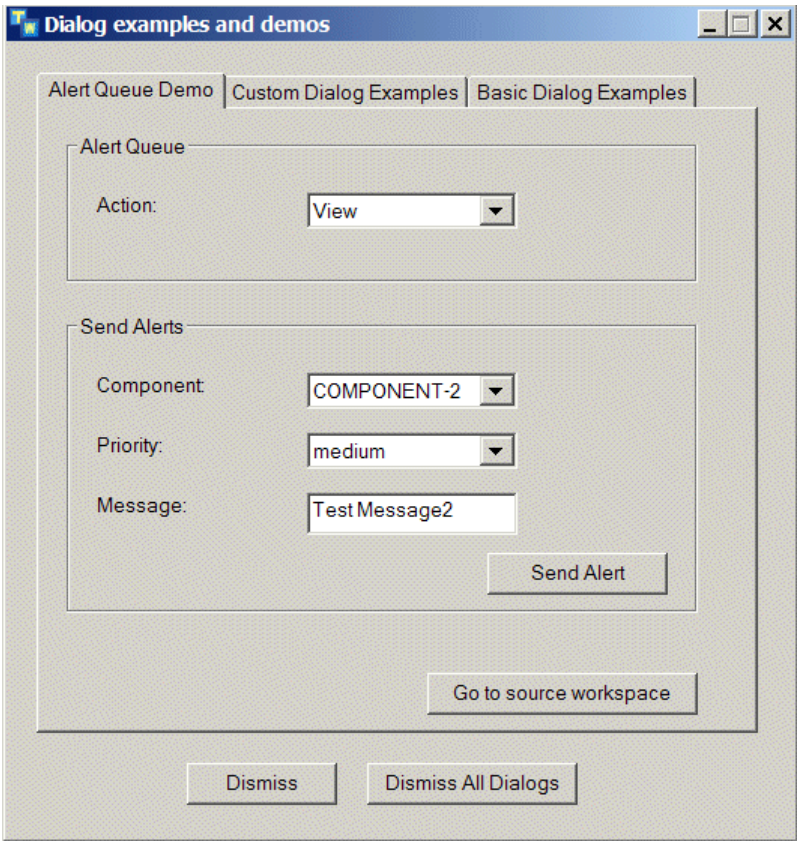
1. Click the Alert Queue Demo tab.
2. Choose View from the Alert Queue Action dropdown list.

   An empty alert queue appears:

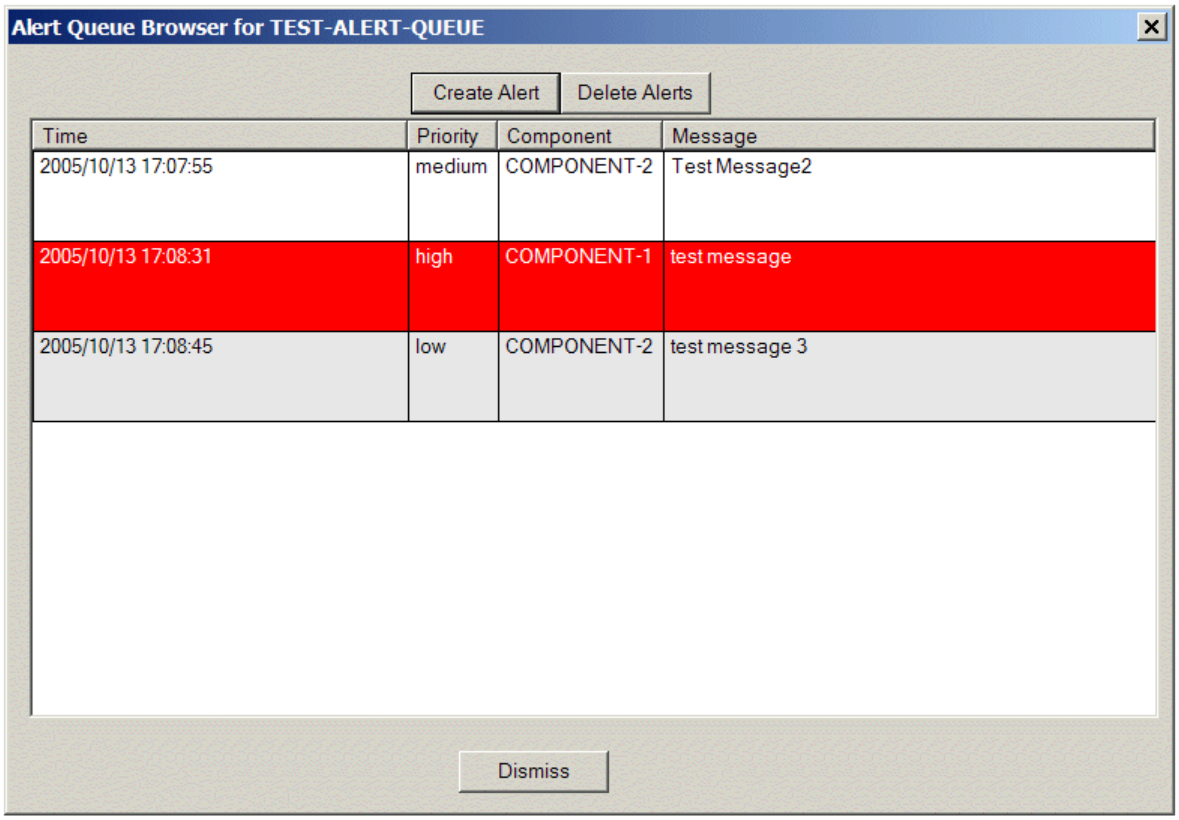


3 Configure the Component, Priority, and Message and click the Send Alert button to create several messages.

For example, here is the dialog configured to create a high priority alert message on component-1 with message text Test Message 2:



Here is the resulting message queue with three messages added:



4 To delete all messages, click the Delete Alerts button in the queue or choose Clear from the Alert Queue Action dropdown list.

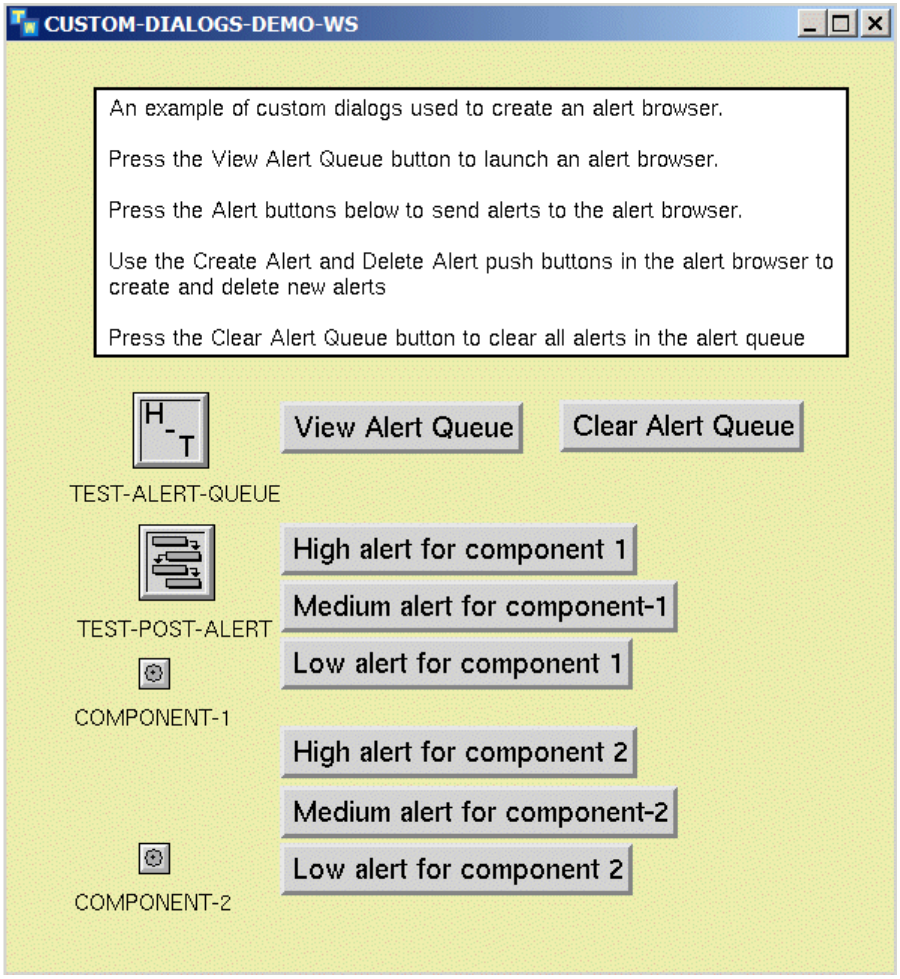
Viewing the Source Workspace for the Alert Queue

Notice that for a tabular view, the *new-value* argument to the dialog update callback returns only the selected-rows of the control-value structure, not the columns and rows. The reason is that the data cannot be changed by the user, and there may be many rows of data. Additionally, the architecture is such that the model and view are kept separate, and a separate controller is responsible for handling callbacks from the view and updating the model. To modify the view, you use the g2-modify-custom-dialog system procedure.

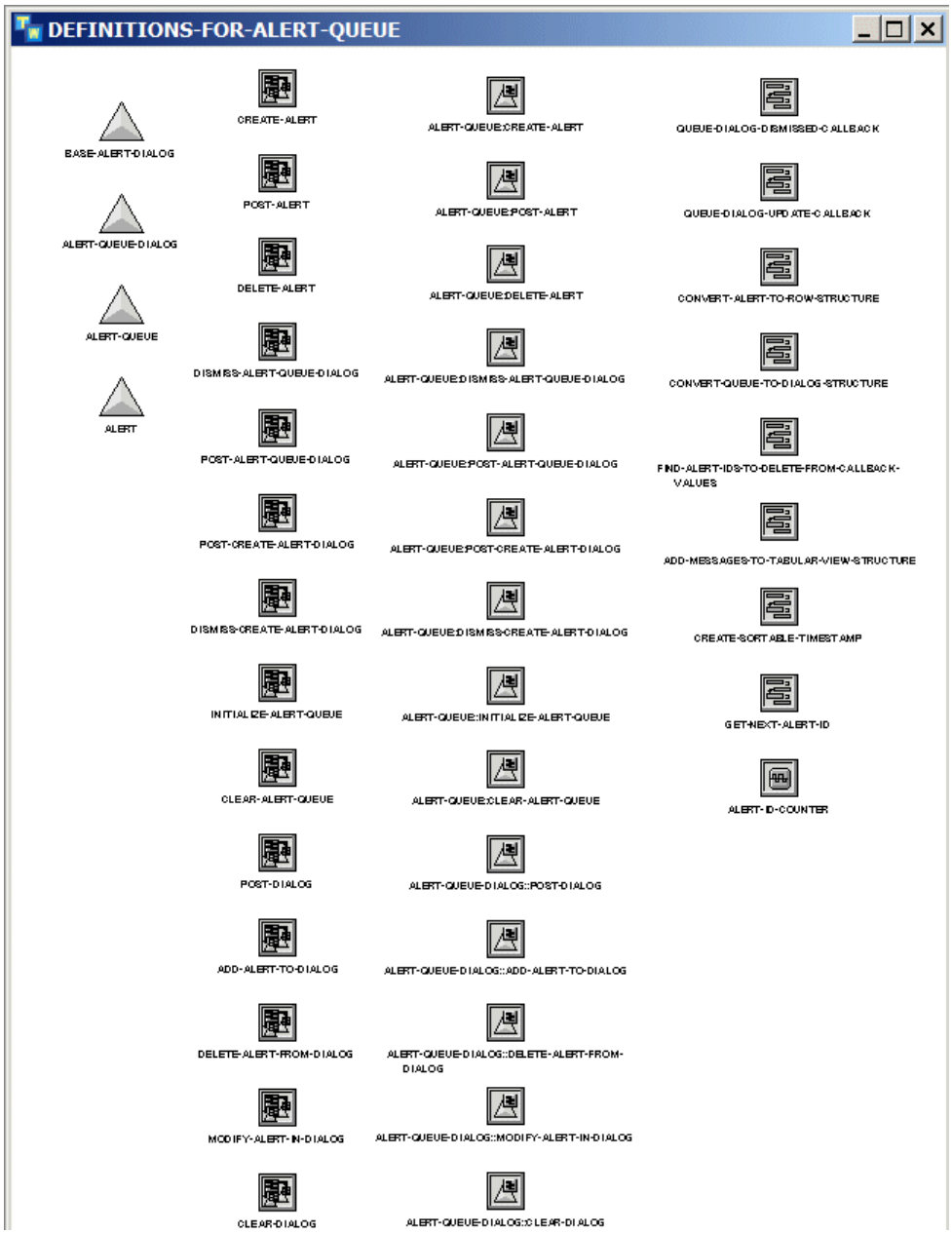
To view the source workspace for the alert queue:

1 Click the Go To Source Workspace button on the Alert Queue Demo tab:

The workspace contains an alert-queue instance, a procedure that posts alerts, and various buttons for viewing and clearing the alert queue, and creating various types of alerts:



2 Go to the definitions-for-alert-queue workspace to see the definitions for the alert-queue and alert classes, and their associated methods:



43

Custom Windows Dialogs

Describes the complete specification for creating custom Windows dialogs,

Introduction

This chapter describes the API for creating custom Windows dialogs in Telewindows.

Note You must be running Telewindows on a Windows machine to post custom dialogs. Posting dialogs in G2 or in Telewindows running on UNIX generates an error. Custom Windows dialogs are only supported in Telewindows Next Generation (`twng.exe`). See Dialog Specification.

You specify the custom dialog as a structure, which describes the format and layout of the controls within the dialog. You post dialogs on a specific G2 window, which returns a unique dialog ID. When the user updates or dismisses a dialog, the G2 application receives callbacks. You can modify dialogs dynamically after they have been posted, for example, to enable and disable controls.

You can create dialogs with these Windows controls in these categories:

- Text controls
  - label
  - text-box
  - masked-edit
- Button controls
  - check-box
  - push-button
  - radio-button
  - toggle-button

- List controls
  - combo-box
  - list-box
  - checkable-list-box
  - tree-view-combo-box
- Color controls
  - color-picker
  - full-color-picker
- Time and date controls
  - calendar
  - duration
  - time-of-day
- Numeric input controls
  - spinner
  - slider
  - track-bar
- Tabular controls
  - grid-view
  - tabular-view
- Grouping controls
  - group
  - tab-frame
- Miscellaneous controls
  - progress-bar
  - image
  - workspace

You specify the dimensions of dialogs and controls in **dialog units**, which is a device-independent measure to use for layout. One horizontal dialog unit is equal to one-fourth of the average character width for the current system font. One vertical dialog unit is equal to one-eighth of an average character height for the current system font. For more information, see Microsoft documentation.

Custom dialogs obey the Windows desktop font preferences, which you specify in the Display Properties dialog on the Appearance tab. In particular, dialogs use the Message Box font, which you specify by clicking the Advanced button.

For examples of creating custom dialogs, see Windows Dialogs.

For information on creating basic Windows notification, query, file, and print dialogs, see Dialog Views in User Interface Operations in the *G2 System Procedures Reference Manual*.

For examples of basic and custom Windows dialogs, see `dialogs-demo.kb` located in `g2\kbs\demos` (Windows) or `g2/kbs/demos` (UNIX). Connect to the demo using Telewindows and click the Post Examples dialog on the Dialogs Examples Home workspace to display a custom dialog from which you can run all the examples.

For information on converting GUIDE dialogs to custom Windows dialogs and system procedures for adding dialog controls to custom Windows dialogs, see the *G2 Dialog Utility User's Guide*.

Posting a Custom Dialog

To post a dialog on a remote client's window, call the following system procedure:

g2-ui-post-custom-dialog
(*dialog-specification*: structure, *user-data*: value, *win*: g2-window)
-> *dialog-handle*: integer

The procedure returns an integer that is a handle to the posted dialog. This procedure signals an error if the specified window does not support standard Windows.

Argument	Description
dialog-specification	A structure that describes the dialog. For a description of this argument, see Dialog Specification.
user-data	An item or value passed to the callback when it is invoked. For example, you can use the same callback for multiple registrations and specify different *user-data* to differentiate each callback.
win	The g2-window on which to post the dialog. This window must be running on a Windows machine.

Return Value	Description
dialog-handle	An integer that provides a handle to the custom dialog that gets created.

Dialog Specification

The *dialog-specification* argument to the g2-ui-post-custom-dialog system procedure has these attributes:

Attribute	Type	Required	Default	Description
components	sequence	yes	N/A	A sequence of dialog component structures. Dialog Component Structure.
container	symbol or integer	no	mdi-child	One of the following: • pane — Displays the dialog in a docked pane. • mdi-child — Displays the dialog in a floating pane. • The handle of a listbar-style shortcut bar, as an integer, in which case the neighbor option is the number of the folder within the listbar into which to create the dialog.
dialog-dismissed-callback	symbol or procedure class	no	none	The procedure to call when the dialog is dismissed. See Dialog Dismissed Callback.
dialog-height	integer	yes	N/A	The height of the dialog in dialog units.

Attribute	Type	Required	Default	Description
dialog-is-mdi-child	truth-value	no	false	Whether to create the dialog as a Multiple Document Interface (MDI) child, which means the dialog has a minimize button. MDI child dialogs are modeless (not modal).
dialog-is-modal	truth-value	no	true	Whether the dialog is modal, which means the user cannot interact with any other objects while the dialog is open.
dialog-title	text	no	"G2"	The title of the dialog window.
dialog-update-callback	symbol or procedure class	no	none	The procedure to call when the dialog is updated. See Dialog Update Callback
dialog-width	integer	yes	N/A	The width of the dialog in dialog units. Actual width may vary depending on the display device.

Attribute	Type	Required	Default	Description
dialog-x-position	integer or symbol	no	center	The x position of the dialog in the specified window. You can specify any of these symbols: • left, right, center • working-area-center • working-area-{left|top|right|bottom} • near-working-area-{left|top|right|bottom} • desktop-area-center • desktop-area-{left|top|right|bottom} • near-desktop-area-{left|top|right|bottom} The working area is the desktop area of the display, excluding taskbars, docked windows, and docked toolbars. The desktop area is the entire Windows desktop. You can also specify the x position in the G2 coordinate system, as an integer.
dialog-y-position	integer or symbol	no	center	The y position of the dialog in the specified window. See dialog-x-position for the options.

Attribute	Type	Required	Default	Description
dock	symbol	no	none	Where to dock the dialog in the window. The options are one of these symbols: none, left, top, right, bottom, float, or within. **Note:** This option is only relevant if dialog-is-modal and dialog-is-mdi-child are both false. If so, and if dock is any value other than none, then the dialog appears in a dockable pane, initially docked to the given side of neighbor, or within neighbor as a new tab, or floating.
icon	item or symbol	no	none	A G2 class name, an item, or a built-in GMS icon. For a list of built-in GMS icons, see image.
left	integer	no	N/A	The initial position of the left side of the dialog, in pixels. By default, the dialog is centered in the overall window.

Attribute	Type	Required	Default	Description
neighbor	integer	no	0	The integer handle of another pane to dock against. The neighbor can be a handle to a dialog, tree view, or shortcut bar pane. The default is 0, which means the overall window. To specify that the dialog be placed in a tab pane within another pane, specify these options: dock: the symbol within and neighbor *h*, where *h* is the pane within which to place the tree view pane. **Note:** This option is only relevant if dialog-is-modal and dialog-is-mdi-child are both false. If so, and if dock is any value other than none, then the dialog appears in a dockable pane, initially docked to the given side of neighbor, or within neighbor as a new tab, or floating.
top	integer	no	N/A	The initial position of the top of the dialog, in pixels. By default, the dialog is centered in the overall window.

Dialog Component Structure

The dialog component sequence attribute of the *dialog-specification* argument specifies a set of component structures contained within the dialog. Each structure specifies an individual element within a dialog. Each dialog component is associated with a separate window.

Component Structure Attributes

Each dialog component structure has these attributes for specifying the component within the dialog:

Attribute	Type	Required	Default	Description
anchor	symbol | sequence | integer	no	none	How the component should be moved or resized as the dialog is resized. The options are one of the symbols none, top, left, bottom, right, top-left, bottom-right, top-left-bottom-right, a sequence of these symbols, or an integer bitmask. The default value is top-left.
control-background-color	text	no	N/A	A string representing the background color for the text in the control.
control-id	integer or symbol	yes	N/A	A unique ID for the control. Within a single dialog, each control must have a unique ID.
control-type	symbol	yes	N/A	The type of control. See Control Types.
control-value	structure	yes	N/A	A structure that describes the value of the control. The structure attributes depend on the control type. See Control Types.
height	integer	yes	N/A	The height of the control in number of dialog units.

Attribute	Type	Required	Default	Description
horizontal-scrollbar	truth-value	no	false	Whether the component has a horizontal scrollbar.
is-disabled	truth-value	no	false	Whether the initial state of the control is disabled.
is-enabled	truth-value	no	true	Whether the initial state of the control is enabled.
is-tabstop	truth-value	no	Depends on the control type.	Whether the control is a tab stop, which means it gets selected when the user presses the Tab key in the dialog. By default, all control types except label are tab stops.
is-visible	truth-value	no	true	Whether the component is visible.
left	integer	yes	N/A	The number of dialog units from the top-left corner of the control to the left side of the parent window.
parent-control-id	integer or symbol or false	no	false	The unique ID of the parent control. A value of false indicates that the parent control is the top-level window of the dialog.
resizable	truth-value	no	false	Whether the dialog is resizable. The anchor attribute determines how the dialog behaves when it is resized.
response-action	symbol	no	Depends on the control type	The callback that the control triggers when the user updates the control manually. See Response Actions.

Attribute	Type	Required	Default	Description
starts-new-group	truth-value	no	false	Whether the control starts a new group of controls that are contained within a single group. This attribute is applicable to all controls, however, you can use it most effectively with radio button and check box controls. See Example: Creating Groups of Controls.
tabstop-index	integer	no	N/A	The order of the control when pressing the Tab key to navigate to the next control in the dialog. The smaller the index, the higher the priority. Controls with indexes are always given priority over those without indexes.
text-font	text	no	N/A	A string representing the font, as understood by the host system of Telewindows client. e.g. Times New Roman, Tahoma.
text-font-size	integer	no	N/A	A number representing the size of the text in the window.
text-font-color	text	no	N/A	A string representing the g2 color symbol, it will be the color of the text.
top	integer	yes	N/A	The number of dialog units up from the top-left corner of the control to the top of the parent window.
vertical-scrollbar	truth-value	no	false	Whether the component has a vertical scrollbar.

Attribute	Type	Required	Default	Description
width	integer	yes	N/A	The width of the control in number of dialog units.
win32-styles	sequence	no	sequence()	A sequence of win32-specific style symbols. See Windows-Specific Control Styles.

Note When the focus moves from one component to another in a dialog, either by pressing the Tab key or by clicking the control, components can become obscured if they overlap. Therefore, ensure that the heights of each dialog component do not overlap.

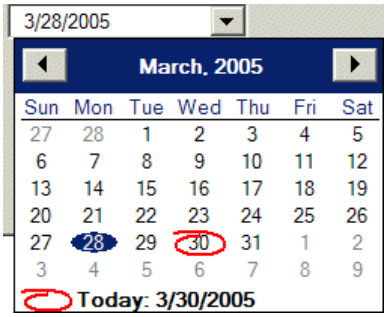
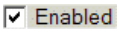
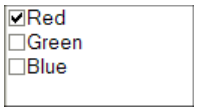
Note The height of all controls except the combo-box is specified in dialog units, where one dialog unit is 1/8 of the font height. Thus, for example, to display *n* rows in a list-box, specify *n**8 as the height. The height of a combo-box specifies the number of visible rows.

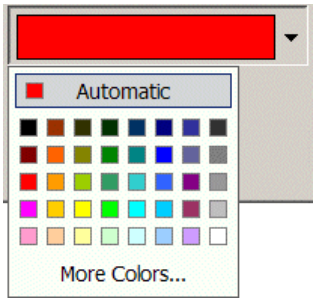
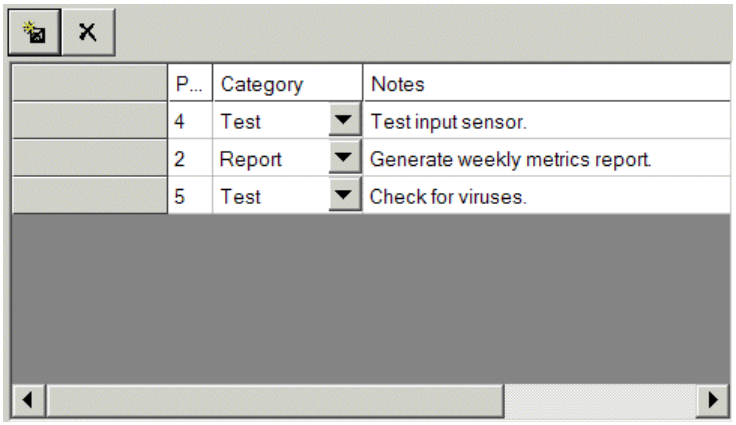
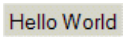
Note The text-font, text-font-size, text-font-color and control-background-color are newly added in May 2015 release, they're experimental support for the new Font feature request.

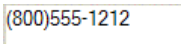
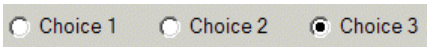
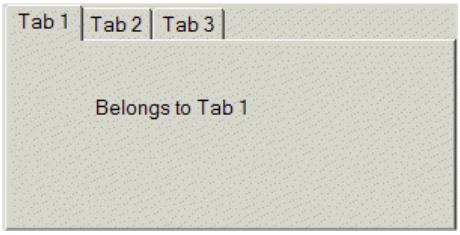
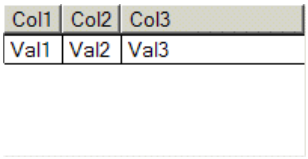
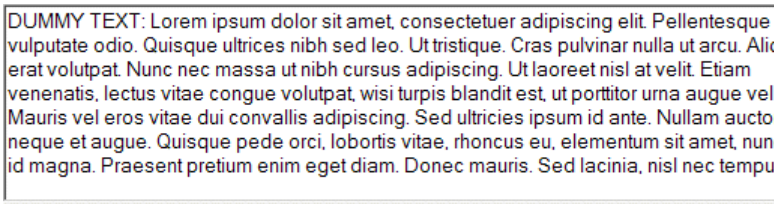
Note For details on controlling how the dialog behaves when it is resized, using the anchor attribute, visit `http://msdn.microsoft.com/library/en-us/cpref/html/frlrfsystemwindowsformsanchorstylesclasstopic.asp`.

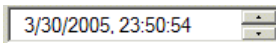
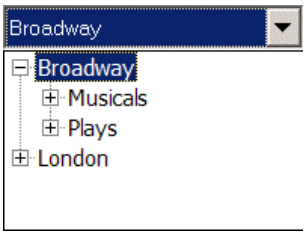
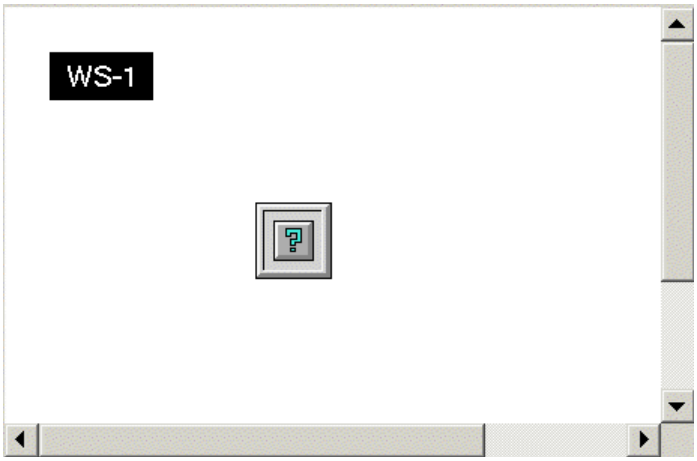
Control Types

The control-type attribute of the dialog component structure specifies the type of control to display in the dialog. The control-type attribute must be one of the following G2 symbols:

Control Type	Example
calendar	
check-box	
checkable-list-box	
color-picker	
combo-box	
duration	

Control Type	Example
full-color-picker	
grid-view	
group	
image	
label	
list-box	

Control Type	Example
masked-edit	
progress-bar	
push-button	
radio-button	
slider	
spinner	
tab-frame	
tabular-view	
text-box	

Control Type	Example
time-of-day	
toggle-button	
track-bar	
tree-view-combo-box	
workspace	

Windows-Specific Control Styles

The win32-styles attribute in the dialog component structure provides an alternate way of specifying the control type. You specify a sequence of symbols, which correspond with style bit-flags in the Win32 API.

The following table categorizes the supported controls into groups according to whether they can share the same style symbols or not:

Group	Controls	Win32 Control Styles
Static controls	label, group, tab-frame	WIN32 Static Control Style Symbols.
Edit controls	text-box	WIN32 Edit Style Symbols.
Button controls	push-button, check-box, list-box, toggle-button	WIN32 Button Style Symbols.
Combo controls	combo-box, group	WIN32 Combo-Box Style Symbols.
Spinners	spinner	WIN32 Spinner Style Symbols.
Tables	tabular-view	WIN32 Tabular-View Style Symbols.

Example: Posting a Simple Dialog

This example posts a simple dialog that contains a label, and OK and Cancel buttons. The procedure takes an instance of a my-custom-dialog-definition, which is a subclass of the built-in dialog-definition class. It calls g2-ui-post-custom-dialog to post the custom dialog, which returns a handle that is the dialog-id of the custom dialog.

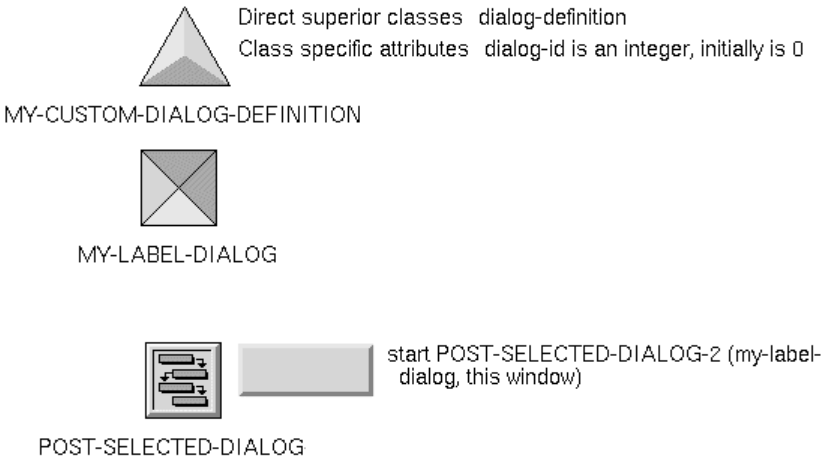
The first argument to g2-ui-post-custom-dialog is the dialog specification, which specifies the dialog title and size, whether the dialog is modal and an MDI child of the parent window, the dialog dismissed and update callbacks, and the dialog components.

The dialog components structure attribute specifies a sequence of structures, where each structure specifies a dialog control, in this case two push-button controls and a label control. Each control structure specifies the control type, control ID, size, position, response action, and control value.

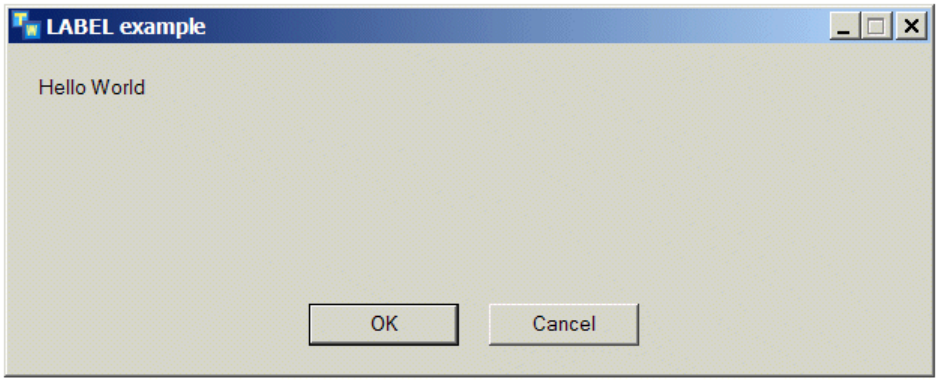
The second argument to the g2-ui-post-custom-dialog procedure is the user data, in this case a symbol, and the last argument is the window on which to post the dialog.

```
post-selected-dialog (dialog: class my-custom-dialog-definition,
     window: class g2-window)
dialog-id: integer;
begin
     dialog-id =  call g2-ui-post-custom-dialog
          (structure
               (dialog-title: "LABEL example",
               dialog-width: 310,
               dialog-height: 124,
               dialog-is-modal: false,
               dialog-is-mdi-child: true,
               dialog-dismissed-callback: dialog-examples-dismissed-callback,
               dialog-update-callback: dialog-examples-update-callback,
               components:
                    sequence
                         (structure
                              (control-type: the symbol push-button,
                              control-id: 1,
                              height: 14,
                              width: 50,
                              left: 100,
                              top: 86,
                              response-action: the symbol ok,
                              control-value: structure (text-value: "OK")),
                         structure (control-type: the symbol push-button,
                              control-id: 2,
                              height: 14,
                              width: 50,
                              left: 160,
                              top: 86,
                              response-action: the symbol cancel,
                              control-value: structure (text-value: "Cancel")),
                         structure (control-type: the symbol label,
                              control-id: 3,
                              width: 50,
                              height: 15,
                              left: 10,
                              top: 10,
                              response-action: the symbol ignore,
                              control-value: structure (text-value: "Hello World")))),
          the symbol hello-world-dialog, window);
     conclude that the dialog-id of dialog = dialog-id
end
```

Here is the class definition for the custom dialog, the dialog definition instance, and the procedure and action button that posts the custom dialog:



Here is the resulting dialog, which consists of a label and two push buttons:



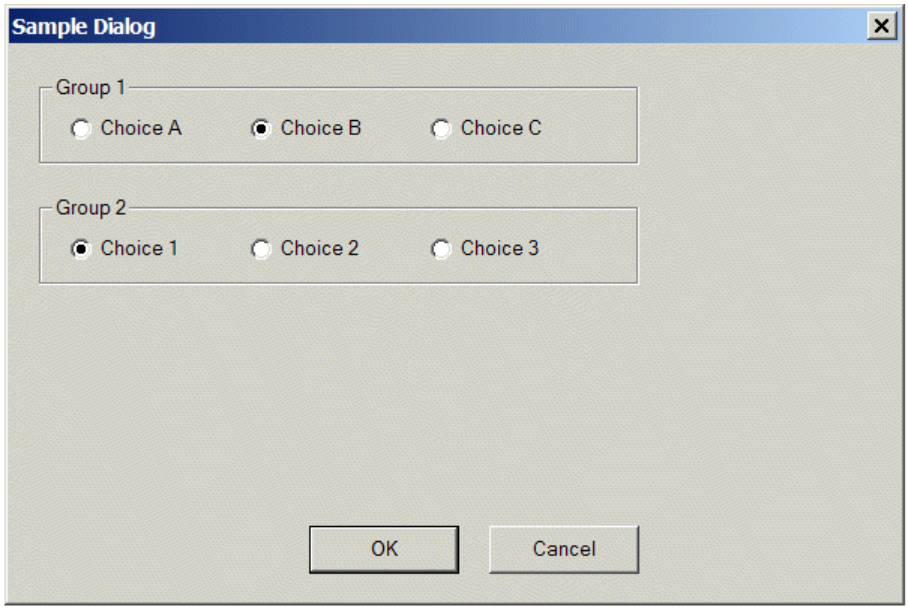
Example: Creating Groups of Controls

To create two groups of radio buttons with three buttons in each group, you would specify six radio-button controls, where the first and fourth buttons in the sequence of structures of dialog components would specify starts-new-group as true. The dialog components would also specify two group controls, where the first three radio buttons would be part of the first group and the second three radio buttons would be part of the second group. By specifying the first three buttons to be part of the first group, and the second three buttons to be part of the second group, this means you can select one radio button in each group.

Compare this technique of creating groups of controls with simply including two group controls in a dialog where three radio buttons appear in each group. This

technique simply displays a box around each set of radio buttons and has no effect on which buttons can be selected within each group.

For example, this figure shows a dialog with two groups of radio buttons in which you can select one radio button in each group:



Here is the specification for the radio buttons and the group controls:

```
structure
    (control-type: the symbol radio-button,
    control-id: the symbol radio-button-1-1,
    control-value: structure (text-value: "Choice A"),
    starts-new-group: true,
    . . .),
structure
    (control-type: the symbol radio-button,
    control-id: the symbol radio-button-1-2,
    control-value: structure (text-value: "Choice B", selected: true),
    . . .),
structure
    (control-type: the symbol radio-button,
    control-id: the symbol radio-button-1-3,
    control-value: structure (text-value: "Choice C"),
    . . .),
structure
    (control-type: the symbol radio-button,
    control-id: the symbol radio-button-2-1,
    control-value: structure (text-value: "Choice 1", selected: true),
    starts-new-group: true,
    . . .),
```

```
structure
    (control-type: the symbol radio-button,
    control-id: the symbol radio-button-2-2,
    control-value: structure (text-value: "Choice 2"),
    . . .),
structure
    (control-type: the symbol radio-button,
    control-id: the symbol radio-button-2-3,
    control-value: structure (text-value: "Choice 3"),
    . . .),
structure
    (control-type: the symbol group,
    control-id: the symbol group-1,
    control-value: structure (text-value: "Group 1"),
    . . .),
structure
    (control-type: the symbol group,
    control-id: the symbol group-2,
    control-value: structure (text-value: "Group 2"),
    . . .)
```

Dialog Callbacks

You can configure custom dialogs to send callbacks to G2 when the user updates controls within the dialog and/or when the user dismisses the dialog, that is, when the user clicks OK, Apply, or Cancel. You can use specific callbacks for dialog update and dismissed events, or you can use a generic dialog callback, which handles dialog update and dismissed events, as well as selection events for the grid-view control, and mouse click and key pressed events for the grid-view and tabular-view controls.

Response Actions

The response-action attribute in the dialog component structure determines which callbacks are triggered by the control when the user updates the control manually.

Response Action Symbol	Description
ignore	Do not send any callbacks.
ok, accept	Dismiss the dialog and respond with all data.

Response Action Symbol	Description
cancel	Immediately cancel the dialog and send a dialog dismissed callback. This response action is for use with the push-button control only.
respond	Send a dialog updated callback.
respond-with-all-data	Send a dialog updated callback, including all the data of all the controls in the dialog.
ok-with-just-my-data	Dismiss the dialog and respond only with the data for this control.
ok-without-data	Dismiss the dialog and do not respond with any data.

Note that whenever the user dismisses a dialog by clicking the Windows close box, the dialog dismissed callback is triggered with *dialog-was-accepted* of false. See Dialog Dismissed Callback.

Dialog Update Callback

The dialog-update-callback attribute in the dialog component structure determines the procedure to call when the user edits the value of a control in the dialog.

Here is the syntax for the dialog update callback:

my-dialog-update-callback
(*win*: class g2-window, *dialog-id*: integer, *control-id*: value,
reason-for-callback: symbol, *new-value*: structure,
user-data: value, *other-values*: sequence)

Argument	Description
win	The window on which the dialog is displayed.
dialog-id	The dialog handle that is returned by the g2-ui-post-custom-dialog system procedure.
control-id	The control-id of the dialog component structure for the updated component.
reason-for-callback	Currently, this value is not used.

Argument	Description
new-value	The control-value of the dialog component structure for the updated component.
other-values	A sequence of control-value structures for controls other than the one for which the update takes place. Currently, this argument contains all other controls.
user-data	An item or value passed to the callback when it is invoked. See Posting a Custom Dialog.

Example: Dialog Update Callback

Here is a simple update callback procedure that posts the results of the callback to the Message Board:

```
dialog-examples-update-callback (win: class g2-window, dialog-id: integer,
    control-id: value, reason-for-callback: symbol, new-value: structure,
    user-data: value, other-values: sequence)
begin
    post "Update callback: g2-window: [the g2-window-remote-host-name of win],
        dialog-id: [dialog-id], control-id: [control-id], reason-for-callback:
        [reason-for-callback], new-value: [new-value], user-data: [user-data],
        other-values: [other-values]";
end
```

Here is the resulting message after clicking the OK button for dialog whose dialog-id is 20. Notice that the reason-for-callback is user-edit, which means the user edited the dialog.

```
#38   10:07:56 a.m.   Update callback: g2-window:
  NNORWALK-N800C,  dialog-id: 20, control-id:
  1, reason-for-callback: USER-EDIT, new-value:
  structure (CONTROL-ID: 1,
  CONTROL-VALUE: structure (TEXT-VALUE:
    "OK")), user-data: 0.0, other-values:
    sequence (structure (CONTROL-ID: 1,
    CONTROL-VALUE: structure (TEXT-VALUE:
      "OK")),
  structure (CONTROL-ID: 2,
    CONTROL-VALUE: structure (TEXT-VALUE:
      "Cancel")))
```

Dialog Dismissed Callback

The dialog-dismissed-callback attribute in the dialog component structure determines the procedure to call when the user dismisses a dialog.

Here is the syntax for the dialog dismissed callback:

```
my-dialog-dismissed-callback
    (win: class g2-window, dialog-id: integer, dialog-was-accepted: truth-value,
    user-data: value,  current-values: sequence)
```

Argument	Description
win	The window on which the dialog is displayed.
dialog-id	The dialog handle that is returned by the g2-ui-post-custom-dialog system procedure.
dialog-was-accepted	True if the user clicked the OK button; false if the user dismissed the dialog without accepting it. See Response Actions.
user-data	An item or value passed to the callback when it is invoked. See Posting a Custom Dialog.
current-values	A sequence of structures for each control in the dialog, where each structure contains the control-id and control-value attributes.

Example: Dialog Dismissed Callback

Here is a simple dismissed callback procedure that posts the results of the callback to the Message Board:

```
dialog-examples-dismissed-callback (win: class g2-window, dialog-id: integer,
    dialog-was-accepted: truth-value, user-data: value, new-values: sequence)
begin
    post "Dismissed callback: g2-window: [the g2-window-remote-host-name of win],
        dialog-id: [dialog-id], dialog-was-accepted: [dialog-was-accepted],
        user-data: [user-data], new-values: [new-values]";
end
```

Here is the resulting message after clicking the OK button for a dialog whose dialog-id is 20. Notice that the dialog-was-accepted is true, which means the user clicked the OK or Apply button.

```
#39   10:07:56 a.m.   Dismissed callback: g2-
  window: NNORWALK-N800C,  dialog-id: 20,
  dialog-was-accepted: true, user-data: 0.0, new-
  values: sequence (structure (CONTROL-ID: 1,
    CONTROL-VALUE: structure (TEXT-VALUE:
      "OK")),
  structure (CONTROL-ID: 2,
    CONTROL-VALUE: structure (TEXT-VALUE:
      "Cancel")))
```

Generic Dialog Callback

The dialog-generic-callback attribute in the dialog component structure determines the procedure to call when a generic dialog update occurs. It is called when the user:

- Edits a control value in the dialog, just like the dialog-update-callback.
- Dismisses the dialog, just like the dialog-dismissed-callback.
- Changes the selected cells in a grid-view.
- Clicks the mouse or presses a key in a grid-view or tabular-view.

The syntax for the generic dialog callback is:

my-dialog-generic-callback
(*event*: symbol, *win*: class g2-window, *dialog-id*: integer, *item*: value, *info*: structure, *user-data*: value)

Argument	Description
event	The event that occurred. The options are: • All controls: USER-EDIT and DISMISSED • grid-view: SELECTION-CHANGED • grid-view or tabular-view: LEFT-CLICK, MIDDLE-CLICK, RIGHT-CLICK, KEY-PRESS, or any combination of modifier keys or DOUBLE combined with LEFT-CLICK, MIDDLE-CLICK, or RIGHT-CLICK, for example: SHIFT+LEFT-CLICK, CONTROL+RIGHT-CLICK, ALT+LEFT-CLICK, DOUBLE+CONTROL+LEFT-CLICK
win	The window on which the dialog is displayed.

Argument	Description
dialog-id	The dialog handle that is returned by the g2-ui-post-custom-dialog system procedure.
item	For the DISMISSED event, true if the user clicked the OK button, or false if the user clicked the Cancel button. For all other events, the control-id of the dialog component structure for the updated component.
info	A structure, which depends on the type of event. For information on the USER-EDIT and DISMISSED events, see below. For information on the selection changed event, and the mouse and key click events, see grid-view and tabular-view.
user-data	Any user-defined value.

The *info* structure for the USER-EDIT event has this syntax:

```
structure
(control-id: control-id,
control-value: new-value,
all-control-values: sequence)
```

The *info* structure for the DISMISSED event has this syntax:

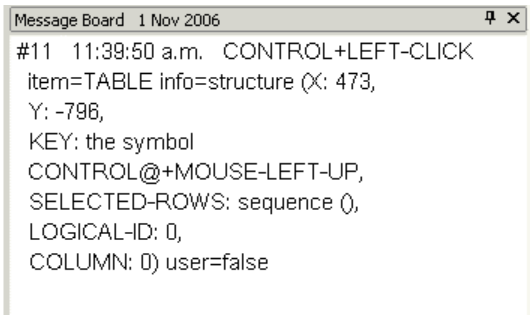
```
structure
(dialog-ok: truth-value,
all-control-values: sequence)
```

Example: Generic Dialog Callback

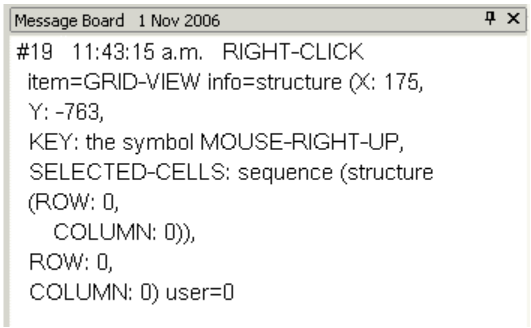
Here is a generic dialog callback procedure that posts the results of the callback to the Message Board:

```
cb (event: symbol, win: class g2-window, control: integer, item: value, info: structure,
    user-data: value)
begin
    post "[event] item=[item] info=[info] user=[user-data]";
end
```

Here is the resulting message after pressing the Control key and clicking the left mouse button in the first row of a tabular-view control:



Here is the resulting message after clicking the right mouse button in the top-left cell of a grid-view control:



Modifying a Custom Dialog

After a dialog has been posted, you can change the contents of a dialog by calling this system procedure:

g2-ui-modify-custom-dialog
(*dialog-handle*: integer, *modify-specification*: sequence, *win*: class g2-window)

Argument	Description
dialog-handle	The dialog handle that is returned by the g2-ui-post-custom-dialog system procedure.
modify-specification	A sequence of structures that describe the dialog components to modify. For a description of this argument, see Modify Specification.
win	The g2-window on which to post the dialog. This window must be running on a Windows machine.

For examples of how to modify individual controls, see the examples under Dialog Controls.

Modify Specification

The *modify-specification* is a sequence of structures that describe the controls to modify, where each structure has the following attributes:

Attribute Name	Type	Required	Default	Description
control-id	integer	Yes	N/A	The control-id of the control to modify.
control-action	symbol	No	replace	The action to perform when modifying the control. See Control Actions.
control-value	Depends on the control action.	Depends on the control action.	N/A	The new value for the control.

Control Actions

The options for control-action are:

Control Action Symbol	Description	Available for Controls	Required Control Value
add	Adds to a control value.	text-box, combo-box, list-box, checkable-list-box	The control value of the item being added.
replace	Replaces all of the existing control values.	label, text-box, push-button, radio-button, toggle-button, spinner, check-box, combo-box, list-box, checkable-list-box, group, full-color-picker, progress-bar, track-bar, slider, image, spinner	The control value of the item being updated.
hide	Hides the control.	All control types	No control value required.

Control Action Symbol	Description	Available for Controls	Required Control Value
show	Shows the control.	All control types	No control value required.
enable	Enables the control.	All control types	No control value required.
disable	Disables the control.	All control types	No control value required.
check	Checks the control.	radio-button, check-box, toggle-button	No control value required.
uncheck	Unchecks the control.	radio-button, check-box, toggle-button	No control value required.
dropped-height	Changes the height.	tree-view-combo-box	The height, in pixels, as an integer.
dropped-width	Changes the width.	tree-view-combo-box	The width, in pixels, as an integer.
selected-tab	Changes the selected tab.	tab-frame	A text of the selected tab. See Example: Modifying the Selected Tab.
add-rows	Adds new rows of data to a tabular-view control.	tabular-view	See Example: Adding a Row to a Tabular-View Control.
add-column	Adds a new column to a tabular-view control.	tabular-view	See Example: Adding Columns to a Tabular View.
remove-rows	Removes rows of data from a tabular-view control.	tabular-view	See Example: Deleting a Row from a Tabular-View Control

Control Action Symbol	Description	Available for Controls	Required Control Value
remove-columns	Removes columns of data from a tabular-view control.	tabular-view	The control-value is a sequence of integer column numbers, where -1 means the last column.
replace-rows	Replaces all values in specific rows in a tabular-view control.	tabular-view	See Example: Replacing Rows in a Tabular View.
replace-cells	Replaces specific cell values in a tabular-view control.	tabular-view	See Example: Replacing Cells in a Tabular View.
remove-all-rows	Removes all the rows in a tabular-view control.	tabular-view	See Example: Removing all Rows in a Tabular View.
remove-all-selected-rows	Removes all the selected rows in a tabular-view control.	tabular-view	
deselect-rows	De-select specific rows in a tabular-view control.	tabular-view	The control-value is a sequence of integer row numbers, where -1 means the last row.

Example: Modifying a Custom Dialog

This example shows the *modify-specification* for a number of controls within a custom dialog:

```
sequence(
    structure(control-id: 1, control-value: structure(text-value: "Yes")),
    structure(control-id: 9, control-action: the symbol disable),
    structure(control-id: 11, control-action: the symbol hide),
    structure(control-id: 21, control-action: the symbol check))
```

For examples of how to modify individual controls, see the examples under Dialog Controls.

Querying a Dialog

After a dialog has been posted, you can query its contents by using the following system procedure:

g2-ui-query-custom-dialog-values
(*dialog-handle*: integer, *controls*: sequence, *win*: class g2-window)
-> *control-values*: sequence

Argument	Description
dialog-handle	The dialog handle that is returned by the g2-ui-post-custom-dialog system procedure.
controls	This argument is currently not used.
win	The g2-window on which to post the dialog. This window must be running on a Windows machine.

Return Value	Description
control-values	A sequence of current control values for all controls in the dialog.

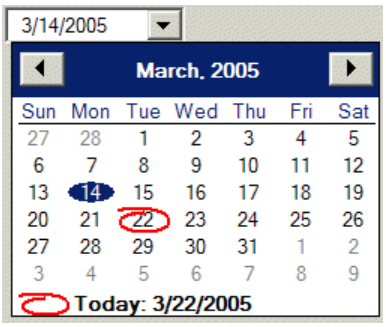
Dialog Controls

This section describes each of the dialog controls in detail by providing:

- An example of each control within a dialog.
- Control-specific attributes.
- An example of the dialog component structure for the control.
- An example of how to modify the control.

calendar

The calendar control provides a dropdown calendar for configuring the date. You configure the date by clicking the date in the calendar. The control circles today's date.



Specific Attributes for Calendar Control

Attribute Name	Type	Required	Default	Description
long-date-format	truth-value	no	false	Whether to display the date in a long format, such as Friday, April 14, 2005. By default, the date is displayed in a short format, such as 4/19/05.
control-value	structure	yes	the current date	A structure that specifies the default date: structure (selected-year: *integer*, selected-month: *integer*, selected-date: *integer*)

The selected-year must be between 1601 and 30827, the selected-month must be between 1 and 12, and the selected-date must be between 1 and 31.

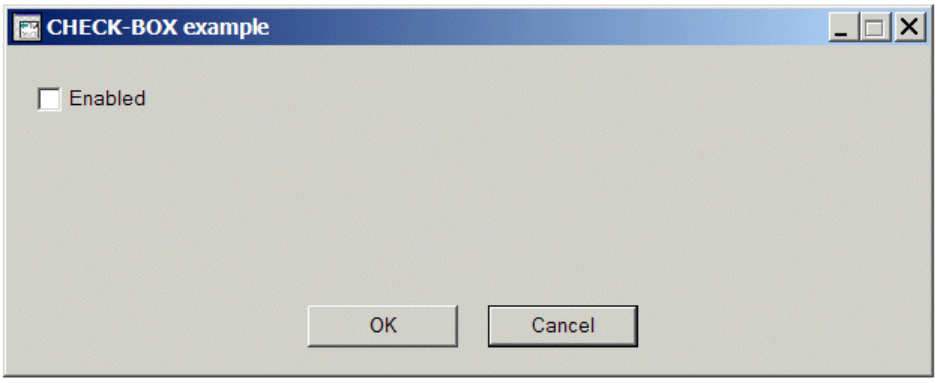
Example: Dialog Specification for Calendar Control

Here is the specification for the calendar control shown above:

```
structure
    (control-type: the symbol CALENDAR,
    control-id: the symbol MY-CALENDAR,
    control-value:
        structure
            (selected-year: 2005,
            selected-month: 3,
            selected-date: 14)
    . . .)
```

check-box

The check-box control provides a toggle for enabling or disabling a value:



Specific Attributes for Check-Box Control

Attribute Name	Type	Required	Default	Description
control-value	structure	yes	N/A	A structure that specifies the label for the check box and whether it is initially selected: structure (text-value: *text*, selected: *truth-value*)

Example: Dialog Specification for Check-Box Control

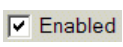
Here is the dialog specification for the check-box control portion of the dialog above, which specifies a single check box that is initially unchecked:

```
structure
    (dialog-title: "CHECK-BOX example",
    dialog-width: 310,
    dialog-height: 124,
    dialog-is-modal: true,
    dialog-update-callback: dialog-examples-update-callback,
    dialog-dismissed-callback: dialog-examples-dismissed-callback,
    components:
        sequence
            (structure
                (control-type: the symbol push-button,
                . . .
                control-value: structure (text-value: "OK")),
            structure
                (control-type: the symbol push-button,
                . . .
                control-value: structure (text-value: "Cancel")),
            structure
                (control-type: the symbol check-box,
                control-id: 3,
                height: 15,
                width: 100,
                top: 10,
                left: 10,
                response-action: the symbol ignore,
                control-value: structure (text-value: "Enabled"))))
```

Example: Checking a Check-Box Control

Use the check control action to check a check-box control:

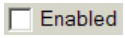
```
call g2-ui-modify-custom-dialog
    (dialog-id,
    sequence
        (structure
            (control-action: the symbol check,
            control-id: 3)),
    window);
```



Example: Unchecking a Check-Box Control

Use the uncheck control action to uncheck a check-box control:

```
call g2-ui-modify-custom-dialog
     (dialog-id,
     sequence
          (structure
               (control-action: the symbol uncheck,
               control-id: 3)),
     window);
```

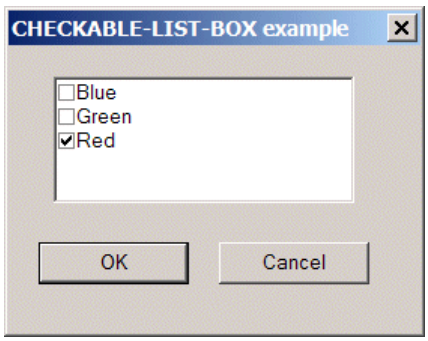


checkable-list-box

The checkable-list-box control is similar to the list-box control except that entries have check-boxes for selecting and deselecting entries in the list. Otherwise, the checkable-list-box supports all the same options as the list-box control, including single, extended, and multiple selection, and sorting. When using extended and multiple selection, you can select multiple elements in the list, then check or uncheck all elements in the selection with a single mouse click.

The control adds vertical scrollbar if the number of items in the list is bigger than the height of the control.

Here is a checkable list box control with one option checked:



Specific Attributes for Checkable-List-Box Control

Attribute Name	Type	Required	Default	Description
extended-selection	truth-value	no	false	When true, enables extended selection and disables multiple selection, if it was enabled. When false, disables extended selection and does nothing to multiple selection.
multiple-selection	truth-value	no	false	When true, enables multiple selection and disables extended selection, if it was enabled. When false, disables both multiple selection and extended selection.

Attribute Name	Type	Required	Default	Description
single-selection	truth-value	no	true	When true, disables both extended selection and multiple selection. When false, enables extended selection and disables multiple selection, if it was enabled.
sort-list	truth-value	no	false	When true, sorts the strings alphabetically. When false, displays the strings in the order supplied in the text-sequence of the control-value.
control-value	structure	yes	N/A	A structure that specifies the sequence of text values for each element in the checked list box, the initially checked values, and the initially selected values: structure (text-sequence: sequence (*text*[, ...]), checked: sequence (*text*[, ...]), selected: sequence (*text*[, ...]))

Example: Dialog Specification for Checkable-List-Box Control

Here is the dialog specification for the checkable-list-box control portion of the dialog above:

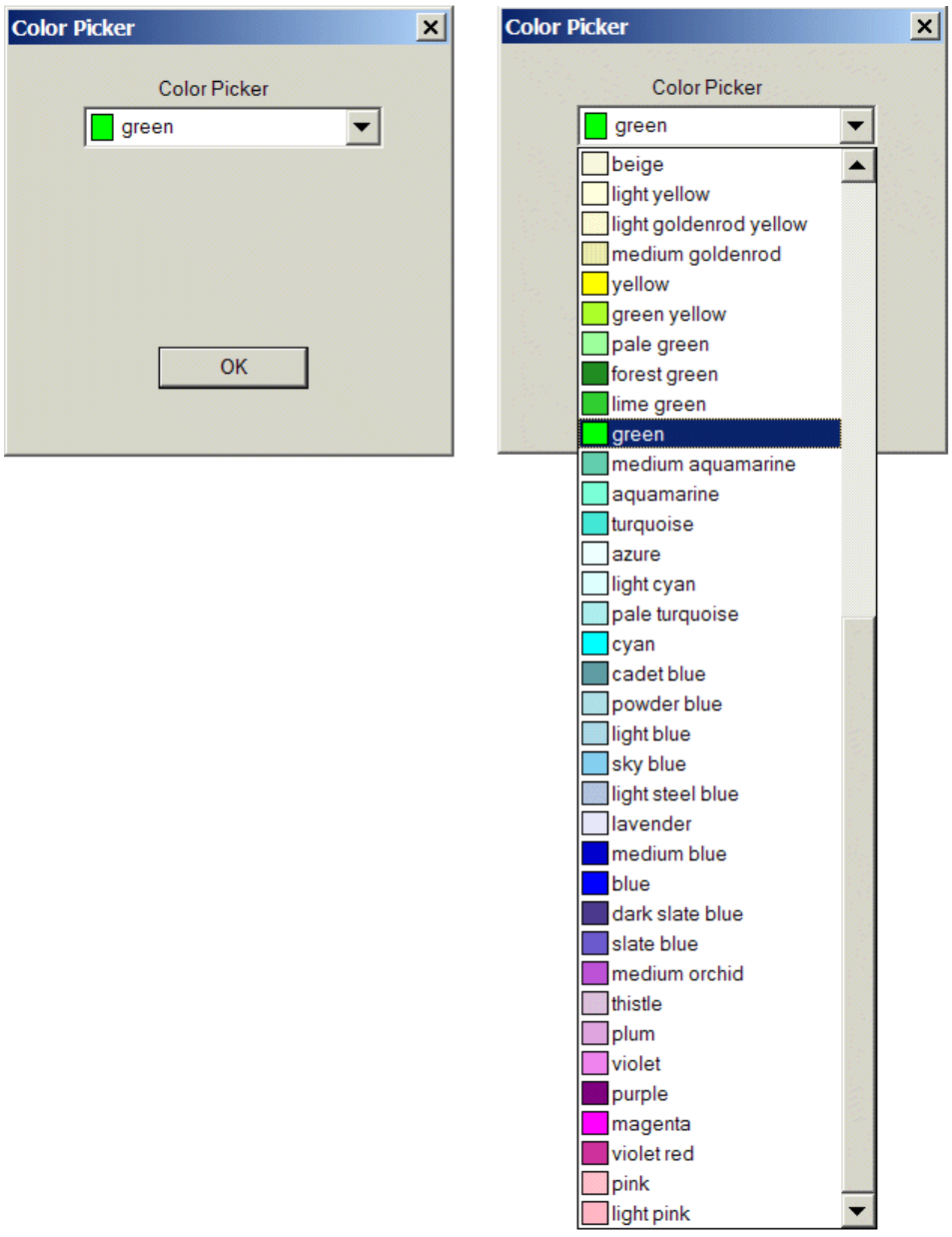
```
structure
    (control-type: the symbol checkable-list-box,
    control-id: 3,
    height: 50,
    width: 100,
    top: 10,
    left: 15,
    response-action: the symbol ignore,
    control-value: structure (text-sequence: sequence ("Red", "Green", "Blue"),
    checked: sequence ("Red")),
```

color-picker

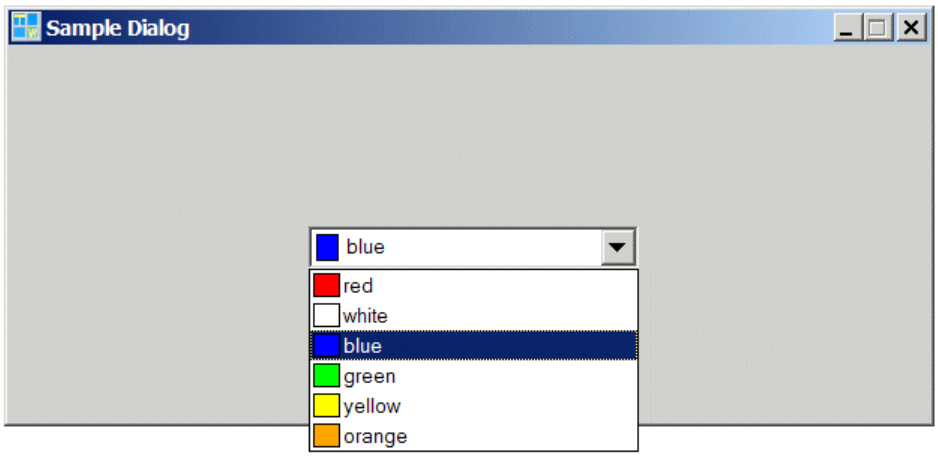
The color-picker control provides dropdown list for choosing a color. By default, the list of colors is determined by the value of the colors-on-2nd-level-color-menu in the Color Parameters system table, whose options are all, standard set, or a list of colors. This attribute determines the colors on the second-level color menu, which you access by choosing More on a G2 color palette.

You can specify the currently selected color in the control-value for the control. You can also provide a specific list of colors, rather than using the default list.

Here is a custom dialog with a color-picker control that uses the default set of colors for the second-level color menu, which is all, with green as the currently selected color:



Here is a custom dialog with a color-picker control that specifies a list of colors:



Specific Attributes for Color-Picker Control

Attribute Name	Type	Required	Default	Description
control-value	structure	yes	structure()	A structure that specifies the currently selected color, as a symbol, and the sequence of colors, as symbols, to display in the control: structure (selected: *color*, colors: *sequence*)

When specifying a list of colors, the return value in the callback contains the list of colors.

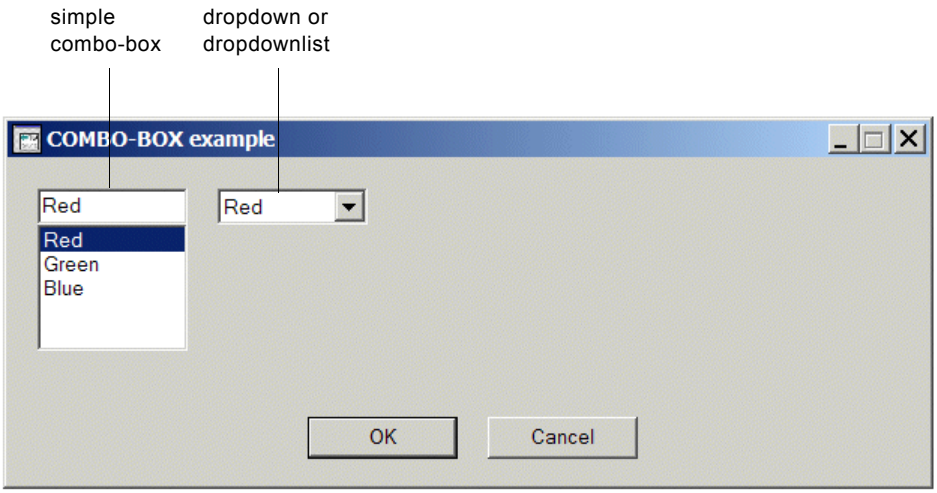
Example: Dialog Specification for Color-Picker Control

Here is the dialog specification for the color-picker control portion of the second dialog above, which specifies a list of colors:

```
structure
    (control-type: the symbol color-picker,
    control-id: the symbol color-picker-with-invalid-symbol,
    control-value: structure
        (colors: sequence
            (the symbol red,
            the symbol white,
            the symbol blue,
            the symbol green,
            the symbol yellow,
            the symbol orange),
        selected: the symbol blue),
    response-action: the symbol respond,
    height: 375,
    width: 110,
    left: 100,
    top: 60)
```

combo-box

The combo-box control provides a static list of choices or a dropdown list. You can configure the combo box to be a static list that is also editable through the type-in box (simple), a dropdown list that is also editable through the type-in box (dropdown), or a dropdown list that is not editable through the type-in box (dropdownlist).



Specific Attributes for Combo-Box Control

Attribute Name	Type	Required	Default	Description
list-box-style	symbol	no	simple	The type of combo box, which can be on of the symbols: • simple — The choices are always visible, and the type-in box is editable. • dropdown — The choices appear in a dropdown list, and the type-in box is editable. • dropdownlist — The choices appear in a dropdown list, and the type-in box is not editable.
control-value	structure	yes	N/A	A structure that specifies a sequence of text values in the combo box, the text of the initially selected value, the selected text, and the width of the combo-box: structure (text-sequence: sequence (*text*[, ...]) | ( ), selected: *text*, text-selection: *index* | sequence (*index, index*) dropdown-width: *integer*)

In the control-value, specify text-selection as an integer to set the initial cursor position at the specified index or as a sequence to set the initial selection between the specified indices, where *index* is a zero-based integer index. The text-selection option only works if the list-box-style is simple or dropdown.

The selection is returned as the text-selection attribute of the control-value, as either an integer or a sequence.

Note The height of a combo-box specifies the number of visible rows. The minimum height is 1. Currently, the height is ignored in standard Telewindows (`tw.exe`).

Example: Dialog Specification for Combo-Box Control

Here is the dialog specification for the combo-box control portion of the dialog above, which specifies a simple combo box and a dropdown list:

```
structure
    (dialog-title: "COMBO-BOX example",
    dialog-width: 310,
    dialog-height: 124,
    dialog-is-modal: true,
    dialog-update-callback: dialog-examples-update-callback,
    dialog-dismissed-callback: dialog-examples-dismissed-callback,
    components:
        sequence
            (structure
                (control-type: the symbol push-button,
                . . .
                control-value: structure (text-value: "OK")),
            structure
                (control-type: the symbol push-button,
                . . .
                control-value: structure (text-value: "Cancel")),
            structure
                (control-type: the symbol combo-box,
                control-id: 3,
                height: 60,
                width: 50,
                left: 10,
                top: 10,
                response-action: the symbol ignore,
                list-box-style: the symbol simple,
                control-value:
                    structure
                        (text-sequence: sequence ("Red", "Green", "Blue")
                        selected: "Red")),
            structure
                (control-type: the symbol combo-box,
                control-id: 4,
                height: 60,
                width: 50,
                left: 70,
                top: 10,
                response-action: the symbol ignore,
                list-box-style: the symbol dropdownlist,
```

```
                control-value:
                    structure
                        (text-sequence: sequence ("Red", "Green", "Blue",
                        "Yellow", "Orange", "Purple", "Mauve", "Violet", "Taupe"),
                        selected: "Red"))))
```

Example: Adding and Replacing Elements in a Combo-Box Control

Use the add control action to add elements to a combo-box control, and use the replace control action to replace elements:

```
call g2-ui-modify-custom-dialog
    (dialog-id,
    sequence
        (structure (  control-action: the symbol add,
                      control-id: 3,
                      control-value: structure
                          (text-sequence: sequence ("New-1", "New-2", "New-3"))),
        structure (   control-action: the symbol replace,
                      control-id: 4,
                      control-value: structure
                          (text-sequence: sequence ("New-1", "New-2", "New-3"))))
    window);
```

duration

The duration control provides spinners for configuring the weeks, days, hours, minutes, and seconds. You select the part of the duration you want to configure, then click the up or down arrow to change its value. You can also enter the value directly.

The duration control provides a tooltip when hovering the mouse over each field to indicate the units: weeks, days, hours, minutes, and seconds.

Note When using the spinners to enter a value for days that is greater than 365, it cycles back to 0, whereas when entering a value that is greater than 365 from the keyboard, the value is the last valid value.

Specific Attributes for Duration Control

Attribute Name	Type	Required	Default	Description
control-value	structure	yes	N/A	A structure that specifies the default weeks, days, hours, minutes, and seconds: structure (number-of-weeks: *integer*, number-of-days: *integer*, number-of-hours: *integer*, number-of-minutes: *integer*, number-of-seconds: *integer*)

Note The duration control converts days to weeks when the number-of-days in the control-value exceeds 6. However, the user can enter a value of up to 365 days in the control, and the return value is given in days.

Example: Dialog Specification for Duration Control

Here is the specification of the duration control above:

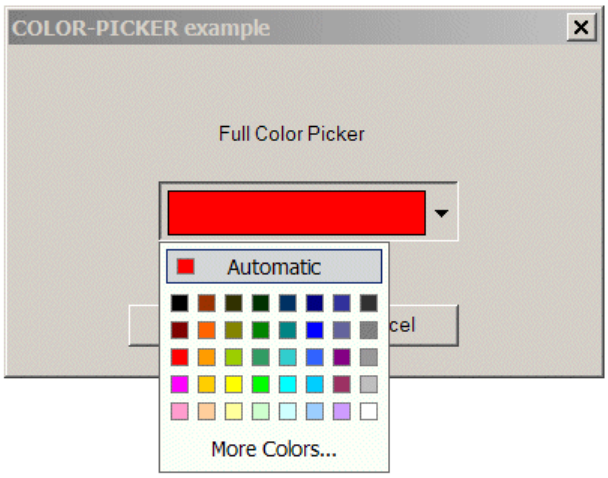
```
structure
    (control-type: the symbol DURATION,
    control-id: the symbol MY-DURATION,
    control-value:
        structure (number-of-weeks: 1,
                number-of-days: 3,
                number-of-hours: 5,
                number-of-minutes: 30,
                number-of-seconds: 15),
    . . . )
```

full-color-picker

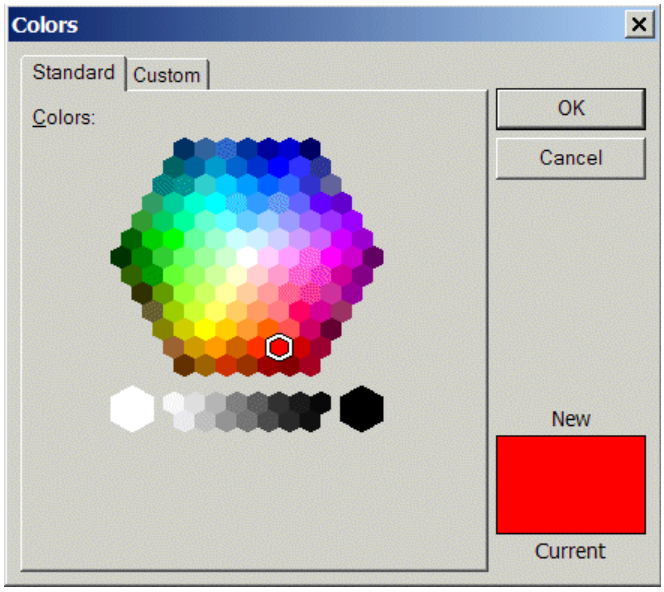
The full-color-picker control provides dropdown list for choosing a 24-bit color. You specify the default color by specifying any RGB-color symbol, for example, RGBFF0000 (red).

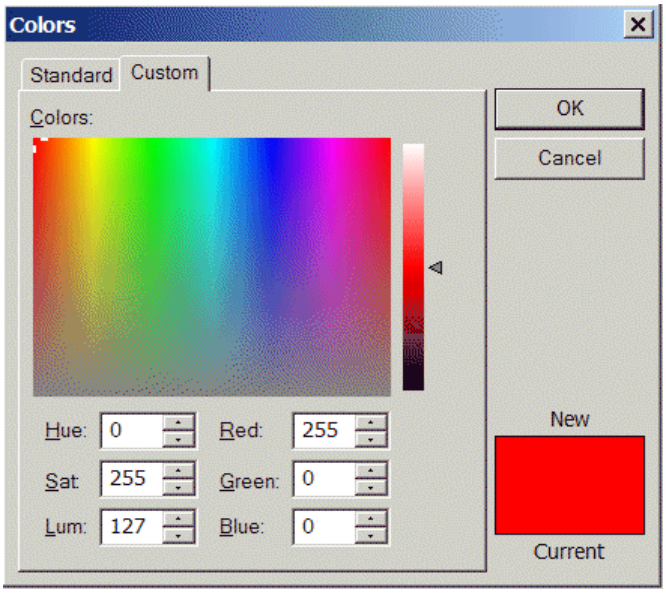
You can specify the currently selected color in the control-value for the control.

Here is a custom dialog with a full-color-picker control, with RGBFF0000 as the currently selected color:



Here are the Standard and Custom tabs of the dialog that appears when you click More Colors:





Specific Attributes for Full-Color-Picker Control

Attribute Name	Type	Required	Default	Description
control-value	structure	yes	structure()	A structure that specifies the currently selected color, as an RGB color symbol, such as RGBFF0000: structure (selected: *rgb-color*)

Example: Dialog Specification for Full-Color-Picker Control

Here is the dialog specification for the full-color-selection control portion of the dialog above:

```
structure
    (control-type: the symbol full-color-picker,
    control-id: the symbol my-full-color-picker,
    control-value: structure (selected: the symbol RGBFF0000),
    response-action: the symbol respond,
    height: 375,
    width: 100,
    left: 25,
    top: 20)
```

grid-view

The grid-view control shows a grid that contains multiple columns and rows of cells, which can contain read-only and editable cells.

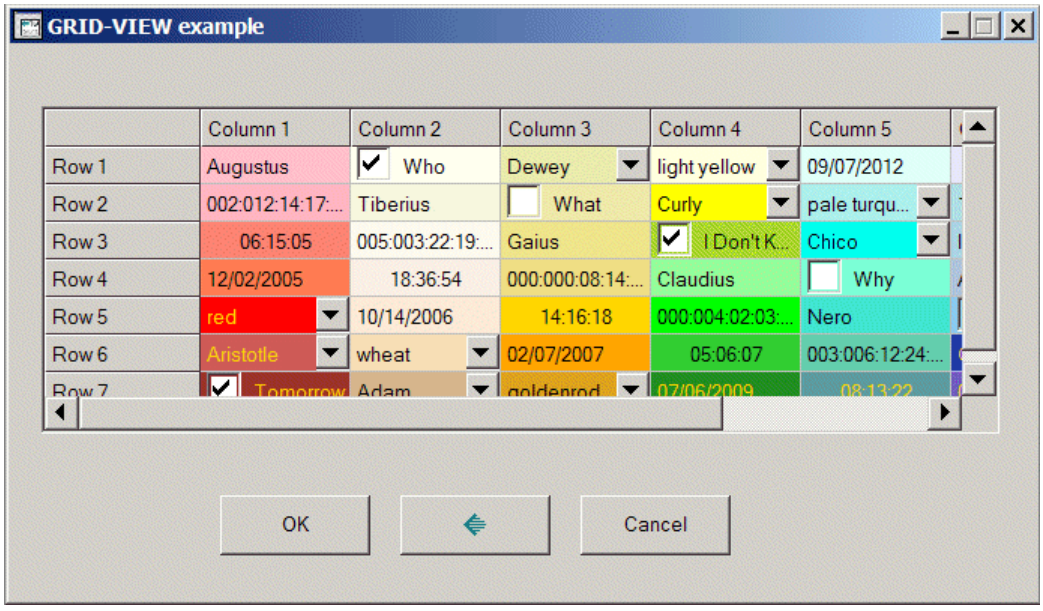
You specify a sequence of columns, including the column header, the column width, the default cell type and value, whether the cells are read-only by default, and the default colors for each column. You specify a sequence of row data for each column, including the row heading, the row height, and the individual cell settings.

Each cell can contain any of these types of data or controls: integer, quantity, text-box, check-box, combo-box, color-picker, duration, calendar, time-of-day, spinner, and image. The cell value specification depends on the type of control in the cell.

The grid-view control supports sorting of columns. When sorting is enabled, clicking the column header displays an arrow for sorting the values in ascending or descending order. When sorting in ascending order, numbers precede uppercase letters, and uppercase letters precede lowercase letters. Numeric sorting only works if all cells in the column contain cells of type integer or quantity.

The grid-view control displays an error when attempting to create a grid view with zero columns.

Here is a grid view that shows examples of each type of control in two columns. The user can interactively resize the column width and row height in the grid.



Control-Value

Attribute Name	Type	Required	Default	Description
control-value	structure	yes	N/A	A structure that specifies the rows and columns in the grid view. The number of cells specified in each row must be the same as the number of columns. The control value also specifies the currently selected cells. structure (columns: *sequence-of-columns*, rows: *sequence-of-rows*, selected-cells: *sequence-of-cells*, use-column-header: *truth-value*, use-row-header: *truth-value*)

The control-value structure has these attribute specifications:

Argument	Type	Required	Default	Description
columns	sequence	yes	N/A	sequence (*column-spec*[, ...])
rows	sequence	yes	N/A	sequence (*row-spec*[, ...])
selected-cells	sequence	no	sequence()	sequence (structure (row: *row-id*, column: *column-id*) [, ...]) The *row-id* and *column-id* are zero-based index values, where (0,0) indicates the top-left cell, not including the headers.
use-column-header	symbol	no	true	Whether to show the column header.
use-row-header	symbol	no	true	Whether to show the row header.

Column-Spec

The *column-spec* is a structure with these attributes:

Attribute Name	Type	Required	Default	Description
alignment	symbol	no	left	The alignment of the control in the cell. The options are left, right, and center. **Note:** The alignment attribute does not apply to cells of type ellipsis-button or image.
background-color	symbol	no	white	The default background color for cells in the row.
bold	truth-value	no	false	When true, specifies that all text in the column is bold.
default-cell-type	symbol	no	text-box	The default cell type for the column. The options are: integer, quantity, text-box, check-box, combo-box, color-picker, duration, calendar, time-of-day, spinner, and image.
default-cell-value	structure	no	N/A	The default value for the controls in the row of cells. The contents vary by cell type. See Cell-Value-Spec.

Attribute Name	Type	Required	Default	Description
ellipsis-button	truth-value	no	false	For text-box cell types, whether to display an ellipsis button (...) next to all cells in the column of type text-box. When true, the value is displayed in a text box that the user cannot edit. Clicking the button sends an event notification in the dialog update callback. This attribute is only relevant for text-box cell types and ignored by all other cell types. For a description of the dialog update callback and an example, see Launching a Custom Cell Editor.
read-only	truth-value	no	false	The default read-only state of the cells in the row.
text-color	symbol	no	black	The default text color for cells in the row.
text-value	text	no	none	The column header text.
width	integer	no	N/A	The width of the column in dialog units. The default width is computed, based on the overall width of the grid view and the number of columns.

Note For any cell in a grid-view control, any attribute that is not specified in the cell-value but is specified in the default-cell-value of the column is inherited, that is, it uses the value from the column. For example, for a combo-box cell, you can supply the text-sequence on a per-column basis and supply just the selected value on a per-cell basis, and the text-sequence is inherited.

Row-Spec

The *row-spec* is a structure with these attributes:

Attribute Name	Type	Required	Default	Description
cell-settings	sequence	yes	N/A	A sequence of structures that specifies the data in each cell, where each structure corresponds with the data for each column in the row. The attributes are similar to those in the *column-spec* except they are for an individual cell.
height	integer	no	N/A	The height of the row in dialog units. The default height is computed, based on the overall height of the grid view and the number of rows.
text-value	text	no	N/A	The row header text.

Cell-Setting-Spec

The *cell-setting-spec* is a sequence of structures, where each structure has these attributes:

Attribute Name	Type	Required	Default	Description
alignment	symbol	no	left	The alignment of the control in the cell. The options are left, right, and center.
background-color	symbol	no	white	The background color for the cell.
bold	truth-value	no	false	When true, specifies that the text is bold.
cell-type	symbol	no	text-box	The cell type for the cell. The options are: integer, quantity, text-box, check-box, combo-box, color-picker, duration, calendar, time-of-day, spinner, and image.

Attribute Name	Type	Required	Default	Description
cell-value	structure	no	N/A	The value for the control in the cell. The contents vary by cell type. See Cell-Value-Spec.
ellipsis-button	truth-value	no	false	For text-box cell types, whether to display an ellipsis button (...) next to the cell. When true, the value is displayed in a text box that the user cannot edit. Clicking the button sends an event notification in the dialog update callback. This attribute is only relevant for text-box cell types. For a description of the dialog update callback and an example, see Launching a Custom Cell Editor.
read-only	truth-value	no	false	The read-only state of the cell.
sorting	truth-value	no	false	When true, allows sorting of cells by clicking the column header. Numeric sorting only works if all cells in the column contain cells of type integer or quantity.
text-color	symbol	no	black	The text color for the cell.

Cell-Value-Spec

The *cell-value-spec* structure has different syntax, depending on the cell-type:

Cell-Type	Syntax	Description
calendar	structure (selected-year: *integer*, selected-month: *integer*, selected-date: *integer*, calendar-format: *symbol*)	The currently selected year, month, and date, and the calendar format. The selected-year must be between 1970 and 3000, the selected-month must be between 1 and 12, and the selected-date must be between 1 and 31. You specify the value through spinners. The calendar-format is one of the symbols MM-DD-YYYY (the default), DD-MM-YYYY, and YYYY-MM-DD. **Note:** If you create a grid-view with a calendar control without specifying the default date, G2 uses "the current date" to create the control. Note that if G2 and Telewindows are in different time zones, and if the dialog is created at a time of day when the date is different in the two time zones, the control uses the date in G2's time zone.
check-box	structure (text-value: *text*, selected: *truth-value*)	• text-value is the text to appear to the right of the check-box. It defaults to no text. • selected is whether the check-box should be checked or unchecked. The default is false (unchecked).

Cell-Type	Syntax	Description
combo-box	structure (text-sequence: sequence (*text*[, ...]) | ( ), selected: *text*, dropdown-width: *integer*, list-box-style: *symbol*)	• text-sequence is a sequence of text values, which are the values for the combo-box specified from the top down. • selected is the text value that is selected initially. • dropdown-width is the width of the dropdown box. • list-box-style is one of the following symbols: – dropdown — Allows entering any value via an edit box, the default. – dropdownlist — Allows only selections from the dropdown list.
color-picker	structure (selected: *symbol*)	The value of the color that is initially selected.
duration	structure (number-of-weeks: *integer*, number-of-days: *integer*, number-of-hours: *integer*, number-of-minutes: *integer*, number-of-seconds: *integer*)	The number of weeks, days, hours, minutes, and seconds. **Note:** You can only enter values directly in the control for hours, minutes, and seconds. You cannot currently enter values directly for weeks and days, although you can enter their values by using the spinners.
image	structure (icon: *item-or-symbol*)	The icon to display in the cell, which can be a G2 class name, an item, or a built-in GMS icon. For a list of built-in GMS icons, see image. When specifying an item, the icon updates dynamically as the G2 icon changes. If the icon is too large for its cell, the portion that fits appears in the cell. If an icon is smaller than 32x32 pixels, the icons is drawn at its actual size.
integer	structure (current-value: *integer*)	The cell accepts integer values only.

Cell-Type	Syntax	Description
quantity	structure (current-value: *quantity*)	The cell accepts quantity values only.
spinner	structure (current-value: *quantity*, low-value: *quantity*, high-value: *quantity*, increment: *quantity*)	The current value, low and high values, and increment for the spinner.
text-value	structure (text-value: *text*)	The text to appear in the text box. It defaults to no text.
time-of-day	structure (selected-hour: *integer*, selected-minute: *integer*, selected-second: *integer*)	The currently selected hour, minute, and second. The selected-hour must be between 0 and 23, the selected-minute must be between 0 and 59, and the selected-second must be between 0 and 59. **Note:** If you use the default value, which is the current time, the current time is displayed in GMT of the Telewindows process. If you set the time explicitly, the control returns the hour exactly as it appears in the control. If Telewindows is running in a different time zone from G2, it is up to the application developer to decide how to interpret the time.

Dialog Dismissed Callback

The *current-values* argument to the dialog dismissed callback returns the currently selected cells in the grid, using this syntax:

```
selected-cells:
    (sequence
        (structure (row: integer, column: integer),
        (structure (row; integer, column: integer),
        . . .)
```

For example, the dialog dismissed callback returns this value for selected-cells if the first and second cells of the first row are selected at the time the dialog is dismissed:

```
selected-cells:
    (sequence
        (structure (row: 1, column: 1),
        (structure (row; 1, column: 2))
```

Generic Dialog Callback

The dialog-generic-callback for a custom dialog, if specified, is called whenever the set of selected cells changes in a grid-view on the dialog, with the following arguments:

- *event*: SELECTION-CHANGED
- *control-id*: integer
- *info*: structure (row: *integer*, column: *integer*, selection: *sequence*)

  where:

  - row and column describe a particular cell in the selection, either the cell with the focus, or if no cell has the focus, the top-left-most cell in the selection. The value for row and column are -1 if the selection is empty.
  - selection is a sequence of structures representing the rectangular regions of contiguously selected cells in the unsorted grid, where each structure has this syntax:

    ```
    structure
    (minrow: integer,
    maxrow: integer,
    mincol: integer,
    maxcol: integer)
    ```

- *user-data*: value

The dialog-generic-callback for a custom dialog, if specified, is called whenever the user clicks the mouse or presses a key in the grid-view on the dialog, with the following arguments:

- *event*: LEFT-CLICK, MIDDLE-CLICK, RIGHT-CLICK, KEY-PRESS, or any combination of modifier keys or DOUBLE combined with LEFT-CLICK, MIDDLE-CLICK, or RIGHT-CLICK, for example: SHIFT+LEFT-CLICK, CONTROL+RIGHT-CLICK, ALT+LEFT-CLICK, DOUBLE+CONTROL+LEFT-CLICK
- *control-id*: integer

- *info*: structure
  (x: *integer*,
  y: *integer*,
  selected-cells: *sequence*,
  key: *symbol*,
  row: *integer*,
  column: *integer*)

  where:

  - x and y are the x-y coordinates of the mouse click in the grid view.
  - selected-cells is a sequence of structures representing the row and column of each selected cell, where each structure has this syntax:

    structure
    (row: *integer*,
    column: *integer*)

  - key is the key that was pressed, if any.
  - row and column describe a particular cell in the selection, either the cell with the focus, or if no cell has the focus, the top-left-most cell in the selection. The value for row and column are -1 if the selection is empty.
- *user-data*: value

For an example, see Example: Generic Dialog Callback.

System Procedures

Use the following system procedures to insert and delete columns and rows and to replace cells in a grid-view:

g2-ui-grid-view-insert-column
(*dialog*: integer, *control*: value, *column*: integer, *column-spec*: structure, *win*: class g2-window)

Inserts a column into the grid-view specified by *control*, which is the control-id of the control, in *dialog*. The *column* is the *column-id* of the new column, after it has been inserted. The *column* argument must not be greater than the number of columns in the grid after the new column has been added. For example, if the grid has columns 0 - 4 (5 columns), the *column* argument can be 0 - 5, inclusive. A column number of -1 means to insert as the last column.

For a description of *column-spec*, see grid-view in Custom Windows Dialogs in the *G2 Reference Manual*. The *column-spec* must specify the same number of rows as the existing columns in the grid.

g2-ui-grid-view-delete-column
(*dialog*: integer, *control*: value, *column*: integer, *win*: class g2-window)

Deletes a column from the grid-view specified by *control*, which is the control-id of the control, in *dialog*. The *column* is the *column-id* of the column to delete, which must be a valid column in the grid. A column number of -1 means to delete the last column.

g2-ui-grid-view-insert-row
(*dialog*: integer, *control*: value, *row*: integer, *row-spec*: structure, *win*: class g2-window)

Inserts a row into the grid-view specified by *control*, which is the control-id of the control, in *dialog*. The *row* is the *row-id* of the new row, after it has been inserted. The *row* argument must not be greater than the number of rows in the grid after the new row has been added. For example, if the grid has rows 0 - 4 (5 rows), the *row* argument can be 0 - 5, inclusive. A row number of -1 means to insert as the last row.

For a description of *row-spec*, see Row-Spec. The *row-spec* must specify the same number of columns as the existing rows in the grid.

g2-ui-grid-view-delete-row
(*dialog*: integer, *control*: value, *row*: integer, *win*: class g2-window)

Deletes a row from the grid-view specified by *control*, which is the control-id of the control, in *dialog*. The *row* is the *row-id* of the row to delete, which must be a valid row in the grid. A row number of -1 means to delete the last row.

g2-ui-grid-view-replace-cell
(*dialog*: integer, *control*: value, *row*: integer, *column*: integer, *cell-spec*: structure, *window*: class g2-window)

Replaces the cell at the given *row* and *column* positions in a grid-view specified by *control*, which is the control-id of the control, in *dialog*. You can replace all grid view attributes of all control types, using the following syntax for *cell-spec*:

structure
(cell-type: the symbol *cell-type*,
cell-value: structure ([*attribute*: *value*] ...))

For example, to replace a cell with a text-box cell with a given text and background color, the *cell-spec* would look like this:

```
structure
(cell-type: the symbol text-box,
cell-value:
    structure (text-value: “New Text”, background-color: the symbol blue) )
```

Example

This example shows the specification for the grid-view shown earlier:

```
sequence (structure (control-type: the symbol grid-view,
   control-id: the symbol my-sole-grid-view,
   height: 110,
   width: 320,
   left: 10,
   top: 20,
   response-action: the symbol respond,
   control-value: structure (columns: sequence (structure (width: 100,
        text-value: "Column 1",
        read-only: false),
      structure (text-value: "Column 2",
        read-only: false),
      structure (text-value: "Column 3",
        read-only: false),
      structure (text-value: "Column 4",
        read-only: false),
      structure (text-value: "Column 5",
        read-only: false),
      structure (text-value: "Column 6",
        read-only: false),
      structure (text-value: "Column 7",
        read-only: false)),
    rows: sequence (structure (text-value: "Row 1",
      cell-settings: sequence (structure (text-color: the symbol black,
         background-color: the symbol pink,
         cell-value: structure (text-value: "Augustus")),
        structure (cell-type: the symbol check-box,
         background-color: the symbol ivory,
         cell-value: structure (selected: true,
          text-value: "Who")),
        structure (cell-type: the symbol combo-box,
         background-color: the symbol medium-goldenrod,
         cell-value: structure (text-sequence: sequence ("Huey",
            "Dewey",
            "Louie"),
          selected: "Dewey")),
        structure (cell-type: the symbol color-picker,
         background-color: the symbol light-yellow,
         cell-value: structure (selected: the symbol light-yellow)),
        structure (cell-type: the symbol calendar,
         background-color: the symbol light-cyan,
         cell-value: structure (selected-year: 2012,
          selected-month: 9,
          selected-date: 7)),
        structure (cell-type: the symbol time-of-day,
         background-color: the symbol lavender,
         show-date: true,
```

```
      cell-value: structure (selected-year: 2005,
        selected-month: 8,
        selected-date: 24,
        selected-hour: 1,
        selected-minute: 2,
        selected-second: 3)),
    structure (cell-type: the symbol duration,
      background-color: the symbol thistle,
      cell-value: structure (number-of-weeks: 1,
        number-of-days: 1,
        number-of-hours: 1,
        number-of-minutes: 1,
        number-of-seconds: 1)))),
structure (text-value: "Row 2",
  cell-settings: sequence (structure (cell-type: the symbol duration,
      background-color: the symbol light-pink,
      cell-value: structure (number-of-weeks: 2,
        number-of-days: 12,
        number-of-hours: 14,
        number-of-minutes: 17,
        number-of-seconds: 35)),
    structure (background-color: the symbol beige,
      cell-value: structure (text-value: "Tiberius")),
    structure (cell-type: the symbol check-box,
      text-color: the symbol black,
      background-color: the symbol pale-goldenrod,
      cell-value: structure (selected: false,
        text-value: "What")),
    structure (cell-type: the symbol combo-box,
      background-color: the symbol yellow,
      cell-value: structure (text-sequence: sequence ("Moe",
          "Larry",
          "Curly"),
        selected: "Curly")),
    structure (cell-type: the symbol color-picker,
      background-color: the symbol pale-turquoise,
      cell-value: structure (selected: the symbol pale-turquoise)),
    structure (cell-type: the symbol calendar,
      text-color: the symbol black,
      background-color: the symbol powder-blue,
      cell-value: structure (selected-year: 2013,
        selected-month: 10,
        selected-date: 18)),
    structure (cell-type: the symbol time-of-day,
      background-color: the symbol plum,
      cell-value: structure (selected-hour: 12,
        selected-minute: 24,
        selected-second: 36)))),
structure (text-value: "Row 3",
  cell-settings: sequence (structure (cell-type: the symbol time-of-day,
      background-color: the symbol salmon,
```

```
      cell-value: structure (selected-year: 2001,
        selected-month: 2,
        selected-date: 17,
        selected-hour: 6,
        selected-minute: 15,
        selected-second: 5)),
    structure (cell-type: the symbol duration,
      background-color: the symbol floral-white,
      cell-value: structure (number-of-weeks: 5,
        number-of-days: 3,
        number-of-hours: 22,
        number-of-minutes: 19,
        number-of-seconds: 55)),
    structure (text-color: the symbol black,
      background-color: the symbol khaki,
      cell-value: structure (text-value: "Gaius")),
    structure (cell-type: the symbol check-box,
      background-color: the symbol green-yellow,
      cell-value: structure (selected: true,
        text-value: "I Don't Know")),
    structure (cell-type: the symbol combo-box,
      background-color: the symbol cyan,
      cell-value: structure (text-sequence: sequence ("Chico",
          "Groucho",
          "Harpo"),
        selected: "Chico")),
    structure (cell-type: the symbol color-picker,
      background-color: the symbol light-blue,
      cell-value: structure (selected: the symbol light-blue)),
    structure (cell-type: the symbol calendar,
      background-color: the symbol violet,
      cell-value: structure (selected-year: 2015,
        selected-month: 6,
        selected-date: 18)))),
structure (text-value: "Row 4",
  cell-settings: sequence (structure (cell-type: the symbol calendar,
      background-color: the symbol coral,
      cell-value: structure (selected-year: 2005,
        selected-month: 12,
        selected-date: 2)),
    structure (cell-type: the symbol time-of-day,
      background-color: the symbol linen,
      cell-value: structure (selected-hour: 18,
        selected-minute: 36,
        selected-second: 54)),
    structure (cell-type: the symbol duration,
      background-color: the symbol light-goldenrod,
      cell-value: structure (number-of-weeks: 0,
        number-of-days: 0,
        number-of-hours: 8,
        number-of-minutes: 14,
```

```
        number-of-seconds: 23)),
      structure (text-color: the symbol black,
        background-color: the symbol pale-green,
        cell-value: structure (text-value: "Claudius")),
      structure (cell-type: the symbol check-box,
        background-color: the symbol aquamarine,
        cell-value: structure (selected: false,
          text-value: "Why")),
      structure (cell-type: the symbol combo-box,
        background-color: the symbol light-steel-blue,
        cell-value: structure (text-sequence: sequence ("Athos",
            "Porthos",
            "Aramis"),
          selected: "Aramis")),
      structure (cell-type: the symbol color-picker,
        background-color: the symbol magenta,
        cell-value: structure (selected: the symbol magenta)))),
  structure (text-value: "Row 5",
    cell-settings: sequence (structure (cell-type: the symbol color-picker,
        text-color: the symbol gold,
        background-color: the symbol red,
        cell-value: structure (selected: the symbol red)),
      structure (cell-type: the symbol calendar,
        background-color: the symbol antique-white,
        cell-value: structure (selected-year: 2006,
          selected-month: 10,
          selected-date: 14)),
      structure (cell-type: the symbol time-of-day,
        background-color: the symbol gold,
        show-date: true,
        use-24-hour-time: true,
        cell-value: structure (selected-year: 2007,
          selected-month: 10,
          selected-date: 2,
          selected-hour: 14,
          selected-minute: 16,
          selected-second: 18)),
      structure (cell-type: the symbol duration,
        background-color: the symbol green,
        cell-value: structure (number-of-weeks: 0,
          number-of-days: 4,
          number-of-hours: 2,
          number-of-minutes: 3,
          number-of-seconds: 10)),
      structure (background-color: the symbol turquoise,
        cell-value: structure (text-value: "Nero")),
      structure (cell-type: the symbol check-box,
        background-color: the symbol sky-blue,
        cell-value: structure (selected: true,
          text-value: "Because")),
      structure (cell-type: the symbol combo-box,
```

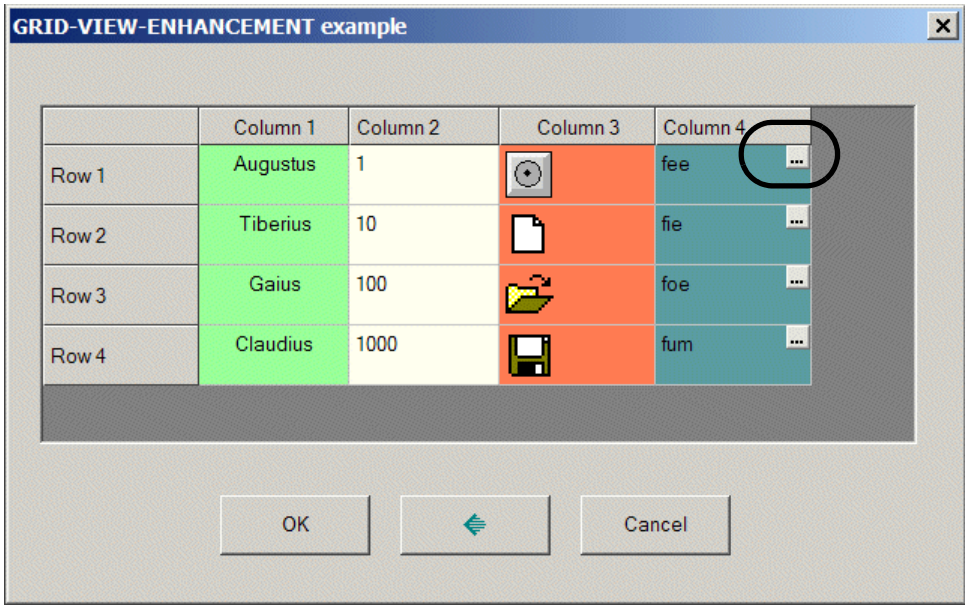
```
      text-color: the symbol gold,
      background-color: the symbol medium-orchid,
      cell-value: structure (text-sequence: sequence ("Pompey",
         "Crassus",
         "Sulla"),
        selected: "Crassus")))),
 structure (text-value: "Row 6",
  cell-settings: sequence (structure (cell-type: the symbol combo-box,
      text-color: the symbol gold,
      background-color: the symbol indian-red,
      cell-value: structure (text-sequence: sequence ("Socrates",
         "Plato",
         "Aristotle"),
        selected: "Aristotle")),
    structure (cell-type: the symbol color-picker,
      background-color: the symbol wheat,
      cell-value: structure (selected: the symbol wheat)),
    structure (cell-type: the symbol calendar,
      background-color: the symbol orange,
      cell-value: structure (selected-year: 2007,
        selected-month: 2,
        selected-date: 7)),
    structure (cell-type: the symbol time-of-day,
      background-color: the symbol lime-green,
      cell-value: structure (selected-hour: 5,
        selected-minute: 6,
        selected-second: 7)),
    structure (cell-type: the symbol duration,
      text-color: the symbol black,
      background-color: the symbol medium-aquamarine,
      cell-value: structure (number-of-weeks: 3,
        number-of-days: 6,
        number-of-hours: 12,
        number-of-minutes: 24,
        number-of-seconds: 36)),
    structure (text-color: the symbol gold,
      background-color: the symbol blue,
      cell-value: structure (text-value: "Galba")),
    structure (cell-type: the symbol check-box,
      text-color: the symbol gold,
      background-color: the symbol violet-red,
      cell-value: structure (selected: false,
        text-value: "Today")))),
 structure (text-value: "Row 7",
  cell-settings: sequence (structure (cell-type: the symbol check-box,
      text-color: the symbol gold,
      background-color: the symbol brown,
      cell-value: structure (selected: true,
        text-value: "Tomorrow")),
    structure (cell-type: the symbol combo-box,
      background-color: the symbol tan,
```

```
          cell-value: structure (text-sequence: sequence ("Adam",
             "Hoss",
             "Little Joe"),
           selected: "Adam")),
        structure (cell-type: the symbol color-picker,
          background-color: the symbol goldenrod,
          cell-value: structure (selected: the symbol goldenrod)),
        structure (cell-type: the symbol calendar,
          text-color: the symbol gold,
          background-color: the symbol forest-green,
          cell-value: structure (selected-year: 2009,
            selected-month: 7,
            selected-date: 6)),
        structure (cell-type: the symbol time-of-day,
          text-color: the symbol gold,
          background-color: the symbol cadet-blue,
          show-date: true,
          show-time: false,
          cell-value: structure (selected-year: 1993,
            selected-month: 6,
            selected-date: 19,
            selected-hour: 8,
            selected-minute: 13,
            selected-second: 22)),
        structure (cell-type: the symbol duration,
          text-color: the symbol gold,
          background-color: the symbol slate-blue,
          cell-value: structure (number-of-weeks: 1,
            number-of-days: 2,
            number-of-hours: 4,
            number-of-minutes: 8,
            number-of-seconds: 16)),
        structure (text-color: the symbol gold,
          background-color: the symbol purple,
          cell-value: structure (text-value: "Otho")))))),
  structure (control-type: the symbol push-button,
    control-id: the symbol ok-button,
    height: 20,
    width: 50,
    left: 70,
    top: 150,
    response-action: the symbol ok,
    control-value: structure (text-value: "OK")),
  structure (control-type: the symbol push-button,
    control-id: the symbol apply-button-with-icon,
    height: 20,
    width: 50,
    left: 130,
    top: 150,
    response-action: the symbol respond,
    control-value: structure (icon: the symbol gms-gensym-icon)),
```

```
structure (control-type: the symbol push-button,
  control-id: the symbol cancel-button,
  height: 20,
  width: 50,
  left: 190,
  top: 150,
  response-action: the symbol cancel,
  control-value: structure (text-value: "Cancel")))
```

Launching a Custom Cell Editor

The grid-view control allows you to place an ellipsis button (...) in a text-box cell, which displays the value in a text box that the user cannot edit. When the user clicks the button, an event notification is sent to the dialog update callback, which you can use, for example, to launch a custom cell editor.



For example, this code fragment shows a portion of the rows specification for the grid-view shows above, where the cell in Column 4 and Row 1 has an ellipsis button.

```
rows: sequence
(structure
    (text-value: "Row 1",
    height: 40,
    cell-settings: sequence
        (structure
            (cell-type: the symbol text-box,
            height: 40,
            text-color: the symbol black,
            background-color: the symbol pale-green,
            cell-value: structure
                (text-value: "Augustus")),
                structure
                    (cell-type: the symbol spinner,
                    background-color: the symbol ivory,
                    cell-value: structure (current-value: 1,
                    low-value: 0,
                    high-value: 5,
                    increment: 1)),
                structure
                    (cell-type: the symbol image,
                    alignment: the symbol center,
                    background-color: the symbol coral,
                    cell-value: structure (icon: the symbol connection-post)),
                structure
                    (cell-type: the symbol text-box,
                    background-color: the symbol cadet-blue,
                    ellipsis-button: true,
                    cell-value: structure (text-value: "fee"))))
```

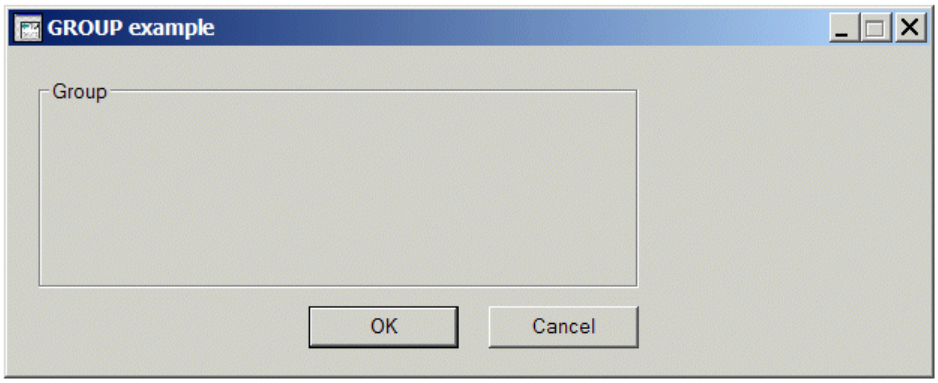
When the user clicks the ellipsis button, the update callback control-value structure includes the following new information at the end of the structure:

```
NOTIFICATION: structure
    (ROW: 0, COLUMN: 0, REASON: the symbol ELLIPSIS-BUTTON)
```

You can use this information in the dialog-update-callback of the dialog specification to launch a text editor, for example, by calling g2-ui-launch-editor.

group

The group control provides a logical grouping and associated label:



Specific Attributes for Group Control

Attribute Name	Type	Required	Default	Description
control-value	structure	yes	N/A	A structure that specifies the group label: structure (text-value: *text*)

Example: Dialog Specification for Group Control

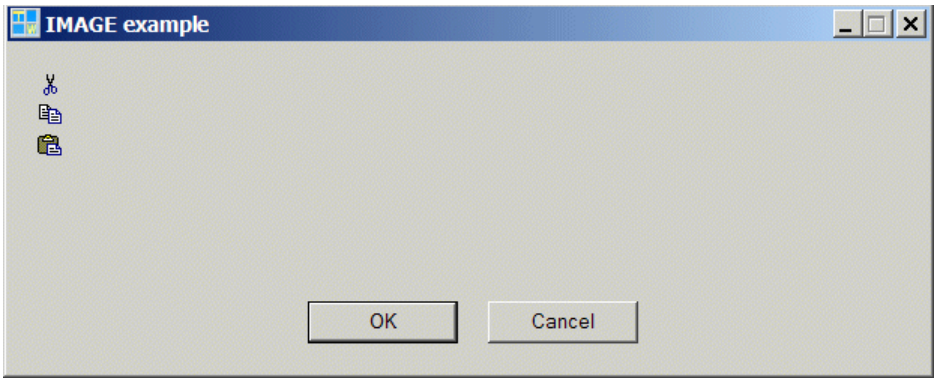
Here is the dialog specification for the group control portion of the dialog above:

```
structure
    (dialog-title: "GROUP example",
    dialog-width: 310,
    dialog-height: 124,
    dialog-is-modal: true,
    dialog-update-callback: dialog-examples-update-callback,
    dialog-dismissed-callback: dialog-examples-dismissed-callback,
    components:
        sequence
            (structure
                (control-type: the symbol push-button,
                . . .
                control-value: structure (text-value: "OK")),
            structure
                (control-type: the symbol push-button,
                . . .
                control-value: structure (text-value: "Cancel")),
            structure
                (control-type: the symbol group,
                control-id: 3,
                height: 70,
                width: 200,
                left: 10,
                top: 10,
                response-action: the symbol ignore,
                control-value: structure (text-value: "Group"))))
```

image

The image control allows you to display an icon in a dialog.

Here is a custom dialog with three image controls that uses the built-in GMS icons:



Specific Attributes for Image Control

Attribute Name	Type	Required	Default	Description
control-value	structure	yes	structure()	A structure that specifies a symbol that is the icon to display: structure (icon: *item-or-symbol*)

When specifying the icon, you can provide a G2 class name, an item, or one of the following GMS icons, as a symbol:

gms-cut-icon
gms-copy-icon
gms-paste-icon
gms-undo-icon
gms-redo-icon
gms-delete-icon
gms-file-icon
gms-folder-icon
gms-save-icon
gms-print-preview-icon
gms-properties-icon
gms-help-icon
gms-find-icon
gms-replace-icon

gms-print-icon
gms-gensym-icon (This is the aquamarine gensym logo.)
gms-binoculars-icon
gms-save-all-icon

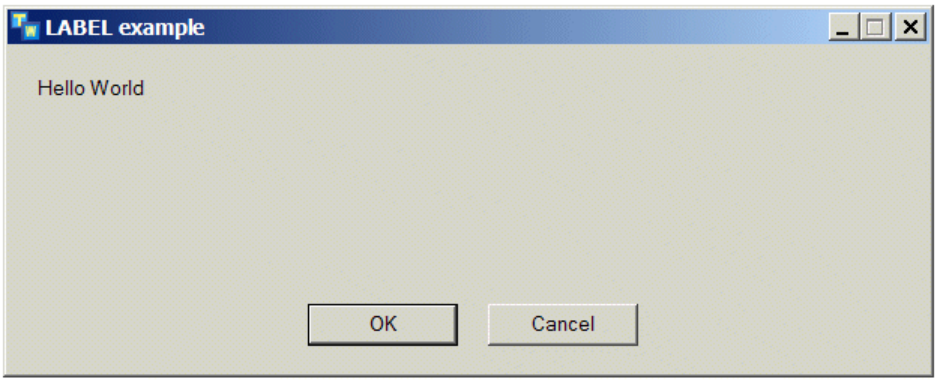
Example: Dialog Specification for Image Control

Here is the dialog specification for one of the image controls in the dialog above:

```
structure
    (control-type: the symbol image,
    control-id: 3,
    width: 90,
    height: 15,
    left: 10,
    top: 10,
    response-action: the symbol ignore,
    control-value: structure (icon: the symbol gms-cut-icon))
```

label

The label control adds a static label to a dialog:



Specific Attributes for Label Control

Attribute Name	Type	Required	Default	Description
alignment	symbol	no	left	The alignment of the label, as one of these symbols: left, right, and center. These are equivalent to ss-left, ss-right and ss-center win32 styles.
control-value	structure	yes	N/A	A structure that specifies the initial text of the label: structure (text-value: *text*)

Example: Dialog Specification for Label Control

Here is the dialog specification for the label portion of the dialog above, which provides a static label:

```
structure
    (dialog-title: "LABEL example",
    dialog-width: 310,
    dialog-height: 124,
    dialog-is-modal: true,
    components:
        sequence
            (structure
                (control-type: the symbol push-button,
                . . .
                control-value: structure (text-value: "OK")),
            structure
                (control-type: the symbol push-button,
                . . .
                control-value: structure (text-value: "Cancel")),
            structure
                (control-type: the symbol label,
                control-id: 3,
                width: 50,
                height: 15,
                left: 10,
                top: 10,
                response-action: the symbol ignore,
                control-value: structure (text-value: "Hello World"))))
```

Example: Updating a Label

Use the replace control action to change the text of a label control:

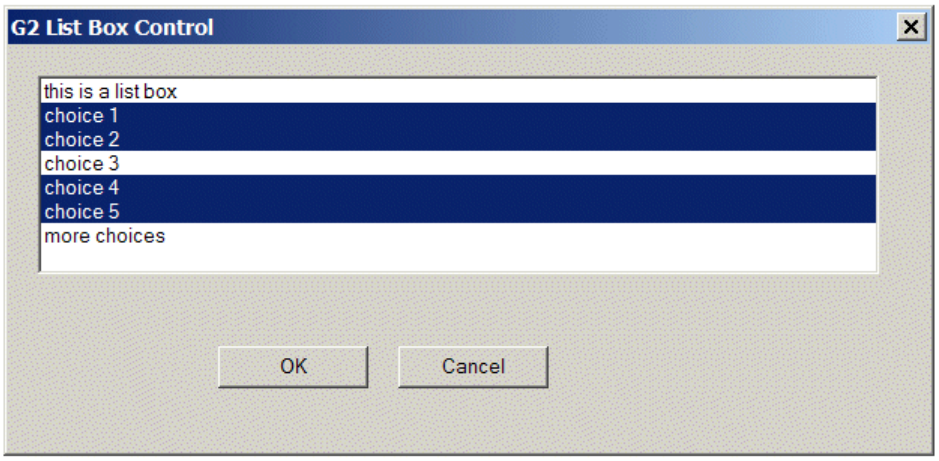
```
call g2-ui-modify-custom-dialog
    (dialog-id,
    sequence
        (structure
            (control-action: the symbol replace,
            control-id: 3,
            control-value: structure(text-value: "New label"))),
    window);
```

list-box

The list-box control supports three modes:

- Single selection allows the user to select a single element in the list box only. Selecting an element deselects the currently selected element.
- Extended selection allows the user to extend the selection to include multiple entries. Clicking an element deselects the currently selected elements. Holding down the SHIFT key and clicking an element selects a series of elements. Holding down the CTRL key and clicking an element adds or removes individual elements to or from the selection.
- Multiple selection allows the user to select of any number of elements in the list box by single clicking the element. Clicking a selected element deselects it.

Here is a dialog with a multiple selection list box with choices 1, 2, 4, and 5 selected:



Note When using the list-box control for G2 Version 8.1 Rev. 0 or later, you must use Telewindows Version 8.1 Rev. 0 or later.

Specific Attributes for List-Box Control

Attribute Name	Type	Required	Default	Description
extended-selection	truth-value	no	false	When true, enables extended selection and disables multiple selection, if it was enabled. When false, disables extended selection and does nothing to multiple selection.
multiple-selection	truth-value	no	false	When true, enables multiple selection and disables extended selection, if it was enabled. When false, disables both multiple selection and extended selection.
single-selection	truth-value	no	true	When true, disables both extended selection and multiple selection. When false, enables extended selection and disables multiple selection, if it was enabled.

Attribute Name	Type	Required	Default	Description
sort-list	truth-value	no	false	When true, sorts the strings alphabetically. When false, displays the strings in the order supplied in the text-sequence of the control-value.
control-value	structure	yes	N/A	A structure that specifies the sequence of text values for each element in the list box and the initially selected values: structure (text-sequence: sequence (*text*[, ...]), selected: sequence (*text*[, ...]))

As long as the selected attribute of the control-value is a sequence with zero or more members and each member is a text, no errors are signalled. Thus, you must ensure that all elements in the selected sequence also appear in the text-sequence.

If the list-box is operating in extended selection or multiple selection mode, then the user may select any number of choices, including zero. If it is operating in single selection mode, then only one choice can be selected. In single-selection mode, the selected attribute must still be a sequence.

If a sequence with more than one element is supplied for the selected attribute in single-selection list-box, the first member of the sequence is the initially selected value.

For a single-selection list-box, if no selected attribute is supplied or if an empty sequence is specified, or if a sequence where the first member does not actually appear in the list-box is supplied, then the first entry in the list is the initially selected value.

The return value of a list box is:

```
structure
(control-id: control-id,
control-value:
    structure
        (text-sequence: sequence (text[, ...]),
        selected: sequence (text[, ...])))
```

Example: Dialog Specification for Multiple-Selection List-Box Control

For example, here is the control structure for a multiple-selection list box:

```
structure
    (control-type: the symbol LIST-BOX,
    control-id: the symbol EXAMPLE-LIST-BOX,
    multiple-selection: true,
    control-value:
        structure
            (selected: sequence ("choice 1", "choice 2", "choice 4"),
            text-sequence:
                sequence ("this is a list box",
                          "choice 1",
                          "choice 2",
                          "choice 3",
                          "choice 4",
                          "choice 5",
                          "more choices")),
    response-action: the symbol RESPOND,
    height: 75,
    width: 280,
    left: 10,
    top: 10)
```

Example: Replacing an Element in a List-Box Control

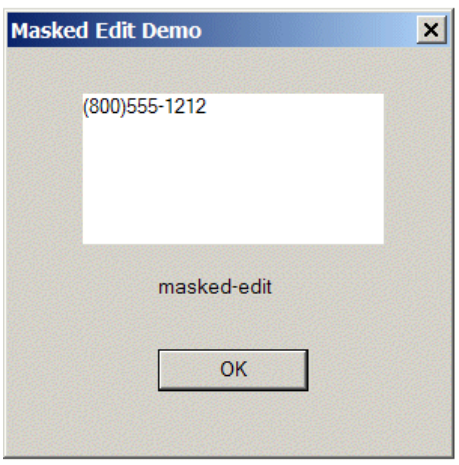
Use the add control action to add an element to a list-box control:

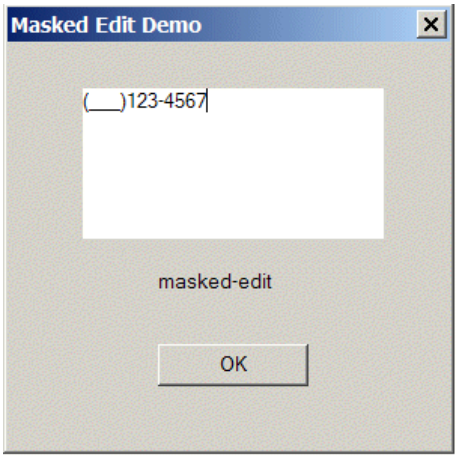
```
call g2-ui-modify-custom-dialog
    (dialog-id,
    sequence
        (structure
            (control-action: the symbol replace,
            control-id: 3,
            control-value: structure
                (text-sequence: sequence ("New 1", "New 2", "New 3"))))
    window);
```

masked-edit

The masked-edit control provides a text box that restricts what the user can enter. You specify the mask to be the allowable characters, for example, characters or numbers, as well as fixed characters, such as dashes or parentheses. You can also specify literal text to enter if the user presses the space bar.

Here is a custom dialog with a masked-edit control that restricts entry to a 10-digit phone number. The control specifies parentheses and dashes as fixed characters and underscores as the literal character to enter if the user inputs a space. The second dialog shows the result of entering three spaces and the numbers 1, 2, 3, 4, 5, 6, and 7. The cursor skips over the fixed characters.





Specific Attributes for Masked-Edit Control

Attribute Name	Type	Required	Default	Description
control-value	structure	yes	structure()	A structure that specifies the current value, the mask, and the literal text for the masked edit box: structure (current-value: *text*, mask: *text*, literal: *text*)

In the control-value, the values of each of the following attributes should have the same length:

- current-value determines the initial value when the control is displayed, as well as the character to enter when the user presses the Backspace or Delete key.
- mask determines the position and type of the values the user can enter and any fixed text. Use these characters to specify the type of allowable text:

  0 — Numeric (0-9)

  9 — Numeric (0-9) or space (' ')

  # — Numeric (0-9) or space (' ') or ('+') or ('+')

  L — Alpha (a-Z)

  ? — Alpha (a-Z) or space (' ')

  A — Alpha numeric (0-9 and a-Z)

  a — Alpha numeric (0-9 and a-Z) or space (' ')

  & — All print character only

  H — Hex digit (0-9 and A-F)

  X — Hex digit (0-9 and A-F) and space (' ')

  > — Forces characters to upper case (A-Z)

  < — Forces characters to lower case (a-z)

  All other characters are considered fixed text.
- literal determines the literal format for the data, where an underscore ("_") indicates where the user can enter data.

Example: Dialog Specification for Masked-Edit Control

Here is the dialog specification for the masked-edit control portion of the dialog above:

```
structure
    (dialog-width: 150,
    dialog-height: 150,
    dialog-title: "Masked Edit Demo",
    dialog-is-modal: false,
    dialog-update-callback: dialog-has-been-updated,
    dialog-dismissed-callback: static-dialog-has-been-dismissed,
    components:
        sequence
            (structure
                (control-type: the symbol PUSH-BUTTON,
                . . .
                control-value: structure (text-value: "OK")),
            structure
                (control-type: the symbol MASKED-EDIT,
                control-id: the symbol my-masked-edit,
                control-value:
                    structure
                        (current-value: "(800)555-1212",
                        mask: "(000)000-0000",
                        literal: "(___)___-____"),
                response-action: the symbol RESPOND,
                height: 50,
                width: 100,
                left: 25,
                top: 15),
            structure
                (control-type: the symbol LABEL,
                . . .
                control-value: structure (text-value: "masked-edit"))))
```

progress-bar

The progress-bar control shows progress in specified increments along a horizontal or vertical progress bar. The progress can show smooth or discrete steps.

Smooth progress-bar.

Vertical progress-bar, discrete increments.

Specific Attributes for Progress-Bar Control

Attribute Name	Type	Required	Default	Description
is-smooth	truth-value	no	false	Whether the progress bar shows increments as smooth or discrete steps. By default, the progress appears in discrete increments.
is-vertical	truth-value	no	false	Whether the progress bar is vertical. By default, the progress bar is horizontal.
control-value	structure	yes	N/A	A structure that specifies the value range and current value: structure (low-value: *integer*, high-value: *integer*, current-value: *integer*)

Example: Dialog Specification for Progress Bar

This example shows the smooth progress bar:

```
structure
    (control-type: the symbol PROGRESS-BAR,
    control-id: the symbol MY-PROGRESS-BAR,
    control-value:
        structure
            (low-value: 0,
            high-value: 100,
            current-value: 50),
    is-smooth: true,
    . . .)
```

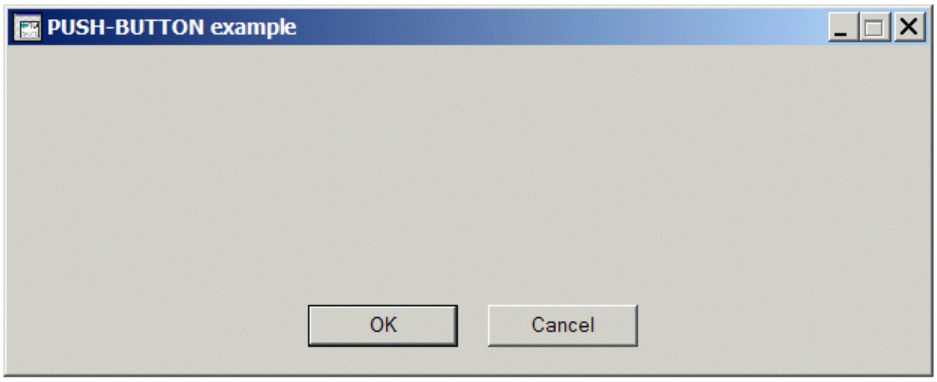
Example: Animating the Progress Bar

This code fragment shows how to animate the progress bar by calling g2-ui-modify-custom-dialog:

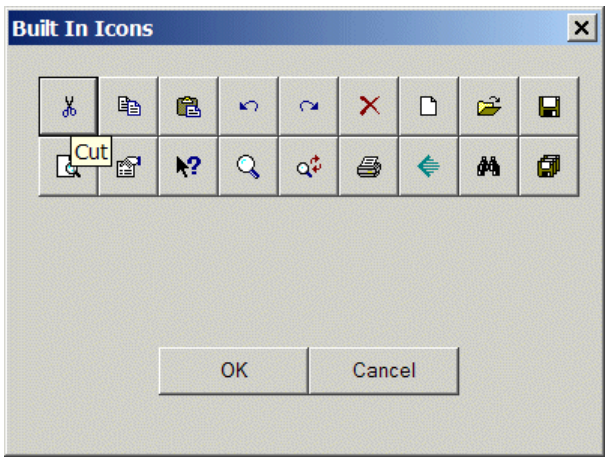
```
h: integer;
n: integer;

h = the dialog-id of dialog;
n = 50;
repeat
    call g2-ui-modify-custom-dialog
        (h, sequence
                (structure
                    (control-id: the symbol MY-PROGRESS-BAR,
                    control-action: the symbol REPLACE,
                    control-value: structure (current-value: n))),
        window);
    n = n + 10;
    if n > 100 then n = 0;
    wait for 1 second;
end
```

push-button

The push-button control provides a button that takes some type of action, such as accepting or dismissing a dialog:



The push-button control also provides an option for specifying an icon instead of text. When specifying an icon, the text value is used as a tool tip for the icon. For example:



Note Currently, you can use only 16x16 icons.

Specific Attributes for Push-Button Control

Attribute Name	Type	Required	Default	Description
is-default	truth-value	no	false	Whether the push button is the default push button for the dialog. **Note:** This attribute is currently not supported.
control-value	structure	yes	N/A	A structure that specifies a text value for the label, or an icon and text value to use as a tool tip. Properties control-background-color and text-font-color also specify the background and font color of push-button control: structure (text-value: *text*, icon: *item-or-symbol*, control-background-color: *text*, text-font-color: *text*)

When specifying an icon, you can specify a G2 class name, an item, or a built-in GMS icon. For a list of built-in GMS icons, see image.

Note The control-value also accepts the icon-name attribute; however, this attribute is deprecated. You should use the icon attribute instead.

Example: Dialog Specification for Push-Button Control

Here is the dialog specification for the dialog above, which provides OK and Cancel buttons:

```
structure
    (dialog-title: "PUSH-BUTTON example",
    dialog-width: 310,
    dialog-height: 124,
    dialog-is-modal: true,
    dialog-update-callback: dialog-examples-update-callback,
    dialog-dismissed-callback: dialog-examples-dismissed-callback,
    components:
        sequence
            (structure
                (control-type: the symbol push-button,
                control-id: 1,
                height: 14,
                width: 50,
                left: 100,
                top: 86,
                response-action: the symbol ok,
                control-value: structure (text-value: "OK")),
            structure (control-type: the symbol push-button,
                control-id: 2,
                height: 14,
                width: 50,
                left: 160,
                top: 86,
                response-action: the symbol cancel,
                control-value: structure (text-value: "Cancel"))))
```

Example: Disabling a Push Button

Use the disable control action to disable a push-button control:

```
call g2-ui-modify-custom-dialog
    (dialog-id,
    sequence
        (structure
            (control-action: the symbol disable,
            control-id: 1)),
    window);
```

Example: Enabling a Push Button

Use the enable control action to enable a push-button control:

```
call g2-ui-modify-custom-dialog
     (dialog-id,
     sequence
          (structure
               (control-action: the symbol enable,
               control-id: 1)),
     window);
```

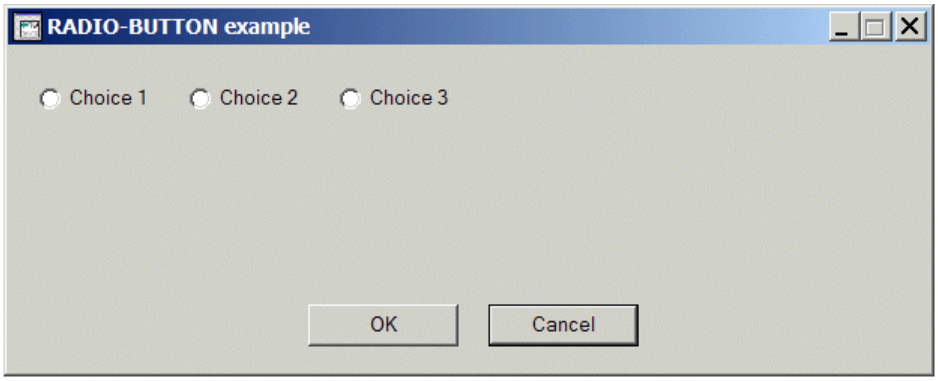
Example: Specifying an Icon for a Push Button

Here is the dialog specification for a push button that uses an icon:

```
structure
     (control-type: the symbol PUSH-BUTTON,
     control-id: the symbol CUT-BUTTON,
     height: 20,
     width: 20,
     left: 10,
     top: 10,
     response-action: the symbol RESPOND,
     is-enabled: true,
     control-value:
          structure
               (icon: the symbol GMS-CUT-ICON,
               text-value: "Cut")
```

radio-button

A radio-button control provides a set of choices from which the user can choose a single value:



Specific Attributes for Radio-Button Control

Attribute Name	Type	Required	Default	Description
control-value	structure	no	N/A	A structure that specifies the label for the radio button and whether the button is initially selected: structure (text-value: *text*, selected: *truth-value*)

Example: Dialog Specification for Radio-Button Control

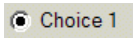
Here is the dialog specification for the radio-button control portion of the dialog above, which provides three radio buttons:

```
structure
    (dialog-title: "RADIO-BUTTON example",
    dialog-width: 310,
    dialog-height: 124,
    dialog-is-modal: true,
    dialog-update-callback: dialog-examples-update-callback,
    dialog-dismissed-callback: dialog-examples-dismissed-callback,
    components:
        sequence
            (structure
                (control-type: the symbol push-button,
                . . .
                control-value: structure (text-value: "OK")),
            structure
                (control-type: the symbol push-button,
                . . .
                control-value: structure (text-value: "Cancel")),
            structure
                (control-type: the symbol radio-button,
                control-id: 3,
                height: 15,
                width: 50,
                left: 10,
                top: 10,
                control-value: structure (text-value: "Choice 1"),
                response-action: the symbol ignore),
            structure
                (control-type: the symbol radio-button,
                control-id: 4,
                height: 15,
                width: 50,
                left: 60,
                top: 10,
                control-value: structure (text-value: "Choice 2"),
                response-action: the symbol ignore),
            structure
                (control-type: the symbol radio-button,
                control-id: 5,
                height: 15,
                width: 50,
                left: 110,
                top: 10,
                control-value: structure (text-value: "Choice 3"),
                response-action: the symbol ignore)))
```

Example: Checking a Radio Button

Use the check control action to check a radio-button control:

```
call g2-ui-modify-custom-dialog
     (dialog-id,
     sequence
          (structure
               (control-action: the symbol check,
               control-id: 3)),
     window);
```



slider

The slider control provides a slider with smooth increments that you can use to set a value.

Note The slider control does not trigger the update callback when the user changes the value with the keyboard. It only triggers the update callback when the user releases the mouse button.

Specific Attributes for Slider Control

Attribute Name	Type	Required	Default	Description
control-value	structure	yes	the current date	A structure that specifies the value range, current value, and increment: structure (low-value: *integer*, high-value: *integer*, current-value: *integer*, increment: *integer*)

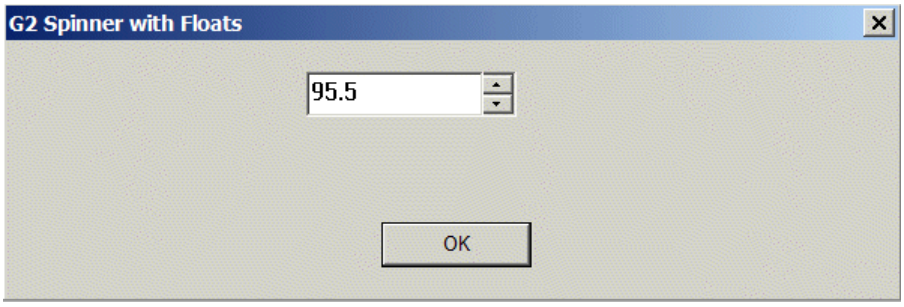
Example: Dialog Specification for Slider Control

Here is the specification for the slider control:

```
structure
    (control-type: the symbol SLIDER,
    control-id: the symbol MY-SLIDER,
    control-value:
        structure
            (low-value: 0,
            high-value: 1000,
            current-value: 750,
            increment: 5),
    . . .)
```

spinner

The spinner control provides text box with a spin control that increments the value by a given quantity between a low and high value:



Specific Attributes for Spinner Control

Attribute Name	Type	Required	Default	Description
alignment	symbol	no	right	The alignment of the arrows relative to the text box, as one of these symbols: left and right. These are equivalent to the ss-left and ss-right win32 styles.
orientation	symbol	no	Windows default	The orientation of the arrows, as one of these symbols: vertical or horizontal.
arrow-keys	truth-value	no	false	Whether you can use the up and down arrow keys on the keyboard to increment the value of the control, in addition to the arrow buttons on the control.

Attribute Name	Type	Required	Default	Description
is-integer	truth-value	no	false	Whether the spinner accepts only integer values, regardless of whether the low-value, high-value, and increment attributes of the control-value structure are floating point numbers or integers.
control-value	structure	yes	structure (current-value: 50, low-value: 0, high-value: 100, increment: 1)	A structure that specifies the default value, low and high values, value for incrementing the spinner, and precision: structure (current-value: *quantity*, low-value: *quantity,* high-value: *quantity*, increment: *quantity*, precision: *ddd.dddd-format*)

The minimum value for increment is .0001.

The precision is specified as a *ddd.dddd-format* expression, which is a symbol that determines the number of decimal places to the right and left of the decimal point. For example, ddd.dd formats a floating point number to the hundredths decimal place.

Example: Dialog Specification for Spinner Control

Here is the dialog specification for the spinner control portion of the dialog above, which specifies values between 0.5 and 300.5 in increments of 1.0, using precision:

```
structure (control-type: the symbol spinner,
    control-id: the symbol my-spinner,
    height: 15,
    width: 70,
    left: 100,
    top: 10,
    response-action: the symbol respond,
    control-value: structure (current-value: 95.5,
        low-value: 0.5,
        high-value: 300.5,
        increment: 1.0,
        precision: the symbol dd.ddd))
```

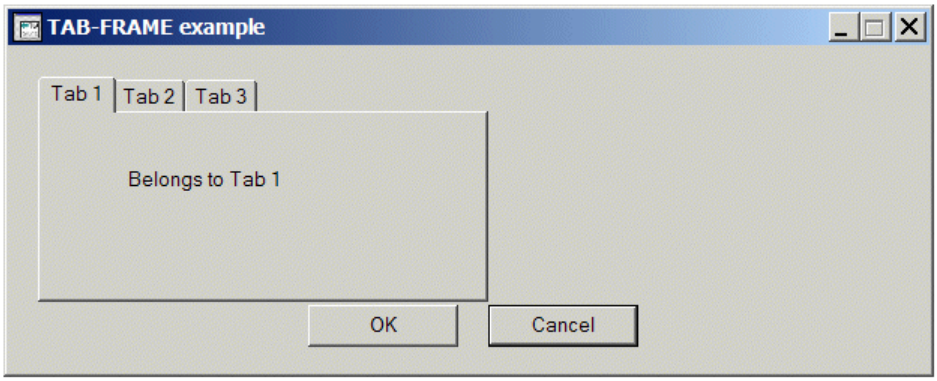
Example: Updating a Spinner Control

Use the replace control action to update the value of a spinner control:

```
call g2-ui-modify-custom-dialog
    (dialog-id,
    sequence
        (structure
            (control-action: the symbol replace,
            control-id: 3,
            control-value: structure (text-value: "15")),
    window);
```

tab-frame

The tab-frame control provides tab pages in a dialog, where each tab has a separate label. You can add child controls to a particular tab by using the parent-control-text and parent-control-id attributes.



Specific Attributes for Tab-Frame Control

Attribute Name	Type	Required	Default	Description
control-value	structure	yes	N/A	A structure that specifies the tab labels, tab icons, initially selected tab, and tab position: structure (tab-labels: sequence (*text*,...), tab-icons: sequence (*icon*, ...), selected-tab: *text*, tab-position: top | left | right | bottom)

In the control-value structure:

- tab-labels is a sequence of labels for each tab, each of which must be unique. If the sequence is empty, a tab-frame with no tabs is created.
- tab-icons is a sequence of icons, where the number of elements in the sequence must match the number of elements in the tab-labels sequence. To specify no icon for a particular tab, use false. The icon can be a G2 class name, an item, or a built-in GMS icon. For a list of built-in GMS icons, see image.

- selected-tab is the initially selected tab.
- tab-position is the position of the tabs in the overall frame, where the default value is top.

Example: Dialog Specification for Tab-Frame Control

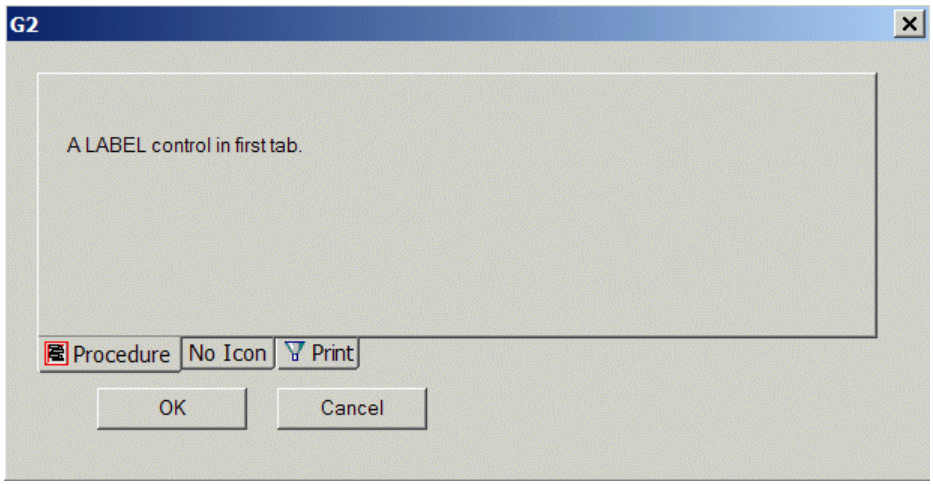
Here is the dialog specification for the tab-frame control portion of the dialog above, which specifies three tabs. Notice the use of parent-control-id and parent-control-text in the label controls to identify the tab on which to place the control.

```
structure (dialog-title: "TAB-FRAME example",
    dialog-width: 310,
    dialog-height: 124,
    dialog-is-modal: true,
    dialog-update-callback: dialog-examples-update-callback,
    dialog-dismissed-callback: dialog-examples-dismissed-callback,
    components:
        sequence
            (structure
                (control-type: the symbol push-button,
                . . .
                control-value: structure (text-value: "OK")),
            structure
                (control-type: the symbol push-button,
                . . .
                control-value: structure (text-value: "Cancel")),
            structure
                (control-type: the symbol tab-frame,
                control-id: 3,
                height: 75,
                width: 150,
                left: 10,
                top: 10,
                response-action: the symbol ignore,
                control-value:
                    structure (tab-labels: sequence ("Tab 1", "Tab 2", "Tab 3"))),
            structure
                (control-type: the symbol label,
                control-id: 4,
                parent-control-id: 3,
                parent-control-text: "Tab 1",
                height: 15,
                width: 70,
                left: 40,
                top: 40,
                response-action: the symbol ignore,
                control-value: structure (text-value: "Belongs to Tab 1")),
```

```
structure
    (control-type: the symbol label,
    . . .
    control-value: structure (text-value: "Belongs to Tab 2")),
structure
    (control-type: the symbol label,
    . . .
    control-value: structure (text-value: "Belongs to Tab 3"))))
```

Example: Specifying Tab Position and Icons

Here is a tab-frame in a dialog with the tabs at the bottom and icons specified for two of the three tabs:



Here is the procedure used to create the dialog:

```
create-tab-frame(side: symbol, Win: class g2-window)
components: sequence = sequence(
   structure (control-type: the symbol push-button,
    control-id: 1, height: 14, width: 50, left: 30, top: 115,
    response-action: the symbol ok,
    control-value: structure (text-value: "OK")),
  structure (control-type: the symbol push-button,
    control-id: 2, height: 14, width: 50, left: 90, top: 115,
    response-action: the symbol cancel,
    control-value: structure (text-value: "Cancel")),
  structure (control-type: the symbol tab-frame,
    control-id: 3, height: 100, width: 280, left: 10, top: 10,
    response-action: the symbol ignore,
    control-value: structure (tab-labels: sequence ("Procedure", "No Icon", "Print"),
                              tab-icons: sequence (this procedure, false, the symbol
                                   GMS-FUNNEL-ICON),
                              tab-position: side)),
  structure (control-type: the symbol label,
    control-id: the symbol L1,  parent-control-id: 3, parent-control-text: "Procedure",
    control-value: structure (text-value: "A LABEL control in first tab."),
    height: 40,  width: 250,  left: 20,  top: 30),
    structure (control-type: the symbol label,
    control-id: the symbol L2,  parent-control-id: 3, parent-control-text: "No Icon",
    control-value: structure (text-value: "A LABEL control in second tab."),
    height: 40,  width: 250,  left: 20,  top: 30),
 structure (control-type: the symbol label,
    control-id: the symbol L3,  parent-control-id: 3, parent-control-text: "Print",
    control-value: structure (text-value: "A LABEL control in third tab."),
    height: 40,  width: 250,  left: 20,  top: 30));
begin
    call g2-ui-post-custom-dialog(structure(dialog-width: 310, dialog-height: 160,
components: components), false, Win);
end
```

Example: Modifying the Selected Tab

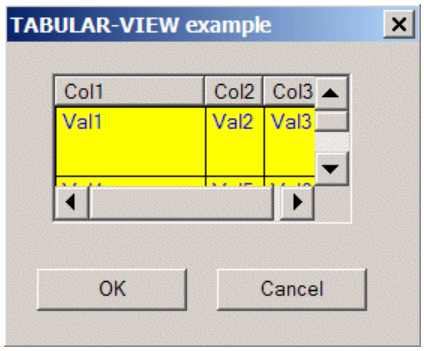
Use the selected-tab control action to update the selected tab of a tab-frame control:

```
call g2-ui-modify-custom-dialog
     (dialog-id,
     sequence
          (structure
               (control-action: the symbol selected-tab,
               control-id: the symbol my-tab-frame,
               control-value: "Tab2")),
     window);
```

```
sequence (structure (control-action: the symbol selected-tab,
     control-id: the symbol this-tab-frame,
     control-value: "Tab2"))
```

tabular-view

The tabular-view control provides support for presenting data in a grid:



Specific Attributes for Tabular-View Control

Attribute Name	Type	Required	Default	Description
background-color	symbol	no	Windows default	A G2 color for the default background color of all rows in the tabular view. The default value is the Windows background color.
text-color	symbol	no	Windows default	A G2 color for the default color of the text in the tabular view.
gridlines	truth-value	no	true	Whether the tabular view has grid lines.
allow-sort-rows	truth-value	no	false	Whether the client can sort the rows by clicking the column headers. When true, clicking the column header of a column that contains cells of type integer or quantity sorts the rows numerically.

Attribute Name	Type	Required	Default	Description
single-selection	truth-value	no	false	Whether the tabular view only allows the user to select a single row.
control-value	structure	yes	N/A	A structure that specifies the column headers, row specifications, and initially selected row: structure (columns: sequence (*column-structure*, ...), rows: sequence (*row-specification*, ...), selected-rows: *sequence*) See below for a description of the structure.

The control-value structure defines these attributes:

- columns is a sequence of *column-structure* specifications for each column, where *column-structure* is:

  structure
  (text-value: *text*,
  width: *integer-or-symbol*,
  icon: *item-or-symbol*,
  alignment: left | right | center)

  where:

  - text-value is the column header text.
  - width is the width of each column in the tabular view. The default width is the width of the widest value in the column (text-width); thus, this attribute is optional. The value can be an integer for the width of each column, specified in dialog units, or one of these symbols:
    - header-width — The width of the column text.
    - text-width — The width of the widest value of the cells in the column.

      To make the columns fit into a tabular view without a scrollbar, specify the overall width of the control to be two or three greater than the sum of the widths of the columns.
  - icon is a G2 class name, an item, or a built-in GMS icon. For a list of GMS icons, see image.
  - alignment is the alignment of the cell contents of each column in the tabular view.

- rows is a sequence of *row-structure* specifications for each row in the tabular view, where *row-specification* is:

  structure
  (logical-id: *integer* | *float* | *symbol* | *text*,
  text-color: *color-symbol*,
  background-color: *color-symbol*,
  row-values: sequence (*value-specification*, ...)

  where:

  - logical-id is an integer, float, symbol, or text that identifies the row. The logical-id defaults to an integer starting at 1.
  - text-color is the color for all text in the tabular view.
  - background-color is the color for the cell background for all cells in the row.
  - row-values is a sequence of value specifications for each row, where *value-specification* is:

    structure
    (text-value: *text*,
    icon: *item-or-symbol*)

    When specifying the icon, you can provide a G2 class name, an item, or a built-in GMS icon. For a list of GMS icons, see image.

- selected-rows is a sequence of logical-id values of the initially selected rows in the tabular view. This attribute is optional.

Note For a tabular view, the *new-value* argument to the dialog update callback returns only the selected-rows of the control-value structure, not the columns and rows. The reason is that the data cannot be changed interactively in the client, and there may be many rows of data. Additionally, the architecture is such that the model and view are kept separate, and a separate controller is responsible for handling callbacks from the view and updating the model. To modify the view, you use the g2-modify-custom-dialog system procedure. For an example, see the description of the Alert Queue Demo in Windows Dialogs.

Generic Dialog Callback

The dialog-generic-callback for a custom dialog, if specified, is called whenever the user clicks the mouse or presses a key in the tabular-view on the dialog, with the following arguments:

- *event*: LEFT-CLICK, MIDDLE-CLICK, RIGHT-CLICK, KEY-PRESS, or any combination of modifier keys or DOUBLE combined with LEFT-CLICK, MIDDLE-CLICK, or RIGHT-CLICK, for example: SHIFT+LEFT-CLICK, CONTROL+RIGHT-CLICK, ALT+LEFT-CLICK, DOUBLE+CONTROL+LEFT-CLICK
- *control-id*: integer
- *info*: structure
  (x: *integer*,
  y: *integer*,
  selected-rows: *sequence*,
  key: *symbol*,
  row: *integer*)

  where:

  - x and y are the x-y coordinates of the mouse click in the tabular view.
  - selected-rows is a sequence of the logical-id values of the selected rows in the tabular view.
  - key is the key that was pressed, if any.
  - row is the logical-id of the row with the focus, or if no row has the focus, the top-most row in the selection. The value for row is -1 if the selection is empty.
- *user-data*: value

For an example, see Example: Generic Dialog Callback.

Example: Dialog Specification for Tabular-View Control

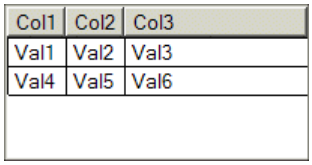
Here is the dialog specification for the tabular-view control portion of the dialog above, which specifies a tabular view with three columns and one row:

```
structure
    (dialog-title: "TABULAR-VIEW example",
    dialog-width: 310,
    dialog-height: 124,
    dialog-is-modal: true,
    dialog-update-callback: dialog-examples-update-callback,
    dialog-dismissed-callback: dialog-examples-dismissed-callback,
    components:
        sequence
            (structure
                (control-type: the symbol push-button,
                . . .
                control-value: structure (text-value: "OK")),
            structure
                (control-type: the symbol push-button,
                . . .
                control-value: structure (text-value: "Cancel")),
            structure
                (control-type: the symbol tabular-view,
                control-id: 3,
                height: 50,
                width: 100,
                left: 10,
                top: 10,
                response-action: the symbol ignore,
                control-value:
                    structure
                        (columns: sequence ("Col1", "Col2", "Col3"),
                        rows:
                            sequence
                                (structure
                                    (logical-id: 0,
                                    row-values:
                                        sequence
                                            (structure (text-value: "Val1"),
                                            structure (text-value: "Val2"),
                                            structure (text-value: "Val3"))))))))
```

Example: Adding a Row to a Tabular-View Control

Use the add-rows control action to add rows to a tabular-view control, where the control-value is a sequence of rows specified in the same way that rows are specified when creating the control. The following example adds a row to the tabular view.

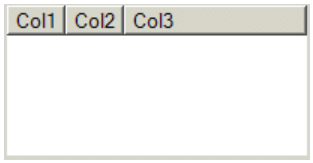
```
call g2-ui-modify-custom-dialog
    (dialog-id,
    sequence
        (structure
            (control-action: the symbol add-rows,
            control-id: 3,
            control-value:
                sequence
                    (structure (logical-id: 2,
                    row-values:
                        sequence
                            (structure (text-value: "Val4"),
                            structure (text-value: "Val5"),
                            structure (text-value: "Val6"))))),
    window);
```



Example: Deleting a Row from a Tabular-View Control

Use the remove-rows control action to remove rows from a tabular-view control, where the control-value is a sequence of logical-id values for the rows to be removed from the tabular-view. The following example removes two rows from the tabular view.

```
call g2-ui-modify-custom-dialog
    (dialog-id,
    sequence
        (structure
            (control-action: the symbol remove-rows,
            control-id: 3,
            control-value: sequence (0))),
    window);
```



Example: Replacing Rows in a Tabular View

Use the replace-rows control action to replace rows in a tabular-view control, where control-value is a sequence of logical-id values for the rows to be modified in the tabular-view. The following example replaces the values in the row with the logical ID of 1 and changes the colors:

```
call g2-ui-modify-custom-dialog
    (dialog-id,
    structure (control-action: the symbol replace-rows,
            control-id: 3,
            control-value:
                sequence
                    (structure
                        (logical-id: 1,
                        background-color: the symbol light-blue,
                        row-values: sequence (structure (text-value: "R1"),
                    structure
                        (text-value: "R2"),
                    structure
                        (text-value: "R3")))),
    window);
```

Example: Replacing Cells in a Tabular View

Use the replace-cells control action to replace cells in a tabular-view control, where control-value is a sequence of structures, where each structure contains the logical-id and index of the cells to be replaced in the tabular-view. The following example replaces the values in the cells logical ID of 1 and changes the colors:

```
call g2-ui-modify-custom-dialog
     (dialog-id,
     structure
          (control-action: the symbol replace-cells,
          control-id: 3,
          control-value:
               sequence
                    (structure
                         (logical-id: 0,
                         index: 1,
                         text-color: the symbol blue,
                         text-value: "New"))),
     window);
```

Example: Removing all Rows in a Tabular View

Use the remove-all-rows control action to remove all rows in a tabular-view control. It does not require any control-value.

```
call g2-ui-modify-custom-dialog
     (dialog-id,
     structure
          (control-action: the symbol remove-all-rows,
          control-id: 3),
     window);
```

Example: Removing all Cells in a Tabular View

Use the remove-all-cells control action to remove all cells in a tabular-view control. It does not require any control-value.

```
call g2-ui-modify-custom-dialog
     (dialog-id,
     structure
          (control-action: the symbol remove-all-selected-rows,
          control-id: 3),
     window);
```

Example: Adding Columns to a Tabular View

Use the add-columns control action to add columns to a tabular-view control, where the control-value is a sequence of column structures with this syntax:

structure
(column: *integer*,
text-value: *text*,
icon: *item-or-symbol*,
width: *integer*,
row-values: *sequence*)

where:

- column — The column number to add.
- text-value —The column header text.
- icon — An icon in place of the column header text, which you specify as a G2 class name, an item, or a built-in GMS icon. For a list of built-in GMS icons, see image.
- width — The width of the column in the tabular view. The default width is the width of the widest value in the column (text-width); thus, this attribute is optional. The value can be an integer for the width of each column, specified in dialog units, or one of these symbols:
  - header-width — The width of the column text.
  - text-width — The width of the widest value of the cells in the column.
- row-values — A sequence of cell structures used to populate the new column, where each structure has this syntax:

  structure
  (logical-id: *integer* | *float* | *symbol* | *text*,
  text-color: *symbol*,
  background-color: *symbol*,
  text-value: *text*
  icon: *item-or-symbol*)

  If row-values is omitted, all cells in the new column are blank.

The following example adds a column to a tabular view:

```
call g2-ui-modify-custom-dialog(dialog,
     sequence
          (structure
               (control-action: the symbol add-columns,
               control-id: the symbol table,
               control-value:
                    sequence
                         (structure
                              (column: pos,
                              icon: icon,
                              row-values: sequence
                                   (structure (logical-id: 0, text-value: "X-[tag]"),
                                   structure (logical-id: 1, text-value: "Y-[tag]"),
                                   structure (logical-id: 2, text-value: "Z-[tag]")))))),
Win);
```

Example: Specifying Cell Alignment

Here is an example of a tabular view with different cell alignment:

```
structure (control-type: the symbol tabular-view,
  control-id: the symbol the-tabular-view,
  height: 200,
  width: 210,
  left: 10,
  top: 10,
  text-color: the symbol blue,
  background-color: the symbol yellow,
  allow-sort-rows: true,
  row-height: 25,
  single-selection: true,
  response-action: the symbol ignore,
  control-value: structure (columns: sequence (structure (text-value: "Center Alignment",
      width: 80,
      alignment: the symbol center),
    structure (text-value: "No Alignment",
      width: 50),
    structure (text-value: "Right Alignment",
      width: 50,
      alignment: the symbol right)),
   rows: sequence (structure (logical-id: 0,
      row-values: sequence (structure (text-value: "Value4"),
        structure (text-value: "Value2"),
        structure (text-value: "-1245.90"))),
```

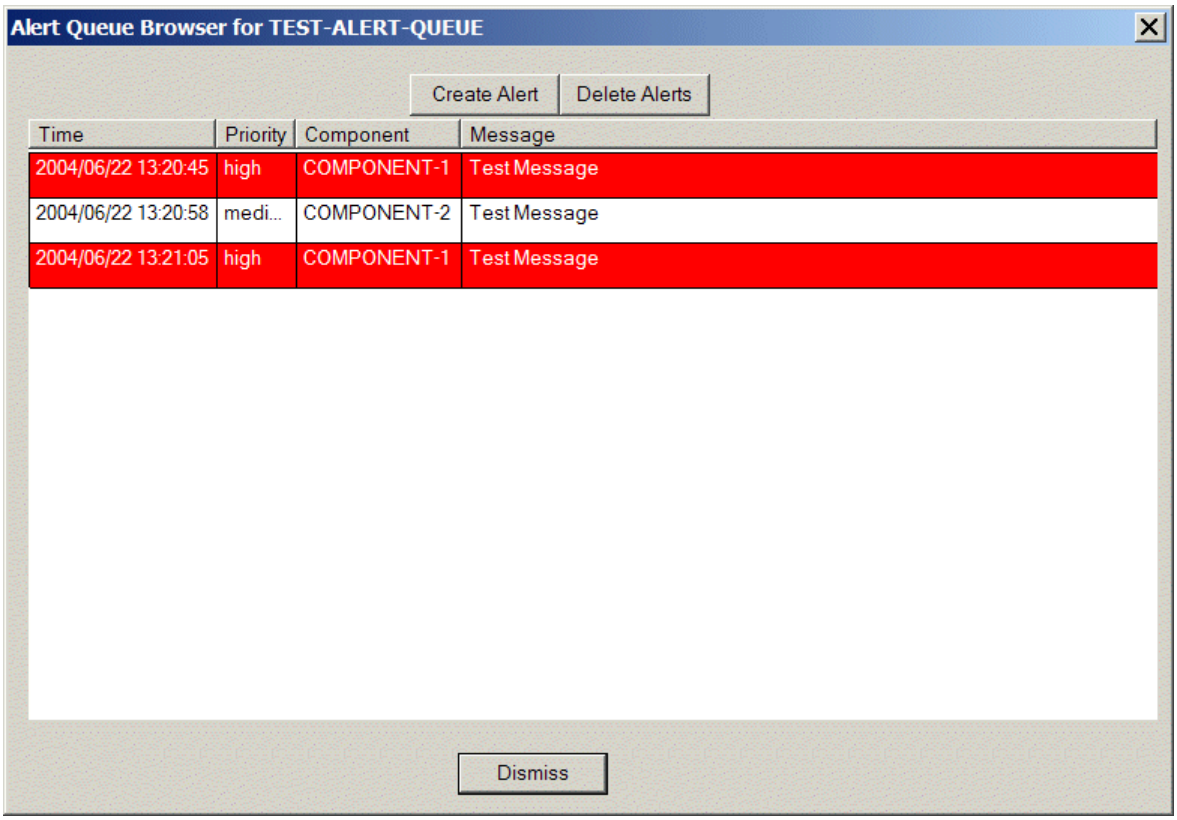
```
structure (logical-id: 1,
  row-values: sequence (structure (text-value: "Override! Right",
     alignment: the symbol right),
   structure (text-value: "Value5"),
   structure (text-value: "3.14"))))))
```

Here is the resulting tabular view:

Center Alignment	No Alignment	Right Alignm...
Value4	Value2	-1245.90
Override! Right	Value5	3.14

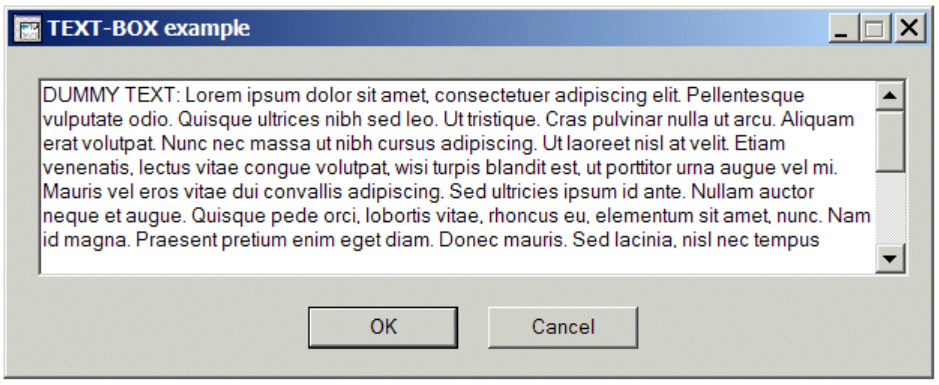
Example: Message Browser

Here is a simple message browser created using a tabular view. For details on this example, click the Alert Queue Demo tab in the custom dialog in the `dialog-demo.kb`.

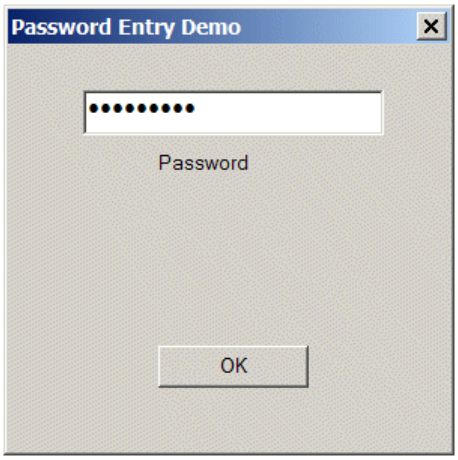


text-box

The text-box control provides a simple text editor:



It also provides a a password text box:



Specific Attributes for Text-Box Control

Attribute Name	Type	Required	Default	Description
is-multiline	truth-value	no	false	Whether the text box supports multiple lines. This is the same as the es-multiline win32 style.
is-readonly	truth-value	no	false	Whether the text box is read only. This is the same as the es-readonly win32 style.
is-autohscroll	truth-value	no	false	Whether the text box supports horizontal scrolling. This is the same as the es-autohscroll win32 style.
is-autovscroll	truth-value	no	false	Whether the text box supports vertical scrolling. This is the same as the es-autovscroll win32 style.
accepts-return	truth-value	no	false	Whether carriage returns are allowed in the text box.
is-lowercase	truth-value	no	true	Whether to coerce input to lower case in the text box.
is-uppercase	truth-value	no	false	Whether to coerce input to upper case in the text box.

Attribute Name	Type	Required	Default	Description
is-password	truth-value	no	false	Whether to display input as password characters. This attribute is only valid when is-multiline is false.
control-value	structure	no	structure (text-value: "")	A structure that specifies the initial text, text color, background color, and initial selection for the text box: structure (text-value: *text*, text-color: *color*, background-color: *color*, selection: *index* | sequence (*index, index*))

In the control-value, specify selection as an integer to set the initial cursor position at the specified index or as a sequence to set the initial selection between the specified indices, where *index* is a zero-based integer index.

The selection is returned as the selection attribute of the control-value, as either an integer or a sequence.

Example: Dialog Specification for Text-Box Control

Here is the dialog specification for the text-box control portion of the dialog above, which provides a multi-line, scrollable text box with sample text:

```
structure
    (dialog-title: "TEXT-BOX example",
    dialog-width: 310,
    dialog-height: 124,
    dialog-is-modal: true,
    dialog-dismissed-callback: dialog-examples-dismissed-callback,
    dialog-update-callback: dialog-examples-update-callback,
    components:
        sequence
            (structure
                (control-type: the symbol push-button,
                . . .
                control-value: structure (text-value: "OK")),
            structure
                (control-type: the symbol push-button,
                . . .
                control-value: structure (text-value: "Cancel")),
            structure
                (control-type: the symbol text-box,
                control-id: 3,
                width: 290,
                height: 66,
                left: 10,
                top: 10,
                response-action: the symbol ignore,
                is-multiline: true,
                vertical-scrollbar: true,
                control-value: structure (text-value: "DUMMY TEXT: ...")))
```

Example: Replacing Text in a Text-Box

Use the replace control action to replace the text in a text-box control:

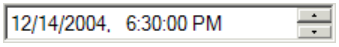
```
call g2-ui-modify-custom-dialog
    (dialog-id,
    sequence
        (structure
            (control-action: the symbol replace,
            control-id: 3,
            control-value: structure
                (text-value: "New text."))),
    window);
```

time-of-day

The time-of-day control provides spinners for configuring the date and the time of day. You select the part of the date or time of day you want to configure, then click the up or down arrow to change its value. You can also enter the value directly in the control.

By default, the control uses a 12-hour clock and does not show the date. You can configure the control to use a 24-hour clock and to show the date.

Here is a time-of-day control that shows the time of day and the date and uses a 12-hour clock:



Specific Attributes for Time-of-Day Control

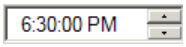
Attribute Name	Type	Required	Default	Description
show-date	truth-value	no	false	Whether to include the date, as well as the time of day. By default, the date does not appear.
show-time	truth-value	no	true	Whether to show the time. By default, the time always appears. To show just the date, specify show-time as false. You can only specify show-time as false if show-date is true.

Attribute Name	Type	Required	Default	Description
use-24-hour-time	truth-value	no	false	Whether to display the value by using a 24-hour clock. By default, the control uses a 12-hour clock and includes AM or PM in the time. This attribute is only relevant if the time is showing.
control-value	structure	yes	N/A	A structure that specifies the default date and time: structure (selected-year: *integer*, selected-month: *integer*, selected-date: *integer*, selected-hour: *integer*, selected-minute: *integer*, selected-second: *integer*)

The selected-year must be between 1601 and 30827, the selected-month must be between 1 and 12, the selected-date must be between 1 and 31, the selected-hour must be between 0 and 23, the selected-minute must be between 0 and 59, and the selected-second must be between 0 and 59.

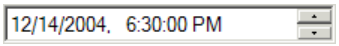
Example: Dialog Specification for Time-of-Day Control

Here is a time-of-day control that shows just the time of day and uses a 12-hour clock, both of which are the defaults. Notice that the selected-hour specifies the hour, using a 24-hour clock.



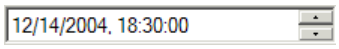

```
structure
    (control-type: the symbol TIME-OF-DAY,
    control-id: the symbol MY-TIME-PICKER,
    control-value:
        structure
            (selected-year: 2004,
            selected-month: 12,
            selected-date: 14,
            selected-hour: 18,
            selected-minute: 30,
            selected-second: 0),
    . . .)
```

Here is a time-of-day control that shows the time of day and the date, and uses a 12-hour clock, the default:




```
structure
    (control-type: the symbol TIME-OF-DAY,
    control-id: the symbol MY-DATE-AND-TIME,
    control-value:
        structure
            (selected-year: 2004,
            selected-month: 12,
            selected-date: 14,
            selected-hour: 18,
            selected-minute: 30,
            selected-second: 0),
    show-date: true,
    . . .)
```

Here is a time-of-day control that shows the time of day and the date, and uses a 24-hour clock:




```
structure
    (control-type: the symbol TIME-OF-DAY,
    control-id: the symbol MY-24-HOUR-DATE,
    control-value:
        structure
            (selected-year: 2004,
            selected-month: 12,
            selected-date: 14,
            selected-hour: 18,
            selected-minute: 30,
            selected-second: 0),
    show-date: true,
    use-24-hour-time: true,
    . . .)
```

Here is a time-of-day control that shows just the date:


```
structure
    (control-type: the symbol TIME-OF-DAY,
    control-id: the symbol MY-JUST-DATE,
    control-value:
        structure
            (selected-year: 2004,
            selected-month: 12,
            selected-date: 14,
            selected-hour: 18,
            selected-minute: 30,
            selected-second: 0),
    show-date: true,
    show-time: false,
    . . .)
```

toggle-button

The toggle-button control provides a button whose state toggles between a selected and unselected state.

Here is a toggle button that shows two different states:

Specific Attributes for Toggle-Button Control

Attribute Name	Type	Required	Default	Description
control-value	structure	yes	N/A	A structure that specifies a text value for the label, or an icon and text value to use as a tool tip, and whether the button is initially selected: structure (text-value: *text*, icon: *item-or-symbol*, selected: *truth-value*)

When specifying an icon, you can provide a G2 class name, an item, or a built-in GMS icon. For a list of built-in GMS icons, see image.

Note The control-value also accepts the icon-name attribute; however, this attribute is deprecated. You should use the icon attribute instead.

Example: Dialog Specification for Toggle-Button Control

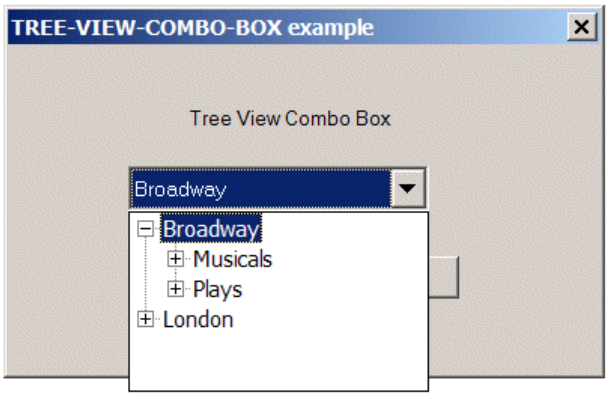
Here is an example:

```
structure (control-type: the symbol toggle-button,
control-id: the symbol my-toggle-button,
height: 14,
width: 50,
left: 125,
top: 25,
response-action: the symbol respond,
control-value:
    structure (icon: the symbol gms-replace-icon,
            text-value: "Toggle Button",
            selected: true))
```

tree-view-combo-box

The tree-view-combo-box control displays a tree view within a combo box.

For more information, see Using Tree Views.



Specific Attributes for Tree-View-Combo-Box Control

Attribute Name	Type	Required	Default	Description
accept-only-leaves	truth-value	no	false	Whether to allow the user to select only leaf nodes in the tree as the selected value.
dropped-height	integer	no	false	The height of the tree view combo box when it is expanded.
dropped-width	integer	no	false	The width of the tree view combo box when it is expanded.
control-value	structure	yes	structure()	Specifies the tree layout, initially selected tree node, and initial size with this syntax: structure (tree-layout: *tree-layout-sequence*, selected: *selected-node-structure*) For a description of this structure, see below.

In the control-value structure:

- *tree-layout-sequence* is a sequence of structure values that specifies the top-level nodes in the tree, where each structure has this syntax:

  structure
  (item-or-name: *item-or-text*,
  children: *sequence-of-structures*)

  The children attribute is optional. The structure can include user-defined attributes. See below.

- *selected-node-structure* is the initially selected item or text in the tree-layout structure, which is a structure that uniquely identifies the tree node. See below for description.

where:

- *item-or-text* is any named G2 item, text string, or symbol.
- *sequence-of-structures* is a sequence of structures with this format:

```
sequence
 (structure
   (item-or-name: item-or-text,
   children: sequence-of-structures)
  . . . )
```

The top-level item-or-name structure attributes in the sequence are the parent nodes in the tree view, and the top-level children are the children of the parent node. The tree structure can be nested as many levels deep as needed to describe the tree.

To specify the selected node, you can refer to the item-or-name of the node in the tree-layout structure, or you can refer to any user-defined attribute in the tree-layout structure that uniquely identifies the node. For example, to select one of several nodes whose item-or-name in the tree-layout structure is identical, you can add a user-defined attribute to the tree-layout structure, such as id, and refer to this attribute in the selected structure.

To create a tree view that is the class hierarchy of a particular class, you can use the g2-get-class-hierarchy system procedure by passing in the class to use as a root of the tree view. Use the return value of the procedure as one of the structure values in the tree-layout sequence. You can also construct your own tree structure.

The *new-value* argument of the dialog update callback is a structure with the selected attribute equal to the selected node and with the tree-layout structure of the original control-value structure.

Example: Dialog Specification for Tree-View-Combo-Box

Here is the dialog specification for the tree-view-combo-box control portion of the dialog above:

```
structure (control-type: the symbol tree-view-combo-box,
   control-id: the symbol my-tvcb,
   width: 100,
   height: 15,
   left: 40,
   top: 40,
   response-action: the symbol ignore,
   control-value: structure (selected: structure (item-or-name: "Hairspray"),
    tree-layout: sequence (structure (item-or-name: "Broadway",
       children: sequence (structure (item-or-name: "Musicals",
          children: sequence (structure (item-or-name: "Hairspray"),
            structure (item-or-name: "Little Women"),
            structure (item-or-name: "Sweet Charity"),
            structure (item-or-name: "Wicked"))),
         structure (item-or-name: "Plays",
          children: sequence (structure (item-or-name: "Doubt"),
            structure (item-or-name: "700 Sundays"),
            structure (item-or-name: "The Glass Menagerie"))))),
      structure (item-or-name: "London",
       children: sequence (structure (item-or-name: "Musicals",
          children: sequence (structure (item-or-name: "Mamma Mia!"),
            structure (item-or-name: "The Far Pavillion"),
            structure (item-or-name: "The Producers"))),
         structure (item-or-name: "Plays",
          children: sequence (structure (item-or-name: "Death of a Salesman"),
            structure (item-or-name: "The Mousetrap"),
            structure (item-or-name: "The Woman in Black")))))))
```

Example: Modifying a Tree-View-Combo-Box

Here is the specification for modifying the control height and width of a tree-view-combo-box:

```
sequence
     (structure
          (control-action: the symbol dropped-height,
          control-id: the symbol my-combo-tree-view,
          control-value: 500),
     structure
          (control-action: the symbol dropped-width,
          control-id: the symbol my-combo-tree-view,
          control-value: 500))
```

track-bar

The track-bar control provides a slider with discrete increments that you can use to set a value.

Specific Attributes for Track-Bar Control

Attribute Name	Type	Required	Default	Description
control-value	structure	yes	the current date	A structure that specifies the value range, current value, and increment: structure (low-value: *integer*, high-value: *integer*, current-value: *integer*, increment: *integer*)

Example: Dialog Specification for Track-Bar Control

For example:

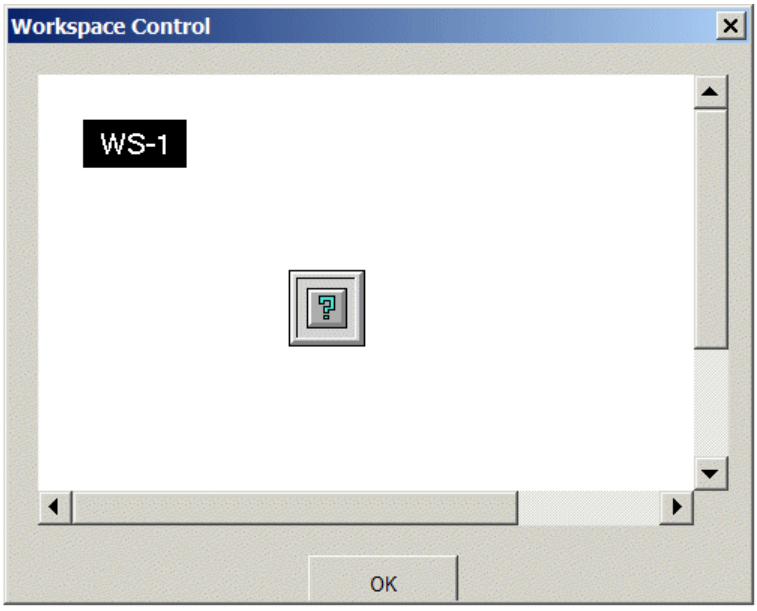
```
structure
    (control-type: the symbol TRACK-BAR,
    control-id: the symbol MY-TRACK-BAR,
    control-value:
        structure
            (low-value: 0,
            high-value: 10,
            current-value: 5,
            increment: 1),
    . . .)
```

workspace

The workspace control displays a G2 workspace within a dialog. You specify the workspace name as a text or a symbol, or the workspace UUID of the workspace to display.

Note Currently, the workspace control must appear in an MDI dialog, that is, a dialog whose dialog-is-mdi-child attribute is true. You cannot use it in a modal or modeless dialog, that is, a dialog whose dialog-is-modal is true or false, and whose dialog-is-mdi-child attribute is false.Also, the workspace control does not currently support the replace control action for modifying the control.

Note The Native Menu System (NMS) API allows you to create popup menus on workspace controls. NMS does not support menu bars or tool bars on workspace controls.



Specific Attributes for Workspace Control

Attribute Name	Type	Required	Default	Description
control-value	structure	yes	N/A	A structure that specifies the workspace name or UUID to display: structure (workspace-name: *text* | *symbol*) or structure (workspace-uuid: *text*)

Example: Dialog Specification for Workspace Control

Here is the dialog specification for a workspace control:

```
structure
    (control-type: the symbol WORKSPACE,
    control-id: the symbol MY-WORKSPACE,
    control-value: structure (workspace-name: the symbol WS-1),
    height: 150,
    width: 230,
    left: 10,
    top: 10)
```

Summary of Control Values

The control-value attribute is used in the following three ways:

- Specifying the initial state of a control.
- Specifying updates to a control.
- Providing the current state of the control in a callback.

The following table summarizes the different uses of the control-value attribute.

Control Type	Control Value Format	Supported Control Actions
calendar	structure (selected-year: *integer*, selected-month: *integer*, selected-date: *integer*)	replace enable disable hide show
check-box	structure (text-value: *text* selected: *truth-value*)	check uncheck replace enable disable hide show
checkable-list-box	structure (text-sequence: sequence (*text*[, ...]), checked: sequence (*text*[, ...]), selected: sequence (*text*[, ...]))	add replace enable disable hide show
color-picker	structure (selected: *color*)	replace enable disable hide show

Control Type	Control Value Format	Supported Control Actions
combo-box	structure (text-sequence: sequence (*text*[, ...]) | ( ), selected: *text*, text-selection: *index*, dropdown-width: *integer*)	add replace enable disable hide show
duration	structure (number-of-weeks: *integer*, number-of-days: *integer*, number-of-hours: *integer*, number-of-minutes: *integer*, number-of-seconds: *integer*)	replace enable disable hide show
full-color-picker	structure (selected: *rbg-color*)	replace enable disable hide show
grid-view	structure (columns: *sequence-of-columns*, rows: *sequence-of-rows*, selected-cells: *sequence-of-cells*)	replace enable disable hide show
group	structure (text-value: *text*)	replace enable disable hide show
image	structure (icon: *item-or-symbol*)	replace enable disable hide show

Control Type	Control Value Format	Supported Control Actions
label	structure (text-value: *text*)	replace enable disable hide show
list-box	structure (text-sequence: sequence (*text*[, ...]), selected: *text*)	add replace enable disable hide show
masked-edit	structure (current-value: *text*, mask: *text*, literal: *text*, text-color: *color*, background-color: *color*, selection: *index* | sequence (*index, index*))	replace enable disable hide show
progress-bar	structure (low-value: *integer*, high-value: *integer*, current-value: *integer*)	replace enable disable hide show
push-button	structure (text-value: *text*, icon: *item-or-symbol*)	replace enable disable hide show

Control Type	Control Value Format	Supported Control Actions
radio-button	structure (text-value: *text,* selected: *truth-value*)	check uncheck add replace enable disable hide show
slider	structure (low-value: *integer*, high-value: *integer*, current-value: *integer*, increment: *integer*, precision: *ddd.dddd-format*)	replace enable disable hide show
spinner	structure (current-value: *quantity*, low-value: *quantity,* high-value: *quantity*, increment: *quantity*, precision: *ddd.dddd-format*)	replace enable disable hide show
tab-frame	structure (tab-labels: sequence (*text*,...), tab-icons: sequence (*item-or-symbol*, ...), selected-tab: *text*, tab-position: top | left | right | bottom)	enable disable hide show

Control Type	Control Value Format	Supported Control Actions
tabular-view	structure (columns: sequence (*column-structure*, ...), rows: sequence (*row-specification*, ...), selected: *integer*)	enable disable hide show add-rows add-column remove-rows replace-rows replace-cells remove-all-rows remove-all-selected-rows
text-box	structure(text-value: *text*)	add replace enable disable hide show
time-of-day	structure (selected-year: *integer*, selected-month: *integer*, selected-date: *integer*, selected-hour: *integer*, selected-minute: *integer*, selected-second: *integer*)	replace enable disable hide show
toggle-button	structure (text-value: *text*, icon: *item-or-symbol*, selected: *truth-value*)	check uncheck replace enable disable hide show

Control Type	Control Value Format	Supported Control Actions
track-bar	structure (low-value: *integer*, high-value: *integer*, current-value: *integer*, increment: *integer*)	replace enable disable hide show
tree-view-combo-box	structure (tree-layout: *tree-layout-sequence*, selected: *selected-node-structure*)	replace enable disable hide show
workspace	structure (workspace-name: *text* | *symbol*) or structure (workspace-uuid: *text*)	enable disable hide show

Win32 Control Types

This section describes the win32 control types and their equivalent attributes. For information on how to use these control types, see Windows-Specific Control Styles.

WIN32 Window Style Symbols

Style Symbol	Win32 API Constant	Equivalent Attribute
ws-overlapped	WS_OVERLAPPED	
ws-popup	WS_POPUP	
ws-child	WS_CHILD	is-child
ws-minimize	WS_MINIMIZE	
ws-visible	WS_VISIBLE	is-visible
ws-disabled	WS_DISABLED	

Style Symbol	Win32 API Constant	Equivalent Attribute
ws-clipsiblings	WS_CLIPSIBLINGS	
ws-clipchildren	WS_CLIPCHILDREN	
ws-maximize	WS_MAXIMIZE	
ws-caption	WS_CAPTION	
ws-dlgframe	WS_DLGFRAME	
ws-vscroll	WS_VSCROLL	
ws-hscroll	WS_HSCROLL	
ws-sysmenu	WS_SYSMENU	
ws-thickframe	WS_SYSMENU	
ws-group	WS_GROUP	
ws-tabstop	WS_TABSTOP	is-tabstop
ws-minimizebox	WS_MINIMIZEBOX	
ws-maximimizebox	WS_MAXIMIZEBOX	

WIN32 Static Control Style Symbols

Style Symbol	Win32 API Constant	Equivalent Attribute
ss-left	SS_LEFT	
ss-right	SS_RIGHT	
ss-center	SS_CENTER	
ss-icon	SS_ICON	
ss-blackrect	SS_BLACKRECT	
ss-grayrect	SS_GRAYRECT	
ss-whiterect	SS_WHITERECT	
ss-blackframe	SS_BLACKFRAME	

Style Symbol	Win32 API Constant	Equivalent Attribute
ss-grayframe	SS_GRAYFRAME	
ss-whiteframe	SS_WHITEFRAME	
ss-useritem	SS_USERITEM	
ss-simple	SS_SIMPLE	
ss-leftnowordwrap	SS_LEFTNOWORDWRAP	
ss-ownerdraw	SS_OWNERDRAW	
ss-bitmap	SS_BITMAP	
ss-enhmetafile	SS_ENHMETAFILE	
ss-echtedhorz	SS_ETCHEDHORZ	
ss-etchedframe	SS_ECHEDFRAME	
ss-typemask	SS_TYPEMASK	
ss-noprefix	SS_NOREFIX	
ss-notify	SS_NOTIFY	
ss-centerimage	SS_CENTERIMAGE	
ss-rightjust	SS_RIGHTJUST	
ss-realsizeimage	SS_REALSIZEIMAGE	
ss-sunken	SS_SUNKEN	
ss-endellipsis	SS_ENDELLIPSIS	
ss-pathendellipsis	SS_PATHEMDELLIPSIS	
ss-wordellipsis	SS_WORDELLIPSIS	
ss-ellipsismask	SS_ELLIPSISMASK	

WIN32 Edit Style Symbols

Style Symbol	Win32 API Constant	Equivalent Attribute
es-left	ES_LEFT	
es-center	ES_CENTER	
es-right	ES_RIGHT	
es-multiline	ES_MULTILINE	is-multiline
es-uppercase	ES_UPPERCASE	
es-lowercase	ES_LOWERCASE	
es-password	ES_PASSWORD	
es-autovscroll	ES_AUTOVSCROLL	is-autovscroll
es-aotohscroll	ES_AUTOHSCROLL	is-autohscroll
es-nohidesel	ES_NOHIDESEL	
es-oemconvert	ES_OEMCONVERT	
es-readonly	ES_READONLY	is-readonly
es-wantreturn	ES_WANTRETURN	
es-number	ES_NUMBER	

WIN32 Button Style Symbols

Style Symbol	Win32 API Constant	Equivalent Attribute
bs-pushbutton	BS_PUSHBUTTON	
bs-defpushbutton	BS_DEFPUSHBUTTON	
bs-checkbox	BS_CHECKBOX	
bs-autocheckbox	BS_AUTOCHECKBOX	
bs-radiobutton	BS_RADIOBUTTON	
bs-3state	BS_3DSTATE	
bs-auto3state	BS_AUTO3STATE	

Style Symbol	Win32 API Constant	Equivalent Attribute
bs-groupbox	BS_GROUPBOX	
bs-userbutton	BS_USERBUTTON	
bs-autoradiobutton	BS_AUTORADIOBUTTON	
bs-ownerdraw	BS_ONEDRAW	
bs-lefttext	BS_LEFTTEXT	
bs-text	BS_TEXT	
bs-icon	BS_ICON	
bs-bitmap	BS_BITMAP	
bs-left	BS_LEFT	
bs-right	BS_RIGHT	
bs-center	BS_CENTER	
bs-top	BS_TOP	
bs-bottom	BS_BOTTOM	
bs-vcenter	BS_VCENTER	
bs-flat	BS_FLAT	
bs-right	BS_RIGHTBUTTON	

WIN32 Combo-Box Style Symbols

Style Symbol	Win32 API Constant	Equivalent Attribute
cbs-simple	CBS_SIMPLE	
cbs-dropdown	CBS_DROPDOWN	
cbs-dropdownlist	CBS_DROPDOWNLIST	
cbs-ownerdrawfixed	CBS_OWNERDRAWFIXED	
cbs-ownerdrawvariable	CBS_OWNERDRAWVARIABLE	
cbs-autohscroll	CBS_AUTOHSCROLL	

Style Symbol	Win32 API Constant	Equivalent Attribute
cbs-oemconver	CBS_OEMCONVER	
cbs-sort	CBS_SORT	
cbs-hasstrings	CBS_HASSTRINGS	
cbs-nointegralheight	CBS_NOINTEGRALHEIGHT	
cbs-disablenoscroll	CBS_DISABLESCROLL	
cbs-uppercase	CBS_UPPERCASE	

WIN32 Spinner Style Symbols

Style Symbol	Win32 API Constant	Equivalent Attribute
uds-wrap	UDS_WRAP	
uds-setduddyint	UDS_SETBUDDYINT	
uds-alignright	UDS_ALIGNRIGHT	
uds-alignleft	UDS_ALIGNLEFT	
uds-autobuddy	UDS_AUTOBUDDY	
uds-arrowkeys	UDS_ARROWKEYS	
uds-horz	UDS_HORZ	
uds-nothousands	UDS_NOTHOUSANDS	
uds-hottrack	UDS_HOTTRACK	

WIN32 Tabular-View Style Symbols

Style Symbol	Win32 API Constant	Equivalent Attribute
lvs-icon	LVS_ICON	
lvs-report	LVS_REPORT	
lvs-smallicon	LVS_SMALLICON	
lvs-list	LVS_LIST	
lvs-typemask	LVS_TYPEMASK	
lvs-singlesl	LVS_SINGLESEL	
lvs-showselalways	LVS_SHOWSELALWAYS	
lvs-sortascending	LVS_SORTASCENDING	
lvs-shareimagelists	LVS_SHAREIMAGELISTS	
lvs-nolabelwrap	LVS_NOLAVELWRAP	
lvs-autoarrange	LVS_AUTOARRANGE	
lvs-editlabels	LVS_EDITLABELS	
lvs-ownerdata	LVS_OWNERDATA	
lvs-noscroll	LVS_NOSCROLL	
lvs-typestylemask	LVS_TYPESTYLEMASK	

44

Windows Views, Panes, and UI Features

Provides examples of how to create Windows chart views, HTML views, shortcut bars, tree views, and other Windows user interface features.

Introduction

G2 provides numerous system procedures for creating Windows views, panes, and other user interface features. Some of these features are only supported in Telewindows Next Generation (`twng.exe`) while others are supported in Telewindows (`tw.exe`).

This chapter provides examples of following Windows user interface features:

- Chart view
- HTML view
- HTML help
- Property grid
- Shortcut bar
- Tree view
- Status bar
- Workspace view
- Tabbed MDI mode

You specify these Windows user interface features as part of your G2 application, using system procedures described in User Interface Operations in the *G2 System Procedures Reference Manual*.

Other Windows user interface features include:

- User interface themes
- Dockable and floating panes
- Resizable dialogs
- Windows Text Editor, Message Board, and Operator Logbook

For details, see:

- Window Handles and Views in User Interface Operations in the *G2 System Procedures Reference Manual*.
- Editor Parameters, Logbook Parameters, and Message Board Parameters.

Using Chart Views

G2 provides a Windows chart view, which you access through Telewindows.

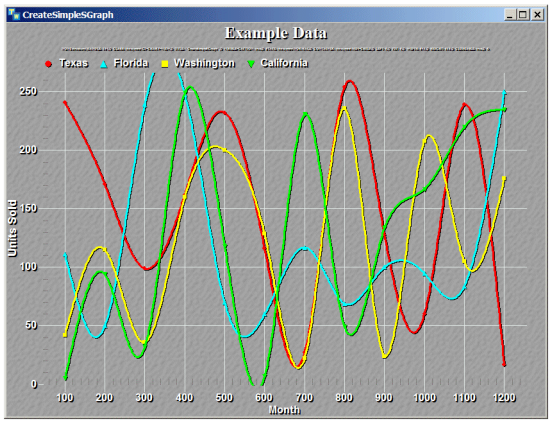
Note Chart views are only supported in Telewindows Next Generation (`twng.exe`).

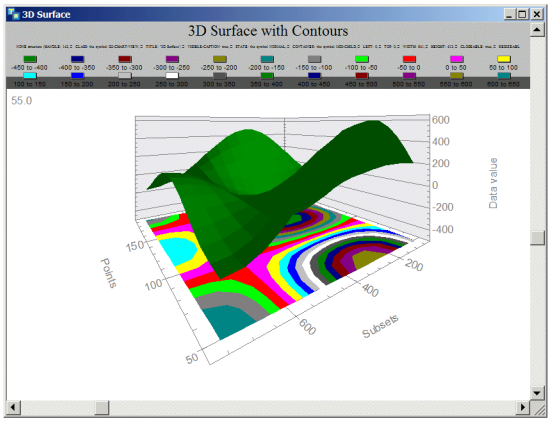
When you install Telewindows, the required chart DLL (`pegrp32d.dll`) is automatically installed.

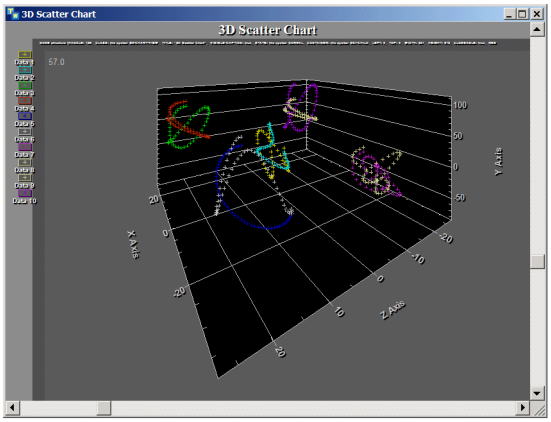
The chart view API supports these features:

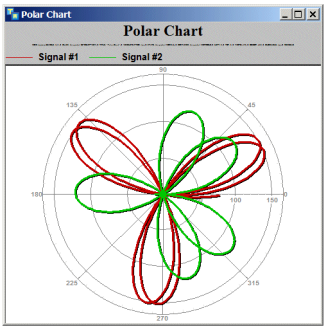
- Creating a chart view and configuring its properties.
- Modifying the chart view properties.
- Setting individual properties of a single data element in a chart view.
- Printing, copying, and destroying a chart view.
- Exporting chart views to an image file.
- Callback procedure when the user clicks a point or closes the chart view.

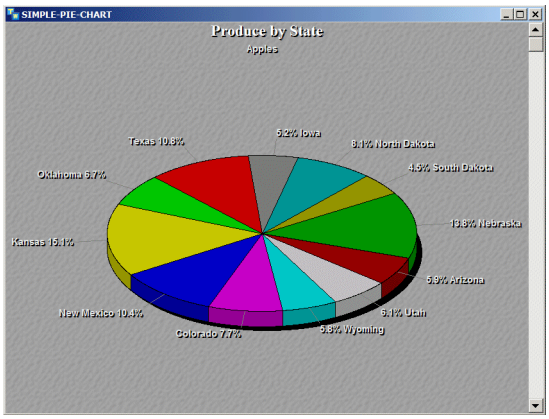
The chart view is based on the ProEssentials™ package from Gigasoft, Inc. (`http://www.gigasoft.com`). Most of the documentation on their Web site applies to the G2 implementation, except that the names of the various properties are hyphenated, rather than using mixed capitalization. For example, `MainTitle` becomes main-title in G2. For a list of all of the property names that the G2 implementation of chart views uses, see Appendix B, Chart Properties and Enumeration Values in the *G2 System Procedures Reference Manual*.

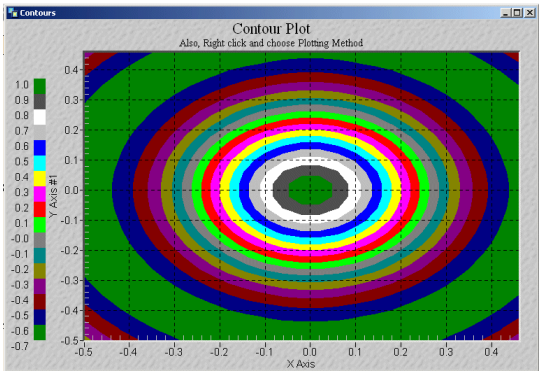












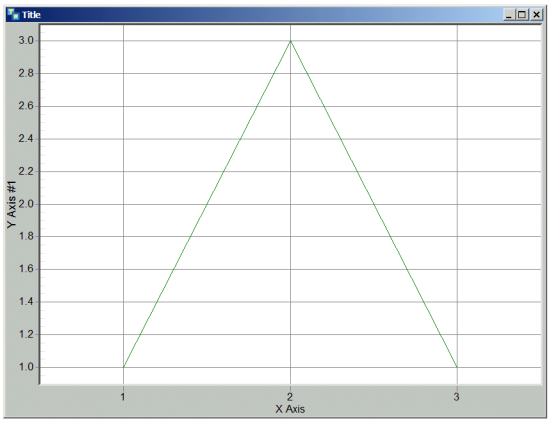
For examples, load `charts.kb` in the `g2\kbs\samples` directory.

For a description of the G2 system procedures for working with chart views, see Chart Views in User Interface Operations in the *G2 System Procedures Reference Manual*.

Creating a Simple Chart

This procedure creates a simple chart:

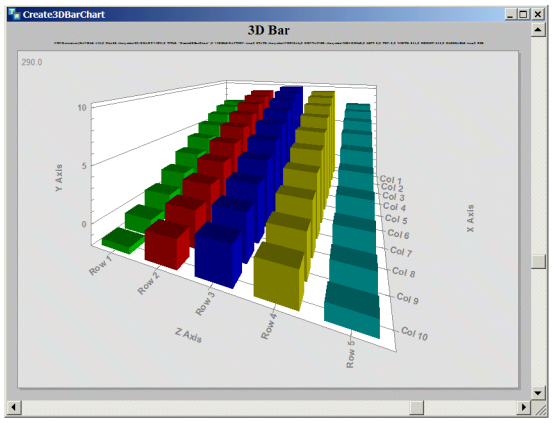
```
start g2-ui-create-chart-view
("Chart View", the symbol NONE, structure(ydata: sequence(1,3,1)), this window)
```



Creating a Simple Bar Chart

This procedure creates a simple bar chart:

```
start g2-ui-create-chart-view
("Bar Chart", the symbol NONE, structure(ydata: sequence(1,3,1),
plotting-method: the symbol PEGPM-BAR), this window)
```



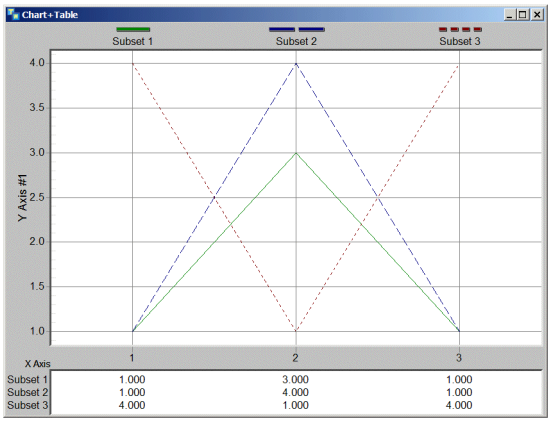
Creating a Simple Chart and Table

This button creates a simple chart and table:

Chart+Table

```
start g2-ui-create-chart-view("Chart+Table",
  the symbol NONE, structure(subsets: 3,
  points: 3, ydata: sequence(1,3,1, 1,4,1, 4,1,4),
  graph-plus-table: the symbol PEGPT-BOTH),
  this window)
```

```
start g2-ui-create-chart-view
("Chart+Table", the symbol NONE, structure(subsets: 3, points: 3,
ydata: sequence(1,3,1, 1,4,1, 4,1,4), graph-plus-table: the symbol PEGPT-BOTH),
this window)
```



Populating a Chart View

The following procedure code plots values in a chart:

```
{ Plot the current history of given G2 variable, with annotations. }
chart-variable-history(var: class g2-variable, win: class g2-window)
props: structure;
point: structure;
xdata, ydata: sequence;
x, y, unix-time: float;
utc-offset: integer = -4;  { US Eastern Daylight Time }
begin
  xdata = sequence();
  ydata = sequence();

  for point = each structure in the history-using-unix-time of var do
    x = unix-time-to-vb-date(the history-collection-time of point, utc-offset);
    y = the history-value of point;
    xdata = insert-at-end(xdata, x);
    ydata = insert-at-end(ydata, y);
  end;
```

```
  props = structure(type: the symbol SGRAPH, { A Scientific Graph }
   main-title: "History of [the name of var]",
   subtitle: "Click on any data point",
   {Enable DateTimeMode}
   date-time-mode: the symbol PEDTM-VB,

   { Various appearance options }
   quick-style: the symbol PEQS-DARK-INSET,
   bitmap-gradient-mode: false,
   mark-data-points: true,
   fixed-fonts: true,
   gradient-bars: 8,
   subtitle-font: "Arial Unicode MS",
   subtitle-bold: true,
   label-bold: true,
   line-shadows: true,

   { Let the user zoom in }
   allow-zooming: the symbol PEAZ-HORZANDVERT,
   zoom-style: the symbol PEZS-RO2-NOT,

   {Make data points be mouse sensitive}
   allow-data-hot-spots: true,
   hot-spot-size: the symbol PEHSS-LARGE,

   { Add some lines showing set points }
   show-annotations: true,
   horz-line-annotation: sequence(the maximum value of var during the last 1 hour, the
minimum value of var during the last 1 hour),
   horz-line-annotation-type: sequence(the symbol PELT-DOT, the symbol PELT-DOT),
   horz-line-annotation-color: sequence(the symbol RGBFF8080, the symbol
RGBFF8080),
   horz-line-annotation-text: sequence("|RHigh", "|RLow"), { "|R" prefix puts text on
right}
   right-margin: "XXXX",            { Make room for some text on right }
   line-annotation-text-size: 110,   { Also increase text size }

   { The data }
   xdata: xdata,
   ydata: ydata,
   subset-colors: sequence(the symbol GREEN));

  call g2-ui-create-chart-view("[the name of var]", the symbol CALLBACK, props, win);
end
```

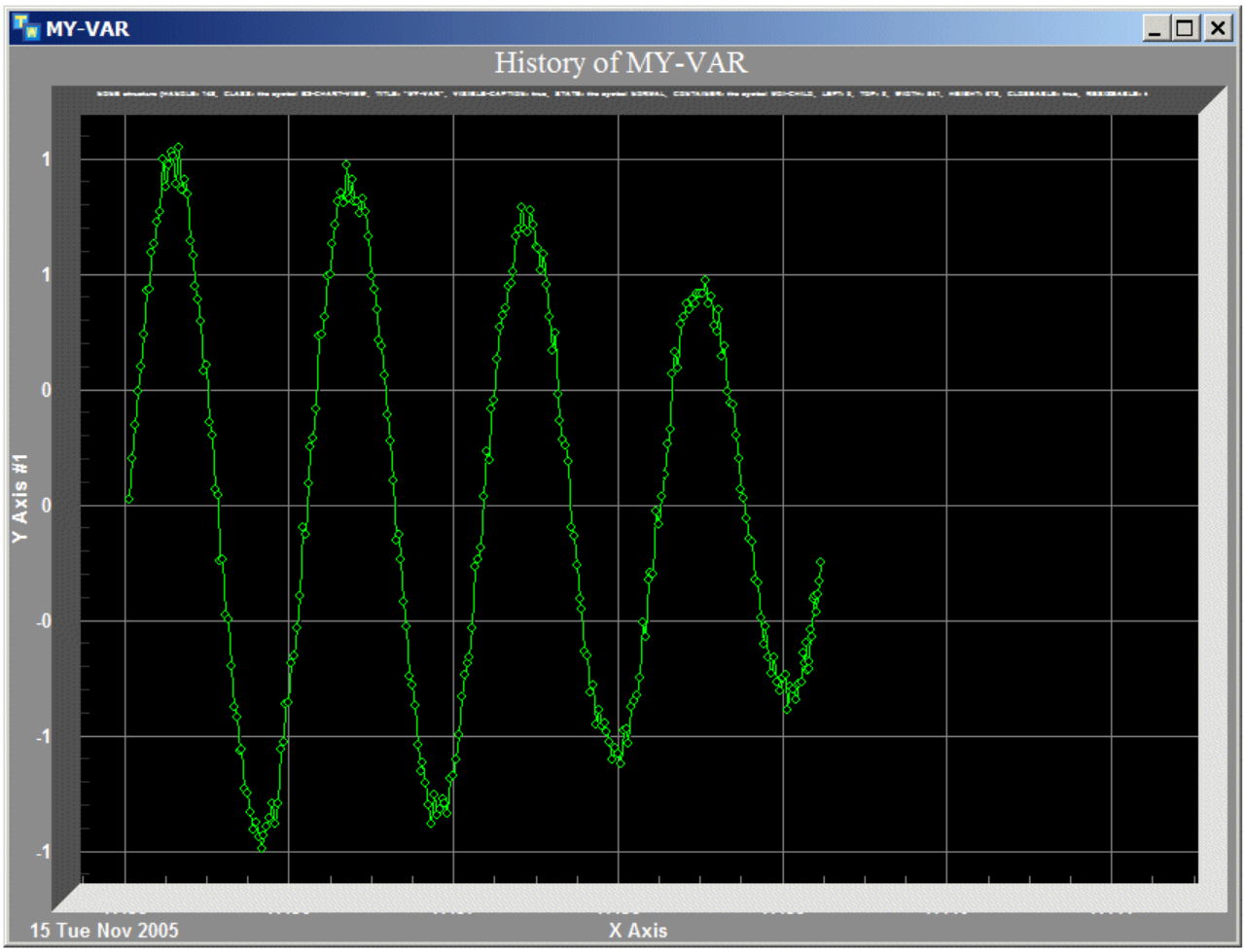
Here is the button that starts the procedure, the variable whose data the chart plots, and the function called in the procedure:

Variable History

start chart-variable-history(MY-VAR, this window)

-0.211, valid indefinitely
random(-100,100)/1000.0 + sin(the current subsecond time/10.0) + 0.5 * sin(the current subsecond time/11.0)
MY-VAR 0.5 seconds

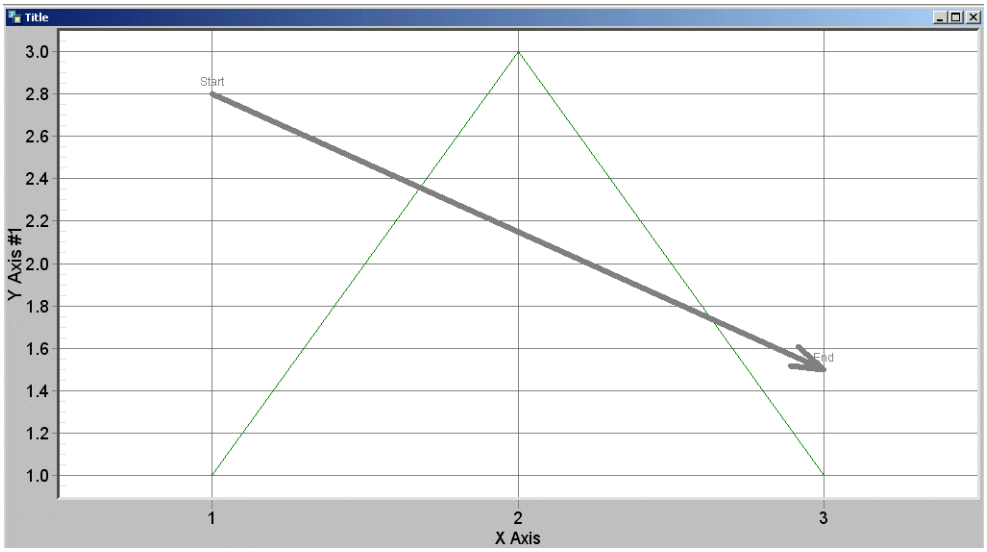
```
unix-time-to-vb-date(unix-time, utc-offset) =
    (unix-time + 3600.0 * utc-offset) / 86400.0
    + 25569
```



Displaying Annotations

The following procedure creates a graph using annotations:

```
start g2-ui-create-chart-view
("Title", the symbol NONE, structure
      (ydata: sequence(1,3,1),
      show-annotations: true,
      graph-annotation-x: sequence(1.0, 3.0),
      graph-annotation-y: sequence(2.8, 1.5),
      graph-annotation-type: sequence(the symbol PEGAT-THICKSOLIDLINE,
            the symbol PEGAT-ARROW-LARGE),
      graph-annotation-text: sequence("Start", "End")),
this window)
```



Exporting a Chart View

This procedure call exports a chart view to a JPEG file:

```
start g2-ui-manage-chart-view
      (the symbol EXPORT, chart, structure(pathname: “c:\my-chart.jpeg”), this window)
```

Printing a Chart View

This procedure call displays a print dialog for a chart:

```
start g2-ui-manage-chart-view
      (the symbol PRINT, chart, structure(), this window)
```

Deleting a Chart View

This procedure call deletes a chart view:

```
start g2-ui-manage-chart-view
    (the symbol DESTROY, chart, structure(), this window)
```

Example Callback: Chart View

Here is an example callback procedure for a chart view:

```
callback (event: symbol, Win: class g2-window, handle: integer, data: value,
    info: structure, user-data: value)
begin
    change the text of DPY to "Event: [event] Handle: [handle]";
    if(event is not CLOSED) then
    call g2-ui-modify-chart-view(handle, structure(subtitle: "[data] [info]"), Win);
end
```

Here is the result of closing a chart view:

Using HTML Views

G2 provides a Windows HTML view, which allows you to embed an Internet Explorer Web Browser window into Telewindows.

Note HTML views are only supported in Telewindows Next Generation (`twng.exe`).

The HTML view API supports:

- Creating HTML views that display an initial URL.
- Sending commands to go forward and back, go to a given URL, go to the home page, stop downloading, refresh, and destroy the view.

For a description of the G2 system procedures for working with HTML views, see HTML Views in User Interface Operations in the *G2 System Procedures Reference Manual*.

Creating an HTML View

This procedure creates an HTML view, passing in a URL as an argument. It stores the handle to the HTML view in an integer parameter named htmlview.

```
make(url: text, win: class g2-window)
hv: integer;
begin
    hv = call g2-ui-create-html-view(url, the symbol CALLBACK,
        structure(width: 500, height:700, left: 10, top:10), win);
    conclude that htmlview = hv;
end
```

This procedure call creates an HTML view that goes to `www.gensym.com`:

```
start make("http://www.gensym.com", this window)
```

Here is the action button that creates the HTML view and the integer parameter that stores the handle:

Here is the resulting HTML view:

Going to a Web Page

This procedure call goes to a web page specified by a message named url, using the HTML view specified by the htmlview parameter:

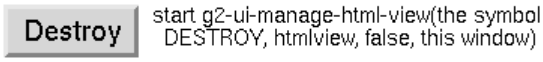
```
start g2-ui-manage-html-view(the symbol GOTO, htmlview, the text of url, this window)
```

Goto start g2-ui-manage-html-view(the symbol GOTO, htmlview, the text of URL, this window) http://google.com URL

Destroying an HTML View

This procedure call destroys the HTML view specified by the htmlview parameter:

```
start g2-ui-manage-html-view(the symbol DESTROY, htmlview, false, this window)
```



Example Callback: HTML View

This callback procedure is called when the user closes the HTML view, which changes the text of a message named msg:

```
callback (event: symbol, Win: class g2-window, control: integer, item: value, info:
structure, user-data: value)
begin
     change the text of MSG to "[event] Control [control] Item @"[item]@" info [info]
          user-data: [user-data]";
end
```

Here is the result of closing the HTML view:

```
CLOSED Control 77 Item "NONE" info structure
  (HANDLE: 77,
  CLASS: the symbol G2-HTML-VIEW,
  CONTAINER: the symbol MDI-CHILD,
  VISIBLE-CAPTION: true,
  STATE: the symbol NORMAL,
  LEFT: 14,
  TOP: 38,
  WIDTH: 492,
  HEIGHT: 668,
  CLOSEABLE: true,
  RESIZEABLE: true,
  MINIMIZEABLE: true,
  MAXIMIZEABLE: true,
  CALLBACK: the symbol CALLBACK)
          user-data: NONE
```

Using HTML Help

You can launch Windows HTML help from Telewindows. The HTML Help API supports:

- Launching Windows HTML Help (`.chm`) files.
- Displaying topics by name or ID.
- Displaying the index or table of contents.
- Displaying popup help.

For a description of the G2 system procedure for launching HTML help, see HTML Views in User Interface Operations in the *G2 System Procedures Reference Manual*.

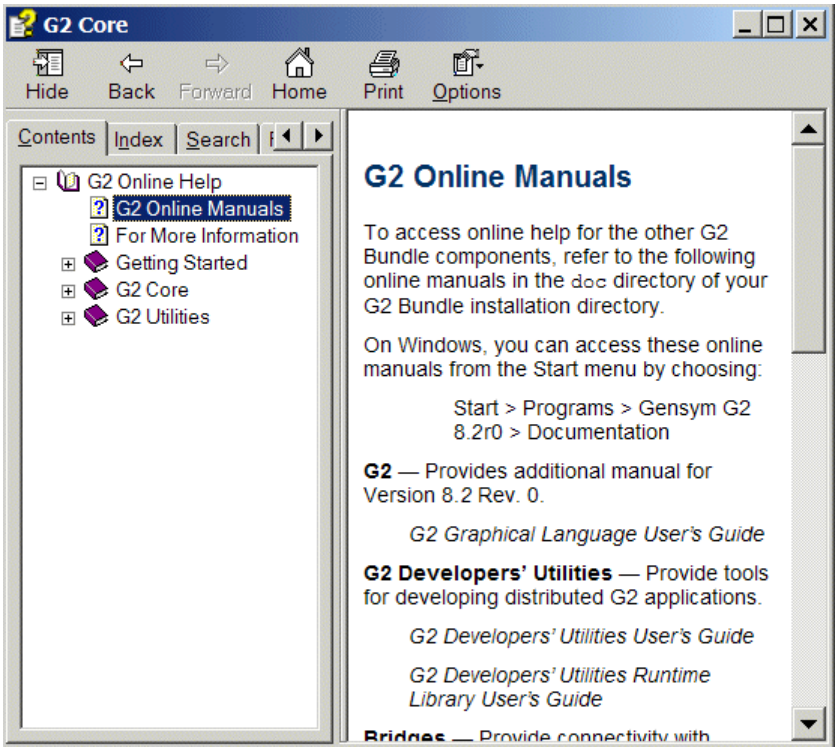
For example, this procedure provides a way of managing HTML Help windows:

```
test-html-help(cmd: symbol, file: text, topic: value, win: class g2-window)
begin
    call g2-ui-html-help(cmd, structure(help-file-directory:
        "c:\Program Files\Gensym\g2-2011\g2\",
        help-file-name: "[file]", topic: topic), win);
end
```

Displaying a Topic

The following code fragment launches Windows Help for the `Master2.html` topic in the `Master.chm` help file:

```
start test-html-help(the symbol DISPLAY-TOPIC, "Master.chm", "Master2.html",
    this window)
```



The following code fragment launches the topic associated with the topic whose integer ID is 1127:

```
start test-html-help(the symbol DISPLAY-TOPIC, "G2ReferenceManualHelp.chm",
    1127, this window)
```

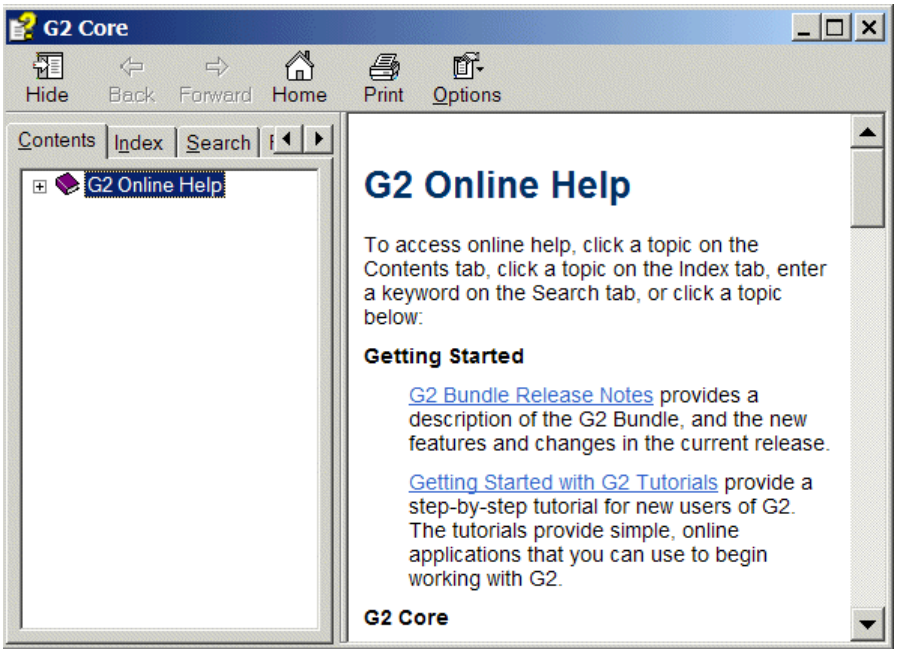
You can also provide a URL such as the following as the topic:

```
start test-html-help(the symbol DISPLAY-TOPIC, "G2ReferenceManualHelp.chm",
    "mk:@MSITStore:c:\Program%20Files\Gensym\g2-2011\g2\
    G2ReferenceManualHelp.chm::/kbs15.html", this window)
```

Displaying the Table of Contents

The following code fragment displays the Table of Contents of the help file named `Master.chm`:

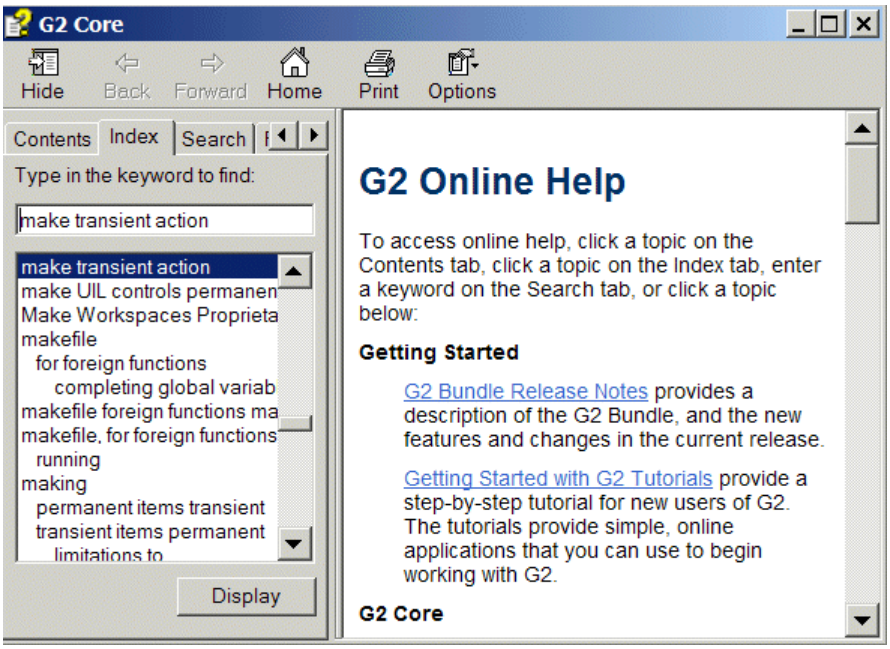
```
start test-html-help(the symbol DISPLAY-CONTENTS, "Master.chm", 0, this window)
```



Displaying the Index

The following code fragment displays the index for the help file named `Master.chm` with the index scrolled to the make transient action entry:

```
start test-html-help(the symbol DISPLAY-INDEX, "Master.chm",
    "make transient action", this window)
```



Displaying Popup Help

The following code fragment displays some text in a popup window:

```
start test-html-help(the symbol DISPLAY-POPUP, "Master.chm",
    "Here is some popup text", this window)
```

Using Property Grid

A property grid displays a two-column list of names and values, called properties, where each value may optionally be edited by the user, in a way that depends on its type and other options.

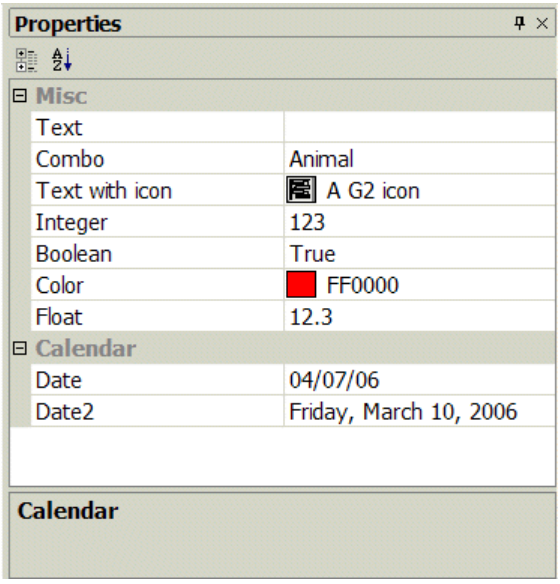
Each property may optionally be assigned to a category.

The grid may have a toolbar positioned above it and a help pane below it. The help pane displays the description of the currently selected property.

For a description of the G2 system procedure for creating property grids, see Property Grid Views in User Interface Operations in the *G2 System Procedures Reference Manual*.

Here is a simple example:

```
create-property-grid(win: class g2-window)
begin
  call g2-ui-create-property-grid("Properties", the symbol CB,
    structure(width: 300, user-data: "User Data", show-toolbar: true,
        contents: sequence(
        "Text",
        structure(property: the symbol COMBO, current-value: "Animal",
            possible-values: sequence("Animal", "Vegetable", "Mineral"),
            description: "This property has a combo box."),
        structure(property: the symbol TEXT-WITH-ICON,
                icon: this procedure, current-value: "A G2 icon", ellipsis: true,
                 edit-in-place: false),
      structure(property: the symbol INTEGER, current-value: 123),
        structure(property: the symbol BOOLEAN,
                  type: the symbol BOOLEAN,
                  current-value: true),
      structure(property: the symbol COLOR, type: the symbol COLOR, current-value:
the symbol RED, select: true),
      structure(property: the symbol DATE, type: the symbol DATE,
            category: "Calendar", current-value: the current subsecond real time),
      structure(property: the symbol DATE2, type: the symbol DATE,
            category: "Calendar", value-format: the symbol LONG-DATE,
            current-value: structure(month: 3, date: 10)),
      structure(property: the symbol FLOAT, current-value: 12.3,
            value-format: the symbol dd.dddd, user-data: "My float"))),
    win);
end
```



Using Shortcut Bars

G2 provides a Windows shortcut bar, which you access through Telewindows.

Note Shortcut bars are only supported in Telewindows Next Generation (`twng.exe`).

The API supports:

- Creating shortcut bars that contain folders and icons, using one of two styles.
- Sending commands to enable and disable, clear, and destroy the shortcut bar.
- Changing the icon size to small or large icons, both interactively through a popup menu and by sending a command.
- Renaming the icon through a popup menu.
- A callback procedure that sends notification when the user selects an icon in a shortcut bar, clicks the right mouse button, and renames the icon.

For a description of the G2 system procedures for working with shortcut bars, see Shortcut Bar Views in User Interface Operations in the *G2 System Procedures Reference Manual*.

Creating a Shortcut Bar

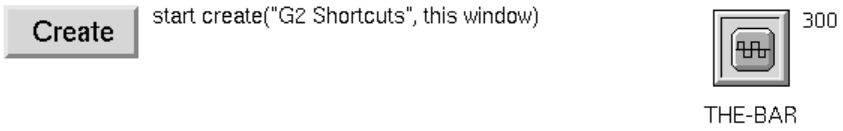
This procedure creates a shortcut bar with a given title text in a window. The procedure stores the shortcut bar handle in an integer parameter named the-bar. The sequence of folders is the return value of the test-folders procedure, which follows. The shortcut bar registers a callback procedure named callback.

```
create(title: text, window: class g2-window)
handle: integer;
folders: sequence;
begin
    folders = call test-folders();
    handle = call g2-ui-create-shortcut-bar(folders, the symbol CALLBACK,
        structure(width: 125, title: title), window);
    conclude that the-bar = handle;
end
```

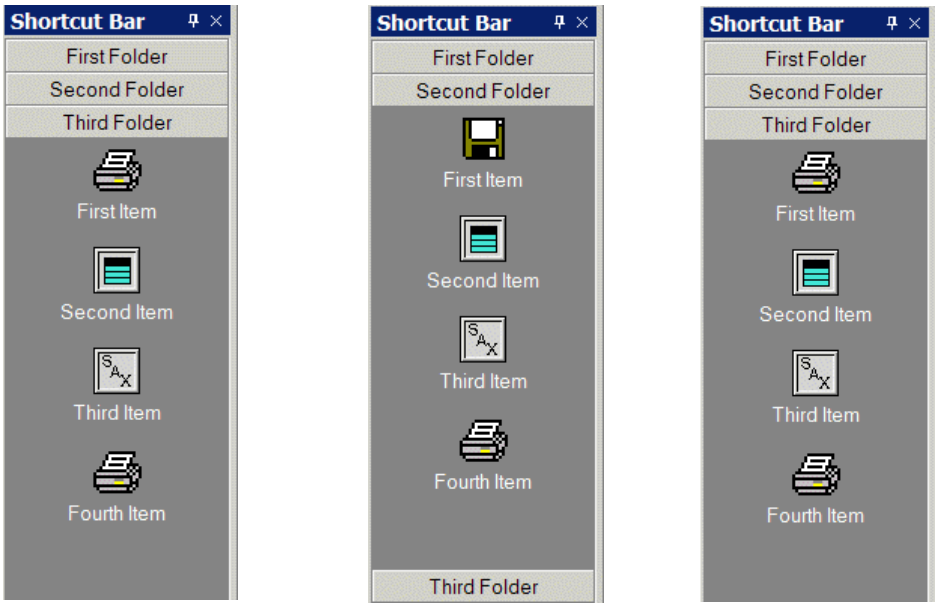
This procedure returns a sequence of structures that describes the folders and the items in each folder to show in the shortcut bar. The procedure creates two folders, one named Items and the other named Icons.

```
test-folders() = (sequence)
begin
    return
    sequence(
    structure(label: "First Folder",
        items: sequence(
            structure(label: "First Item", icon: the symbol PROCEDURE),
            structure(label: "Second Item", icon: the symbol G2-WINDOW),
            structure(label: "Third Item", icon: the symbol G2-LIST),
            structure(label: "Fourth Item", icon: the symbol GMS-PRINT-ICON))),

    structure(label: "Second Folder",
        items: sequence(
            structure(label: "First Item", icon: the symbol GMS-SAVE-ICON),
            structure(label: "Second Item", icon: the symbol USER-MENU-CHOICE),
            structure(label: "Third Item", icon: the symbol SAX-PARSER),
            structure(label: "Fourth Item", icon: the symbol GMS-PRINT-ICON))),

    structure(label: "Third Folder",
        items: sequence(
            structure(label: "First Item", icon: the symbol GMS-PRINT-ICON),
            structure(label: "Second Item", icon: the symbol USER-MENU-CHOICE),
            structure(label: "Third Item", icon: the symbol SAX-PARSER),
            structure(label: "Fourth Item", icon: the symbol GMS-PRINT-ICON)))
    );
end
```

Here is a button that creates a shortcut bar named G2 Shortcuts, and the integer parameter that stores the shortcut bar handle:



Here are the three folders of the resulting shortcut bar:



Using the Listbar Style

The shortcut bar supports the listbar style, which has these features:

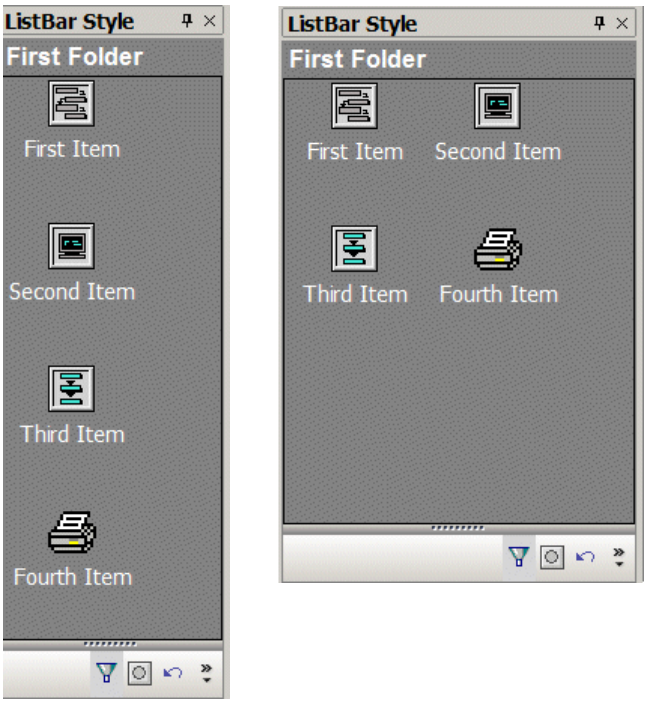
- The ability to show multiple icons per row.
- Icons with tooltips for folders.
- The ability to control the number folders that are visible.

To use the listbar style, provide the style attribute in the *options* structure in the g2-ui-create-shortcut-bar system procedure. For example:

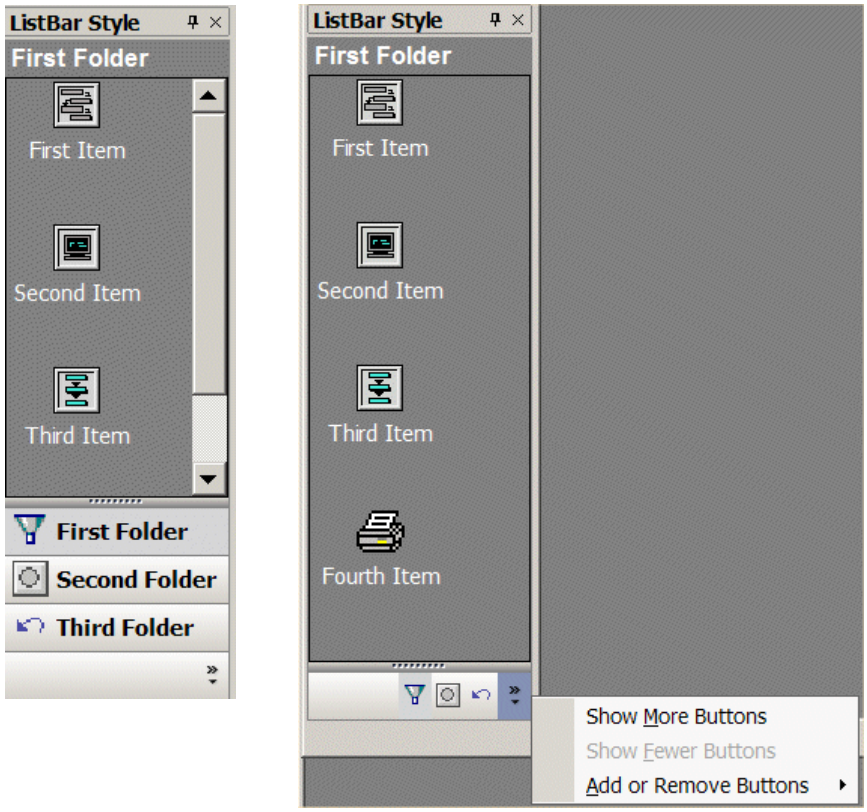
```
create(title: text, window: class g2-window)
handle: integer;
folders: sequence;
begin
    folders = call test-folders();
    handle = call g2-ui-create-shortcut-bar(folders, the symbol NONE,
        structure(width: 125, title: title, style: the symbol LISTBAR), window);
    conclude that THE-BAR = handle;
end
```

The other option for the style attribute is default.

Here is the shortcut bar with the listbar style showing a single column and multiple columns:



Here is the shortcut bar with the folder tabs showing and with the Configure Buttons menu:



Displaying Arbitrary Views in a Listbar Style Shortcut Bar

You can display an arbitrary view in the folders of a listbar-style shortcut bar, for example, a tree view or a dialog view. To support this feature, the container option of any of the g2-ui-create-*view* system procedures and the g2-ui-post-custom-dialog can be the handle of a listbar-style shortcut bar, in which case the neighbor option is the number of the folder within the listbar into which to create the view.

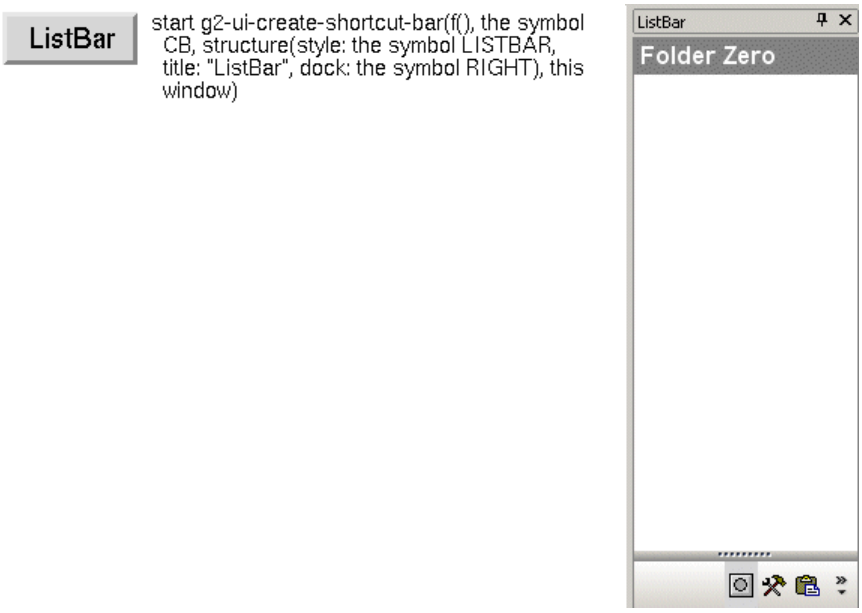
In addition, you can create shortcut bars and listbars with no items initially, then add native views to those folders later.

The following examples show how to create an empty listbar and add views to the folders incrementally.

This procedures creates the listbar with three folders, two of which are initially empty:

```
f()=sequence(
    structure(label: "Folder Zero", icon: cp-1),
    structure(label: "Folder One", items: sequence(
        structure(label: "First", icon: the symbol GMS-SAVE-ICON),
        structure(label: "Second", icon: CP-1),
        structure(label: "Third", icon: the symbol SAX-PARSER))),
    structure(label: "Folder Two", icon: the symbol GMS-PASTE-ICON))
```

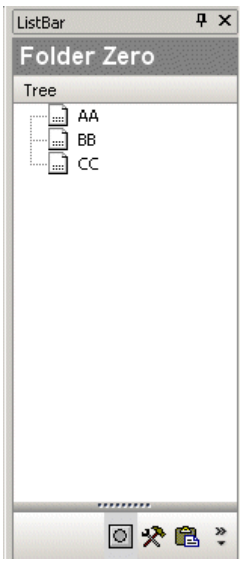
The following button creates the listbar:



This button adds a tree-view to the first folder in the listbar:

Tree

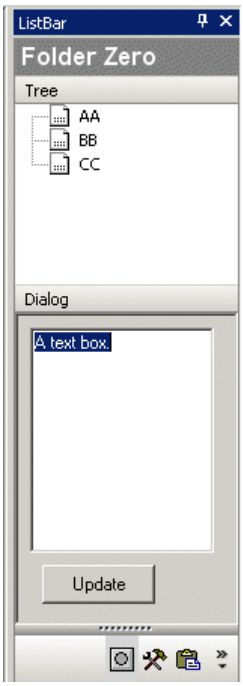
```
start g2-ui-create-tree-view("Tree", the
  symbol CB, structure(height: 120, container:
  "ListBar", tree: sequence("AA", "BB", "CC")),
  this window)
```



This button adds a dialog to the first folder in the listbar:

Dialog

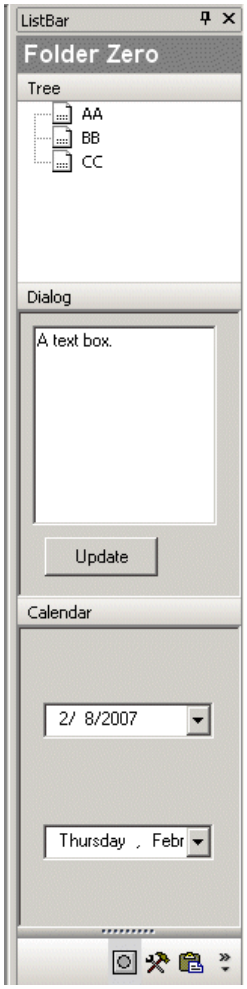
```
start g2-ui-post-custom-
  dialog(structure(dialog-title: "Dialog", dialog-
  width: 100, dialog-height: 120, resizable: true,
  container: "ListBar", components: ok(),
  dialog-update-callback: cb-update), false,
  this window)
```



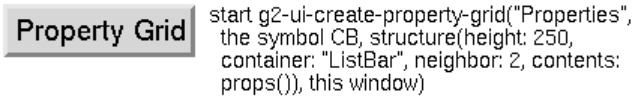
This button adds another dialog to the first folder in the listbar:

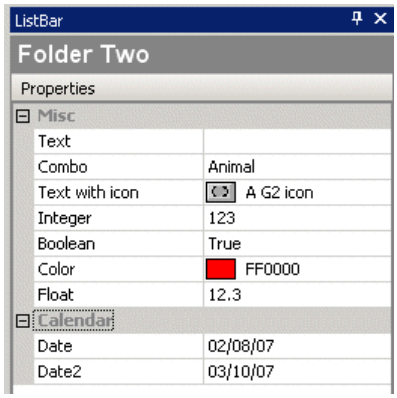
Dialog

```
start g2-ui-post-custom-
  dialog(structure(dialog-title: "Calendar",
  dialog-width: 310, dialog-height: 120,
  container: "ListBar", components: calendar(),
  dialog-update-callback: cb-update), false,
  this window)
```



This button adds a property grid to the third folder in the listbar:





Example Callback: Shortcut Bar

This callback procedure is called when the user clicks an item in a shortcut bar, which changes the text of a message named msg:

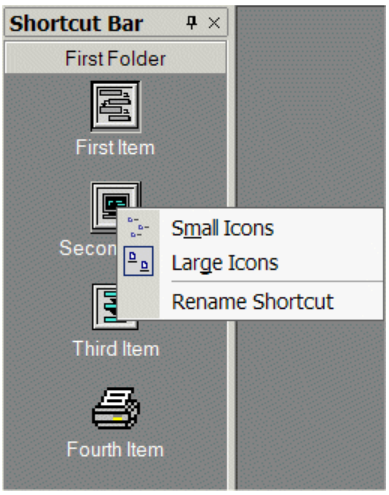
```
callback (event: symbol, Win: class g2-window, control: integer, item: value, info:
structure, user-data: value)
begin
    change the text of MSG to "[event] Control [control] Item @"[item]@" info [info]
        user-data: [user-data]";
end
```

Here is the result of clicking the First Item icon in the First Folder:

```
ITEM-SELECTED Control 104 Item "First Item" info
structure (FOLDER: 0,
  ITEM: 0) user-data: structure (LABEL: "First Item",
  ICON: the symbol PROCEDURE)
```

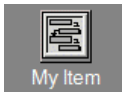
Interacting with Items in the Shortcut Bar

You can click the right mouse button on an item in a shortcut bar to display a popup menu for changing the icon size and renaming the icon:



When you click the right mouse button on an item in the shortcut bar, and you choose Small Icons, Large Icons, or Rename Shortcut, the shortcut bar sends the right-click event to the callback procedure. When you enter a new label for the shortcut, the shortcut bar sends the item-renamed event to the callback.

Here is the result of renaming the First Item in the First Folder to My Item:




```
ITEM-RENAMED Control 104 Item "My Item" info
structure (FOLDER: 0,
  ITEM: 0) user-data: structure (LABEL: "First Item",
  ICON: the symbol PROCEDURE)
```

Changing the Icon Size

The following procedure calls change the size of the icons in the shortcut bar named by the-bar:

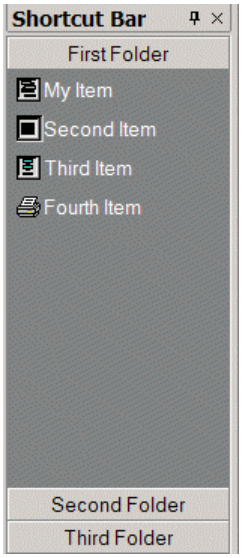
start g2-ui-manage-shortcut-bar (the symbol SMALL-ICONS, the-bar, false, this window)

start g2-ui-manage-shortcut-bar (the symbol LARGE-ICONS, the-bar, false, this window)

Small Icons — start g2-ui-manage-shortcut-bar (the symbol SMALL-ICONS, the-bar, false, this window)

Large Icons — start g2-ui-manage-shortcut-bar (the symbol LARGE-ICONS, the-bar, false, this window)

Here is the shortcut bar with small icons:



Disabling and Enabling a Shortcut Bar

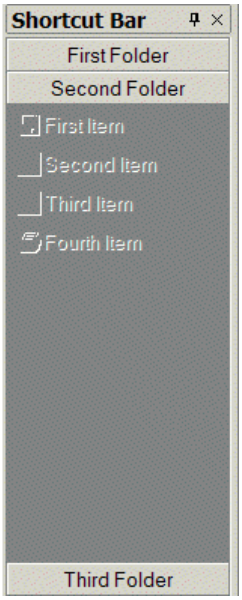
The following procedure calls disable the icons in the second folder (index = 1) in the shortcut bar named by the-bar:

```
start g2-ui-manage-shortcut-bar (the symbol DISABLE-FOLDER, the-bar, 1,
    this window)

start g2-ui-manage-shortcut-bar (the symbol ENABLE-FOLDER, the-bar, 1,
    this window)
```

Disable 2nd Folder — start g2-ui-manage-shortcut-bar (the symbol DISABLE-FOLDER, the-bar, 1, this window)

Enable 2nd Folder — start g2-ui-manage-shortcut-bar (the symbol ENABLE-FOLDER, the-bar, 1, this window)

Here is the shortcut bar with icons disabled:



Clearing a Shortcut Bar

The following procedure call clears all folders and items in the shortcut bar named by the-bar:

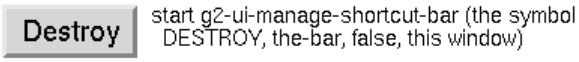
```
start g2-ui-manage-shortcut-bar (the symbol CLEAR, the-bar, false, this window)
```

Clear — start g2-ui-manage-shortcut-bar (the symbol CLEAR, the-bar, false, this window)

Destroying a Shortcut Bar

The following procedure call destroys the shortcut bar named by the-bar:

```
start g2-ui-manage-shortcut-bar (the symbol DESTROY, the-bar, false, this window)
```



Using Tree Views

G2 provides a Windows tree view, which you access through Telewindows.

Note Tree views are only supported in Telewindows Next Generation (`twng.exe`).

The tree view API supports these features:

- Creating and destroying the tree view.
- Populating the tree view with items or texts, using a nested sequence of structures that list the item and its children.
- Showing and hiding the tree view.
- Clearing the tree view.
- Selecting, expanding, collapsing, deleting, and inserting tree nodes.
- Callback procedure when the user clicks an item in the tree view.

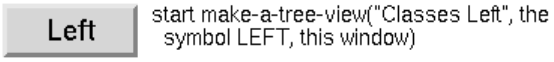
For a description of the G2 system procedures for working with tree views, see Tree Views in User Interface Operations in the *G2 System Procedures Reference Manual*.

Creating a Tree View

This procedure creates a tree view, using callback as the callback. The procedure stores the resulting integer handle in the integer-parameter named treeview:

```
make-a-tree-view(title: text, dock: symbol, win: class g2-window)
tv: integer;
begin
    tv = call g2-ui-create-tree-view(title, the symbol CALLBACK,
        structure(dock: dock), win);
    conclude that treeview = tv;
    allow other processing;
end
```

This action button creates a tree view named Classes in the current window, docked on the left edge of the window:



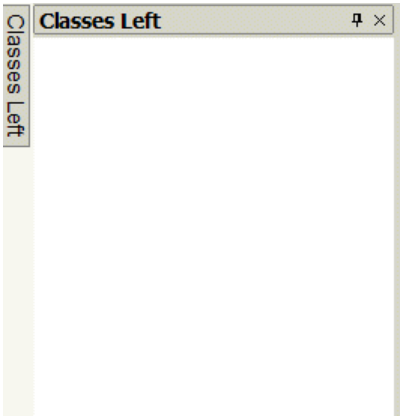

```
start make-a-tree-view("Classes Left", the symbol LEFT, this window)
```

Here is the integer parameter named treeview, which holds the current value of the tree view handle, which the example references:

Here is the tree view docked along the left side of the window:

You can click the close button to remove the tree view from the window, and you can click the hide/show toggle button (push pin icon) to hide and show the tree view. When the tree view is hidden, the title appears in a vertical or horizontal tab depending on where the tree view is docked. You can click the tab or simply hover the mouse over the tab to show the tree view if it is hidden. Once the tree view has been hidden once, the tab always appears.

This figure shows the tree view after it has been hidden and shown again:



Creating the Tree View as a Dialog Control

The g2-ui-create-tree-view system procedure allows the container attribute of the *options* structure to be a dialog handle, and the neighbor attribute to be a control ID for a control in a custom dialog. The tree view is placed in the dialog, replacing the existing control, which must be a label. Also, the dock attribute must be the symbol within.

For example, this procedure calls g2-ui-create-tree-view to replace the label control of a dialog with a tree-view:

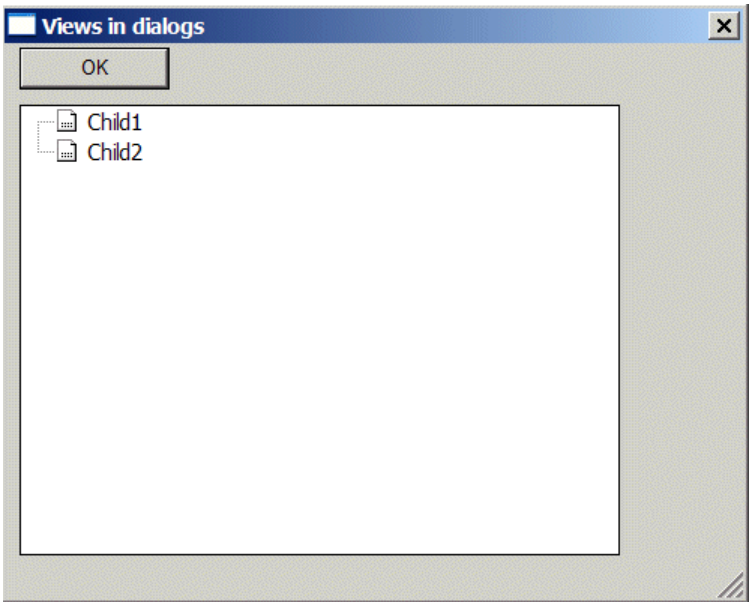
```
tree-view-in-dialog (Win: class g2-window)
dialog, control-id: integer;
begin
    dialog, control-id = call testdlg (win);
    call g2-ui-create-tree-view ("Tree", the symbol CB,
    structure(tree: sequence ("Child1", "Child2"),
    container: dialog,
    dock: the symbol WITHIN,
    neighbor: control-id),
    Win);
end
```

This procedure provides the dialog whose label control the tree-view control replaces:

```
testdlg(Win: class g2-window) = (integer, integer)
dialog: integer;

ok: structure = structure (control-type: the symbol push-button,  control-id: 1,
    height: 14, width: 50, left: 5, top: 1, response-action: the symbol OK,
    control-value: structure (text-value: "OK"));
```

```
    placeholder: structure = structure (control-type: the symbol LABEL, control-id: 2,
        height: 150, width: 200, left: 5, top: 20, border: true,
        anchor: the symbol TOP-LEFT-BOTTOM-RIGHT,
        control-value: structure(text-value: "Placeholder"));

    begin
        dialog = call g2-ui-post-custom-dialog(structure(dialog-title: "Views in dialogs",
            resizable: true, dialog-width:250, dialog-height:200,
            components: sequence (ok, placeholder),
            dialog-dismissed-callback: the symbol DISMISSED-CALLBACK,
            dialog-update-callback: the symbol UPDATE-CALLBACK),
            false, Win);
        return dialog, the control-id of placeholder;
    end
```

Here is the resulting dialog with a tree view control:

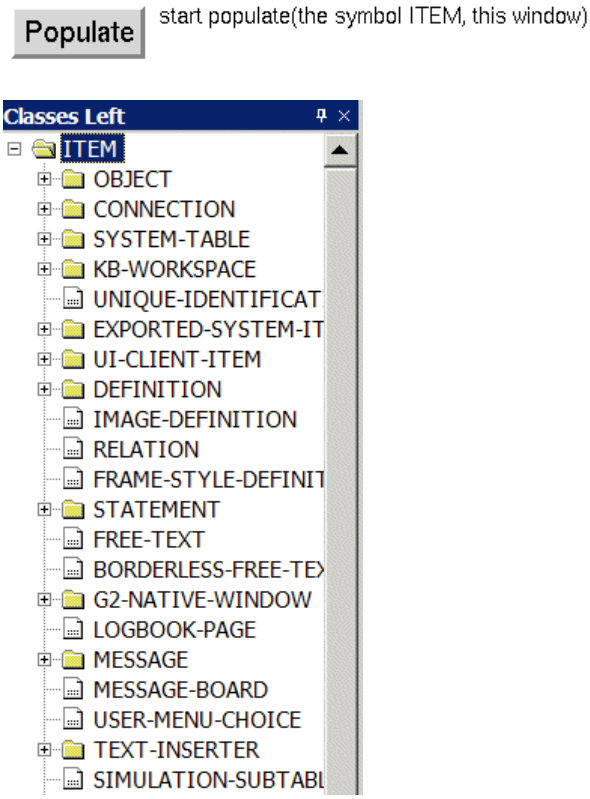


Populating a Tree View

This procedure populates a tree view with the G2 class hierarchy of a given class by calling g2-get-class-hierarchy, which returns a structure of the correct format. The procedure that populates the tree view associated with the current value of the integer parameter named treeview.

```
    populate(class: symbol, win: class g2-window)
    tree: value;
    begin
        tree = call g2-get-class-hierarchy (class);
        call g2-ui-populate-tree-view (treeview, tree, win);
    end
```

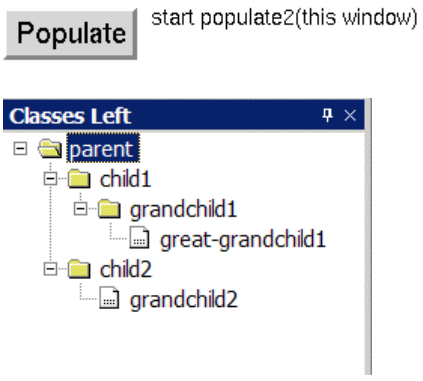
This action button populates the current tree view with the tree associated with the G2 class hierarchy of the item class. The top-level node in the tree is the value of the item-or-name attribute of the structure, and the children are the value of the children attribute of the structure. Here is part of the resulting tree view expanded one level to show the children:



Here is another example, which specifies the tree structure explicitly to show the format of the tree structure:

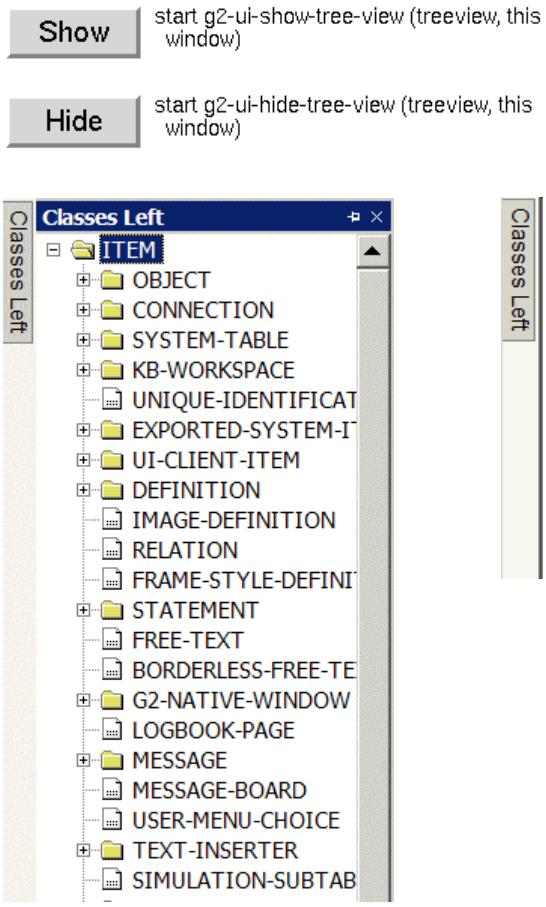
```
populate2(win: class g2-window)
tree: value;
begin
  tree = structure
    (ITEM-OR-NAME: "parent",
    CHILDREN: sequence (structure
        (ITEM-OR-NAME: "child1",
        CHILDREN: sequence (structure
            (ITEM-OR-NAME: "grandchild1",
            CHILDREN: sequence (structure
                (ITEM-OR-NAME: "great-grandchild1")))),
        structure
            (ITEM-OR-NAME: "child2",
            CHILDREN: sequence (structure
                (ITEM-OR-NAME: "grandchild2")))));
  call g2-ui-populate-tree-view (treeview, tree, win);
end
```

This button creates a tree view that looks like this:



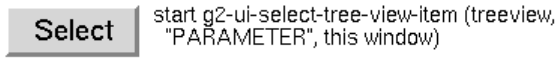
Showing and Hiding a Tree View

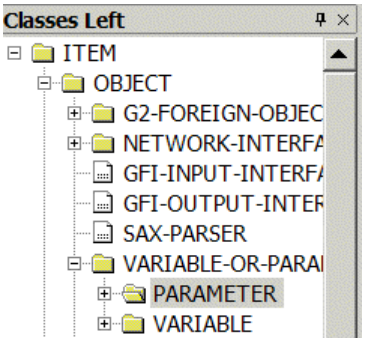
These buttons show and hide the tree view named by the value of the treeview integer parameter. The tree view is shown at the same level of expansion as it was before it was hidden, and the hidden tree view shows just the tab. Once the tree view has been hidden, the tab always appears.



Selecting Items in a Tree View

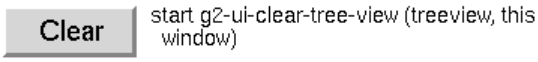
This button selects the "parameter" item in the tree view named by the value of the treeview integer parameter. Selecting a tree view item, either programmatically or interactively, invokes the registered callback for the tree view. For details, see Example Callback: Tree View.





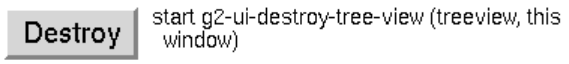
Clearing a Tree View

This button clears the contents of the tree view named by the value of the treeview integer parameter. The tree view still exists but has no items.



Destroying a Tree View

This button destroys the tree view named by the value of the treeview integer parameter:



Example Callback: Tree View

Here is a sample callback procedure that gets called when the user selects an element in the tree view or when an item is selected programmatically via the g2-ui-select-tree-view-item system procedure. The procedure changes the text of the result message to indicate the type event, the integer handle of the tree view control, the value of the selected item, and the user data for the selected item. If the event is right-click, then the procedure selects the item and calls the menu-choose procedure.

```
callback (event: symbol, Win: class g2-window, control: integer, item: value,
    info: structure, user-data: value)
begin
    change the text of RESULT to "[event] Control [control] Item [item] info [info]";
    if (event is RIGHT-CLICK) then
    begin
        call g2-ui-select-tree-view-item (treeview, item, Win);
        call menu-choose(sequence("One", "Two", "Three"), the X of info, the Y of
            info, win);
    end;
end
```

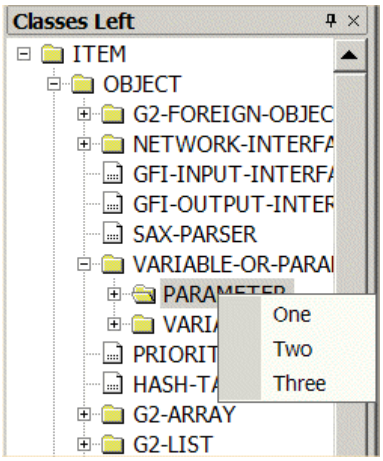
The menu-choose procedure posts a popup menu with a sequence of choices for the selected item, using cb-menu-choice as the callback when a menu choice is selected.

```
menu-choose (choices: sequence, x: integer, y: integer, win: class g2-window)
menu: integer;
choice: text;
begin
    menu = call g2-nms-create-menu(the symbol cb-menu-choose, win);
    for choice = each text in choices do
        call g2-nms-add-choice(menu, choice, choice, win);
    end;
    call g2-nms-manage-popup-menu(MENU, x, y, win);
end
```

Here is the result message after selecting the parameter element in the tree view, which indicates the event type (select), the control handle (7), the selected element (parameter), and the X-Y coordinates of the selected location, which is the value of the *info* argument to the callback (structure (X: -352, Y: 130)):

```
SELECT Control 77 Item PARAMETER Info
  structure (X: -367,
  Y: 88)
```

Here is result of clicking the right mouse button on an item in the tree view, which posts a popup menu:



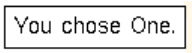
Here is the result message after right clicking the parameter item in the tree view, which indicates that the event type is right-click:

```
RIGHT-CLICK Control 77 Item PARAMETER Info
  structure (X: -568,
  Y: 273)
```

This callback procedure is executed when the user selects a menu choice from the popup menu. The procedure changes the text of the result message to indicate the selected choice or whether the menu was dismissed.

```
cb-menu-choose(window: class g2-window, menu: integer, choice: integer,
    path: sequence)
user-data: item-or-value;
begin
    if (choice = 0) then
        change the text of RESULT to "You dismissed the menu."
    else
        begin
            user-data = call g2-nms-get-key(choice, window);
            change the text of RESULT to "You chose [user-data].";
    end
end
```

Here is the result message after selecting the One choice in the popup menu:



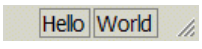
Using Status Bars

You can configure a status bar to include multiple panes, including text and icons. You use the system procedures to:

- Show and hide the status bar.
- Add, remove, and modify status bar panes.
- Configure various pane information, including text, icon, background color, foreground color, tooltips, and borders.

For a description of the G2 system procedure for creating status bars, see Status Bar Operations in User Interface Operations in the *G2 System Procedures Reference Manual*.

Here is an example of a status bar with multiple panes and the code used to create it:




```
start g2-ui-configure-status-bar
(structure
(callback: cb,
panes: sequence
    (structure(id: "X", text: "Hello", user-data: 123456),
    structure(id: "Y", text: "World"))), this window)
```

This example shows how to add a pane to the status bar:


```
start g2-ui-manage-status-bar
(the symbol ADD-PANE, structure(id: "X", text: "Again"), this window)
```

This example shows how to modify a pane by adding an icon:


```
start g2-ui-manage-status-bar
(the symbol MODIFY-PANE, structure(id: "X", icon: cp-1), this window)
```

This example shows how to modify the background color of a pane:


```
start g2-ui-manage-status-bar
(the symbol MODIFY-PANE, structure(id: "X", background-color: the symbol WHEAT),
this window)
```

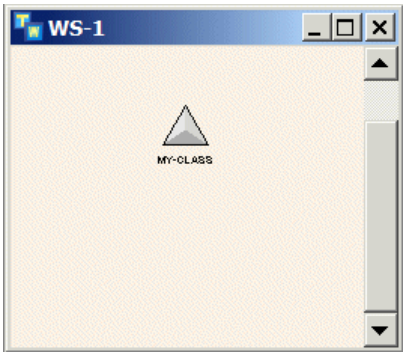
Using Workspace Views

You can create Windows views of kb-workspace instances, using a variety of display options. The options include all those returned by g2-ui-lookup-window-handle.

For a description of the G2 system procedure for creating workspace views and registering callbacks, see Window Handles and Views and Selection API in User Interface Operations in the *G2 System Procedures Reference Manual*.

The following code fragment shows ws-1 on the current window at 50% scale in the x and y dimensions, with a height of 200 pixels, and scrolled 50 pixels in the vertical dimension:

```
start g2-ui-create-workspace-view(ws-1, structure(x-scale: 0.5, y-scale: 0.5,
    height: 200, y-scroll-position: 50), this window)
```



This procedure calls g2-ui-lookup-window-handle to get the display information from a workspace, then restores the workspace, using the same display information:

```
restore-ws(ws: class kb-workspace, window: class g2-window)
handle: integer;
info, info2: structure;
begin
    info = call g2-ui-lookup-window-handle("[the name of ws]", window);
    post "Before: [info]";
    hide ws;
    allow other processing;
    wait for 5 seconds;
    handle = call g2-ui-create-workspace-view(ws, info, window);
    info2 = call g2-ui-lookup-window-handle(handle, window);
    post "After: [info2]";
end
```

Here is the resulting info structure:

```
#24  11:16:48 a.m.  After: structure (HANDLE:
 104,
 CLASS: the symbol G2-WORKSPACE-VIEW,
 TITLE: "WS-1",
 VISIBLE-CAPTION: true,
 STATE: the symbol NORMAL,
 CONTAINER: the symbol MDI-CHILD,
 LEFT: 244,
 TOP: 283,
 WIDTH: 366,
 HEIGHT: 500,
 CLOSEABLE: true,
 RESIZEABLE: true,
 MINIMIZEABLE: true,
 MAXIMIZEABLE: true,
 X-SCALE: 0.5,
 Y-SCALE: 1.0)
```

Using Tabbed MDI Mode

You can configure Telewindows to use tabbed MDI mode for displaying workspaces in tab pages.

Note Tabbed MDI mode is only supported in Telewindows Next Generation (`twng.exe`).

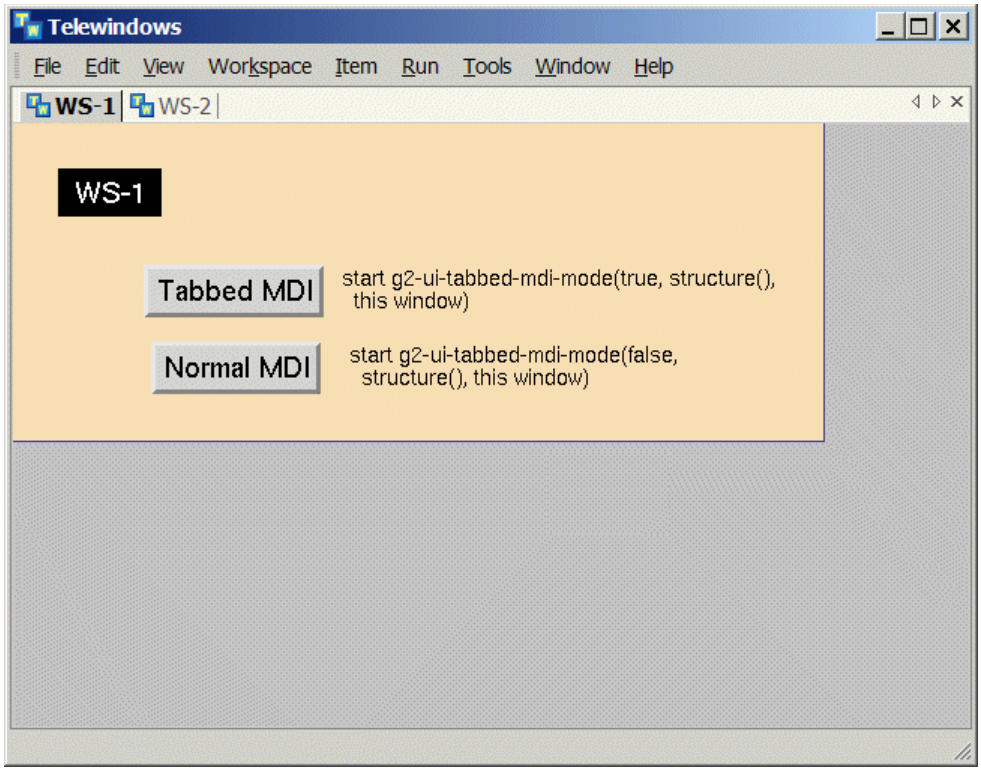
In tabbed MDI mode, when you show a workspace, either interactively or programmatically, the workspace appears as a tab page in the overall window and fills the window. Workspaces include any workspace that appears in its own window, such as KB workspaces, attribute tables, and the G2 Text Editor. You can interactively switch the order of the tabs by dragging, scroll through the tabs by clicking left and right arrows, and close individual workspaces.

For a description of the G2 system procedure for using tabbed MDI mode, see Window Handles and Views in User Interface Operations in the *G2 System Procedures Reference Manual*.

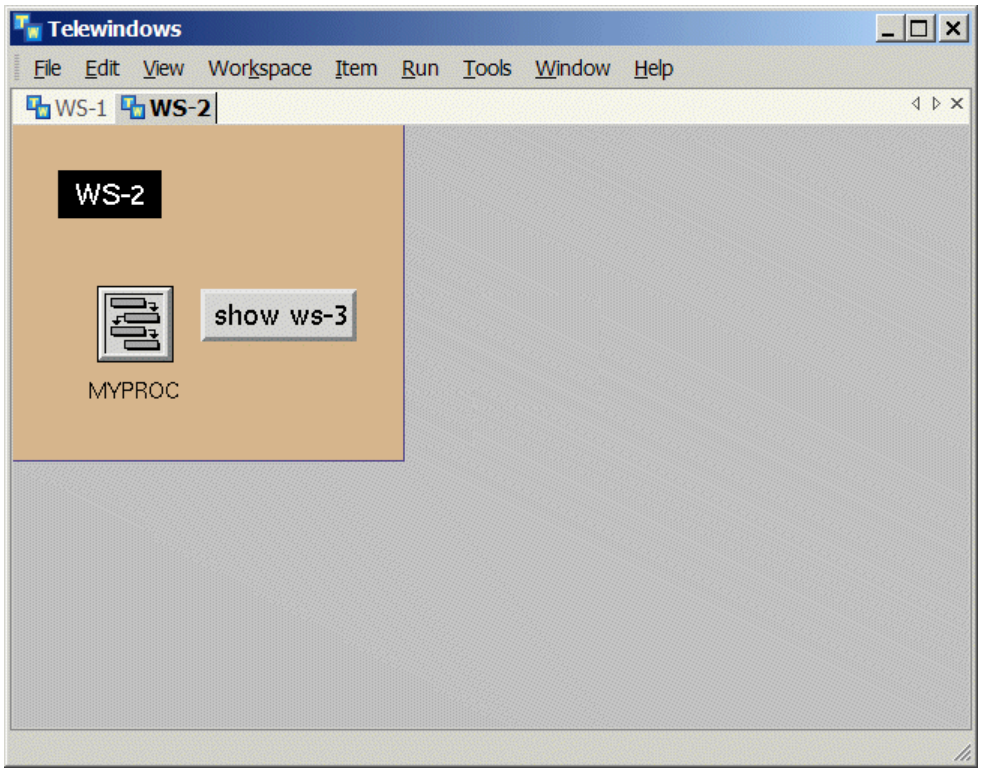
For example, this code fragment causes the current window to use a tabbed MDI mode:

```
start g2-ui-tabbed-mdi-mode(true, structure(), this window)
```

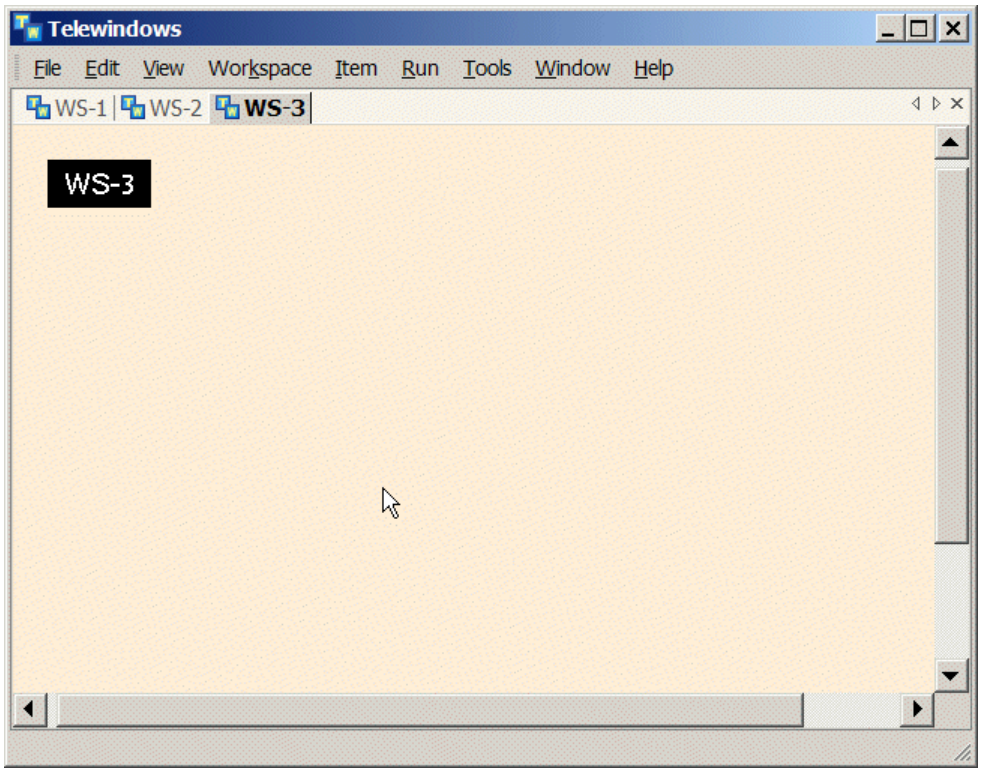
This figure shows the result of using tabbed MDI mode with two workspaces visible. The first tab shows ws-1, which has buttons for controlling the tabbed MDI mode. Notice that the workspace is shrink wrapped, yet the tab page fills the entire window.



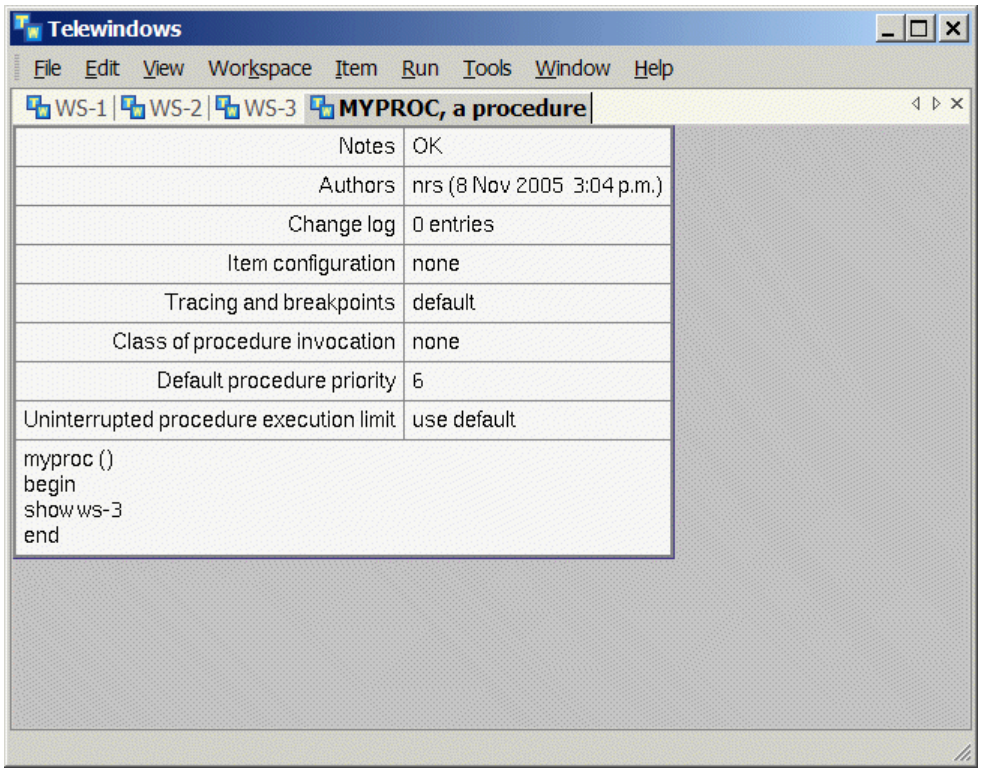
The second tab show ws-2, which has a procedure and button for programmatically showing ws-3:



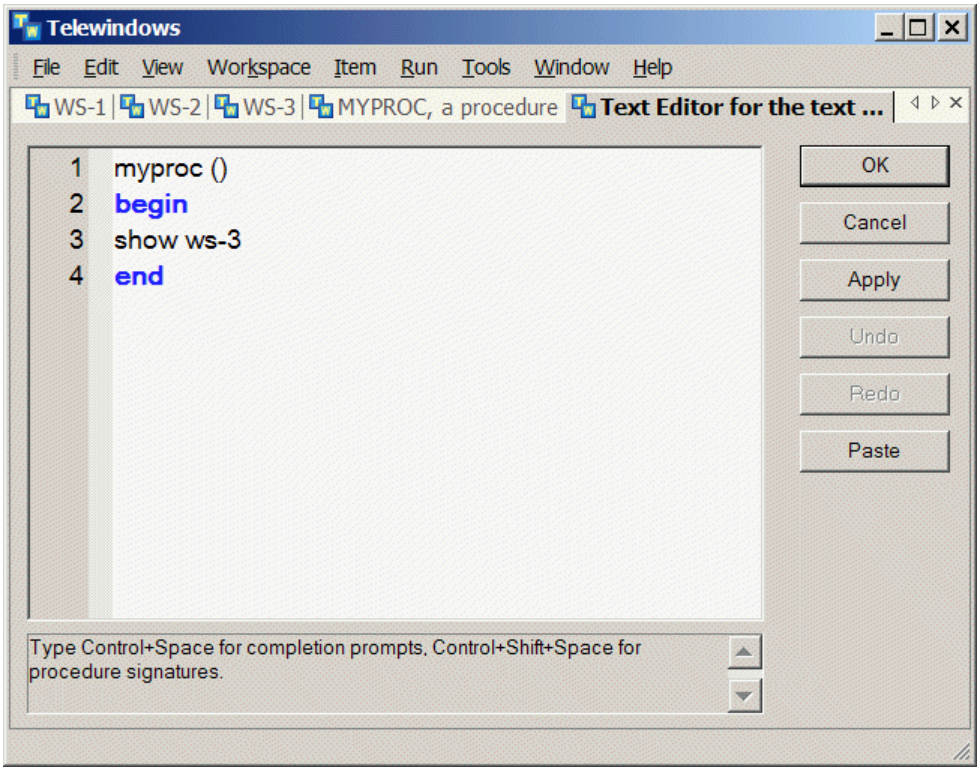
Clicking the show ws-3 button shows ws-3 in a tab page. Notice that this workspace has scroll bars.



On the ws-2 tab, double-clicking the procedure shows its attribute table in a tab page, where the tab label corresponds to the procedure name and class:



Double-clicking the text of the procedure shows the G2 Text Editor in a tab page, where the tab label indicates the attribute being edited. The tooltip shows the complete text: Text Editor for the text of MY-PROC, a procedure.



Part VI

Editors and Facilities

45

The Text Editor

Describes how to create text items and how to use text inserters.

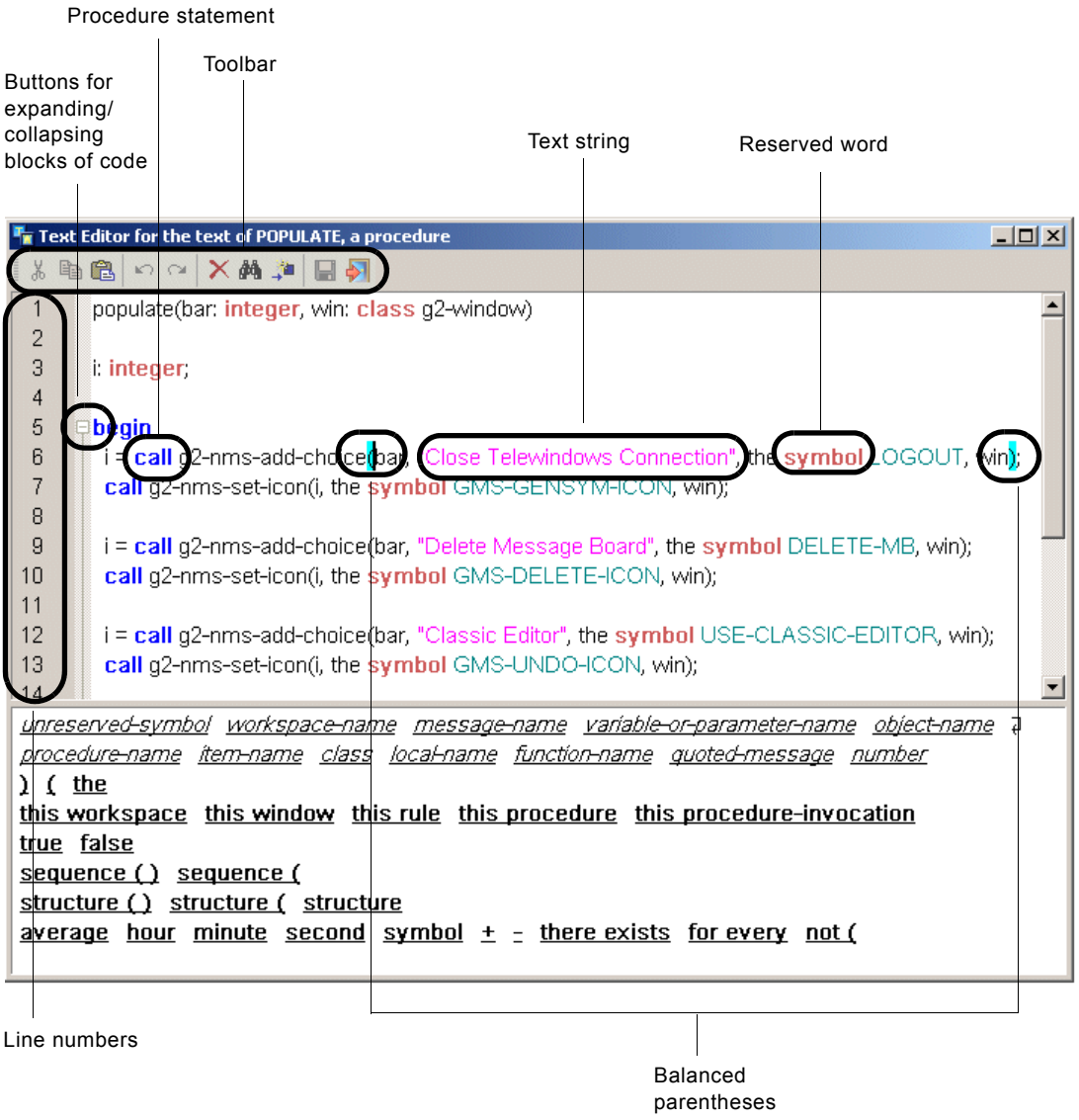
Introduction

G2's interactive Text Editor lets you enter and edit text for statements, rules, functions, attributes, and all other textual components of G2. Whenever you choose to create a new statement, edit a statement, select any text box, or add or edit any other text item, G2 displays the editor.

The editor includes prompts that guide you through the specification of the statement. The editor supports standard editing commands, including cutting and pasting to and from a clipboard. G2 also provides a scrollable editor for entering longer series of statements.

In addition to the basic editor functionality provided in G2, Telewindows provides a Windows text editor that supports numerous features, including line numbers, buttons for expanding and collapsing blocks of code, use of color for text strings, reserved words, and procedure statements, balanced parentheses,

and a toolbar, which includes buttons for cut, copy, paste, undo, redo, delete, find and replace, go to item, save, and save and exit. For example:



This chapter describe the basic functionality of the classic G2 text editor. For information on using the text editor in Telewindows, see Editing Text in Using Telewindows in the *Telewindows User's Guide*.

Text Editor Features

The G2 Text Editor is a special G2 workspace with these features:

- **Natural language prompts** guide you as you create statements, which you enter by clicking the item or typing it.
- **Text Editor buttons** allow you to accept and cancel your edits, as well as to undo and redo the last edit.
- **Edit Operations buttons** will display the edit buttons that are appropriate to the current editing context instead of displaying the default Text Editor buttons.
- **Edit operations menu** offers standard editing operations on selected text, including cut and paste.
- **Editor keystroke commands** allow you to perform standard editor operations from the keyboard.
- **Text Editor buttons** allow you to accept and cancel your edits, as well as to undo and redo the last edit.
- **Scrollable workspaces** allow you to enter or edit statements or compiled attributes.

You can enter text into the editor by typing the text from the keyboard or by selecting the prompts that appear at the bottom of the Text Editor workspace.

You can enter text into the editor by selecting any text anywhere on any displayed workspace. You can also insert text into the editor by cutting and pasting to and from a scrapbook, and by using a type of text item called a **text inserter**.

In addition, you can enter any character in the Unicode character set, from any of G2's supported natural languages, into the editor. You can do this regardless of the current language setting.

Opening the Text Editor

You can open the editor several ways, depending on what you are editing. You can open one or more editors by using any of the following techniques.

To open the editor:

➔ Click on an existing piece of text in the KB, for example, a rule, function, attribute value in a table, attribute display, free text, and so on.

or

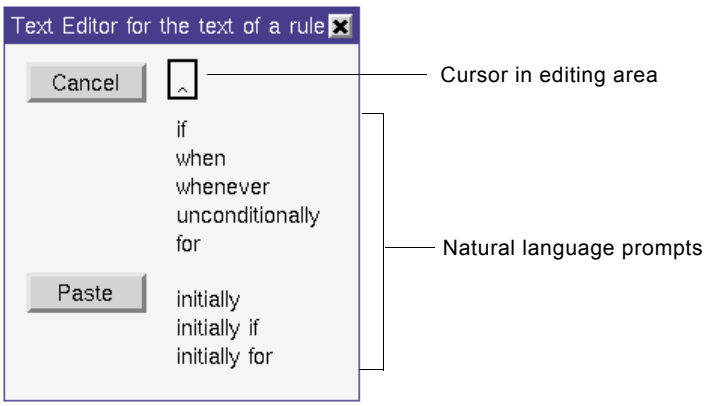
➔ Click on an item to display its menu, and select the edit menu choice.

or

➔ Create a new rule; new definition of type function, foreign function, generic formula, remote procedure declaration, or language translation; or new free text, including a text inserter.

When you edit existing text, G2 places the cursor within the editor's **editing area** at the position where you select the text.

For example, the following figure shows the editor as you create a new rule by using KB Workspace > New Rule. Notice the natural language prompts below the type-in box, and the editor buttons on the left side of the workspace.



Note Because selecting an existing piece of text automatically displays the editor, you must select the margin of an existing piece of text, such as a rule, statement, or piece of free text, in order to display its table or move it.

Setting the Minimum Width of the Editing Area

You can set minimum width of the Text Editor's editing area. The minimum width of the editing area is the greater of:

- G2's default for the minimum width.
- The current value of the minimum-width-for-edit-box attribute in the Editor Parameters system table.

For the standard Text Editor, G2's default for the minimum width of the editing area depends on the text you select to edit. For the scrollable editor, G2's default minimum width of the editing area is approximately 500 workspace units.

The default value of the minimum-width-for-edit-box attribute is 0 workspace units.

To change the minimum width of the editing area:

➔ Edit the value of the minimum-width-for-edit-box attribute in the Editor Parameters system table.

For the minimum-width-for-edit-box attribute's value to affect the minimum width of the editing area for the scrollable editor, set the value to 500 or more.

Tip To see even more text in the editor, display text in a smaller font size. To do this, set the font-for-editing attribute to small in the Fonts system table.

Configuring Editor Menu and Button Options

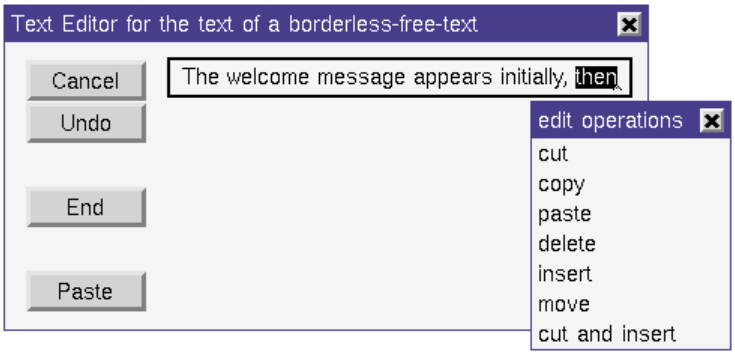
You can configure your editing operations environment by specifying values for two attributes on the Editor Parameters system table:

- pop-up-edit-operations-menu
- buttons-for-edit-operations

By default, the values of these attributes are yes and no, respectively, indicating that:

- The Edit Operations popup menu appears whenever you select text.
- The default edit buttons are displayed, instead of displaying those buttons that are appropriate to the current context.

Selecting text in the Editor displays the edit operations menu:

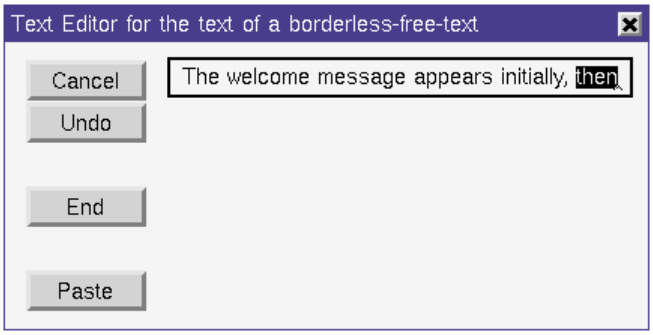


To suppress the edit operations menu:

1 Open the Editor Parameters system table.

2 Specify no for the pop-up-edit-operations-menu attribute.

Selecting text in the Editor no longer displays the edit operations menu:



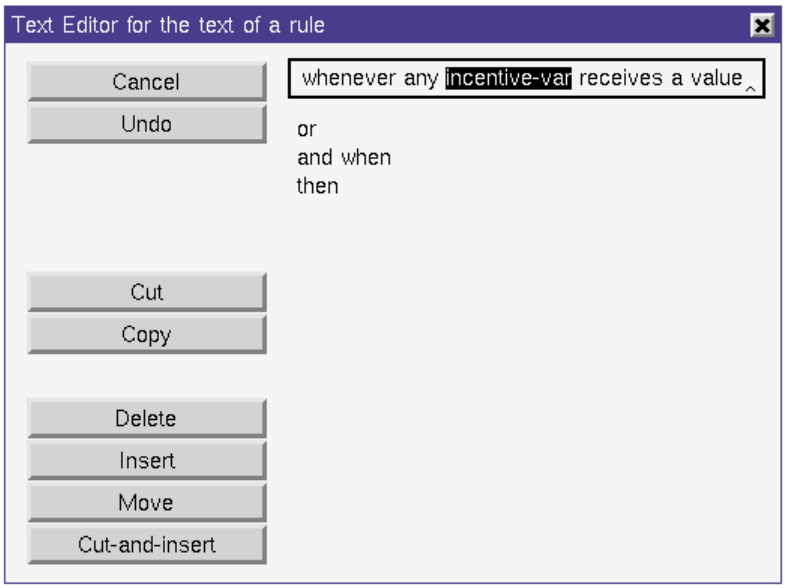
Note When text is highlighted, using the left, right, top, bottom, home, or end navigation keys does not delete any text. Instead, the selected text is no longer highlighted and the cursor moves to the position indicated by the navigation key.

To display context-sensitive Edit Operations buttons:

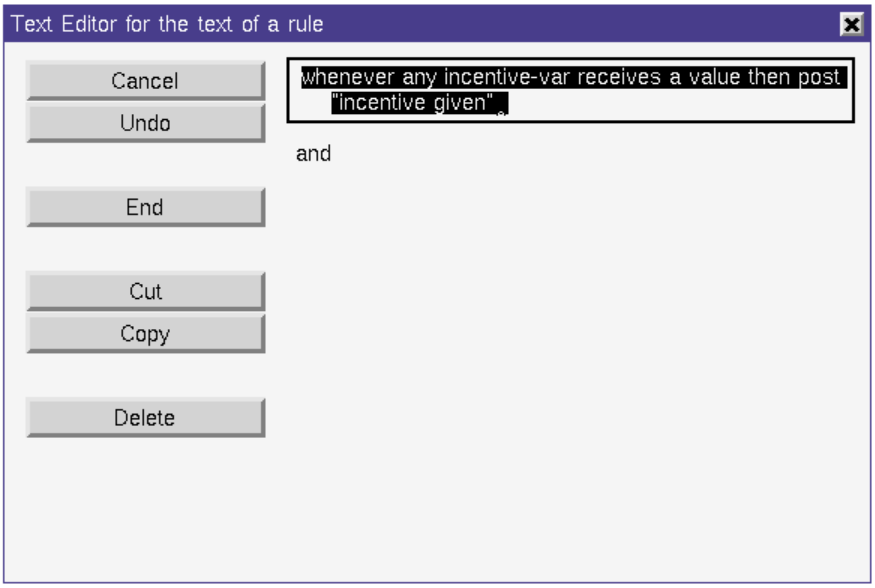
1 Open the Editor Parameters system table.

2 Specify yes for the buttons-for-edit-operations attribute.

Once you have changed this attribute to yes, selecting a piece of text displays the edit buttons that are appropriate to the current context. For example, selecting part of the text displays these edit operations buttons:



Selecting all of the available text displays these buttons:



Note To have the full range of button *or* popup-menu choices described in this chapter, specify yes for *either* the buttons-for-edit-operations attribute *or* the pop-up-edit-operations-menu attribute. The buttons-for-edit-operations attribute is only operational when the pop-up-edit-operations-menu is set to no.

Entering Text

You can enter text by using one of these techniques:

- Typing text at the current cursor location.
- Selecting text and replacing it directly by typing one or more characters.
- Clicking on one of the **natural language prompts** at the bottom of the editor workspace, which guide you in entering valid statements.
- Selecting argument names and types from a workspace which automatically appears when you enter the name of a defined procedure or function.
- Selecting any visible text.
- Selecting text from a text inserter.

As you enter text using any of these methods:

- G2 displays the next allowable term at the bottom of the workspace. You can use these prompts to guide you in entering valid statements.

- G2 flags text that is syntactically incorrect as you enter it by showing ellipses in the text as you type.
- You can undo and redo the last edits that you made.

When you accept the text by closing the editor, the effects are immediate.

Entering Text within the Text Editor

You can enter text within the editor three ways.

To enter text by using the keyboard:

➔ Type the text at the current cursor location.

To replace text:

1 Select the text to be replaced by dragging the mouse cursor over it.

2 Without cutting, replace the selected text directly by typing one or more characters.

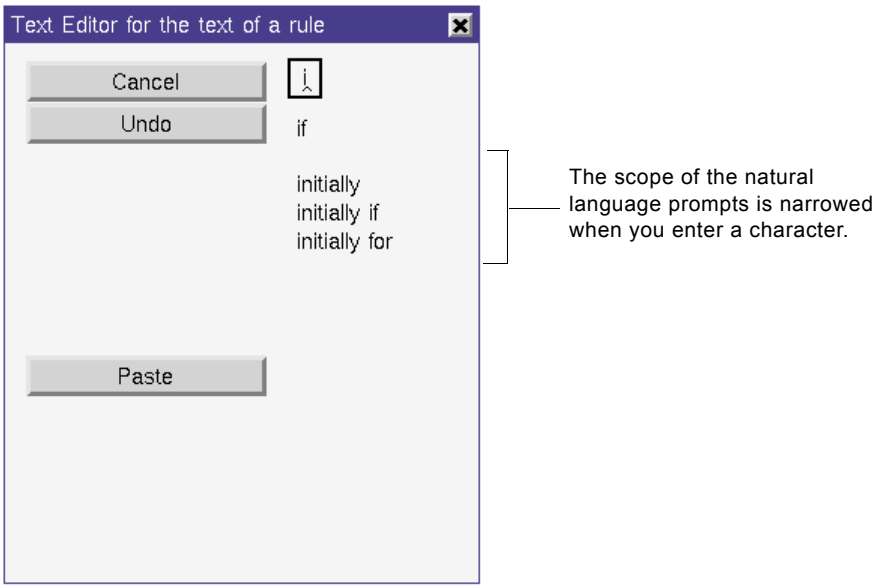
To enter text by using the natural language prompts:

➔ Click on a prompt at the bottom of the workspace.

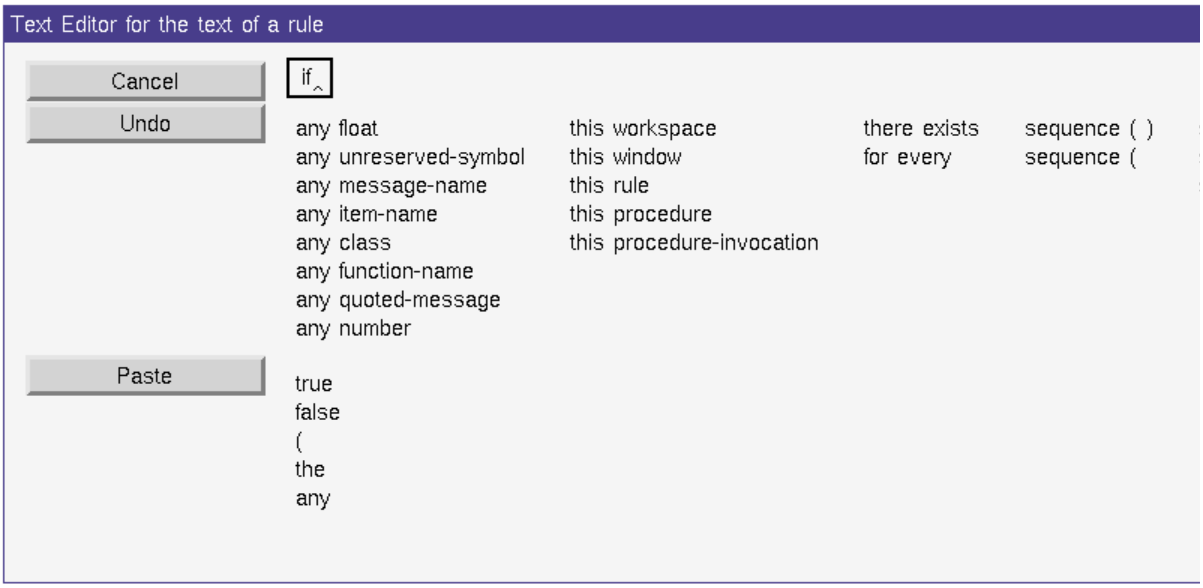
When you enter text by selecting a natural language prompt, G2 also inserts the space following the word. If you have already entered the word without the space and you click on the prompt, G2 inserts the space. If you have entered only part of the word and you click on the prompt, G2 completes the word and inserts a space.

Each time you enter a character in the editor, G2 updates the natural language prompts to indicate the current allowable syntax. For example, when creating a

new rule, if you enter the letter “i” in the editor to enter the word if, G2 updates the prompts to show the two prompts that begin with the letter “i”:



When you have entered a valid word in the editor *and entered a space following the word*, G2 again updates the prompts to show the valid syntax. For example, after you enter the word if and the following space, G2 displays this editor:



The prompts now contain two types of items: generic class-oriented prompts at the top, and natural language prompts at the bottom. The class-oriented prompts indicate the type of item you can enter at this point in the statement. Entering a Class Name describes the two ways of entering class names.

Entering Text by Selecting Visible Text

You can insert any piece of visible text in your KB into the editor. For example, you can insert attribute names from a table, a statement from a rule, free text, or any other visible text in the system.

To enter text by selecting existing text, do any of the following:

➔ Drag the mouse over any piece of text, and release the mouse. The text appears in the text editor exactly as it appears on the screen.

or

➔ Click on the name of an attribute in the left-hand column of any attribute table. The attribute name appears in the editor in lowercase characters, with any spaces converted to dashes, and any special characters escaped with an at-sign (@).

or

➔ Click on the name of any item on a workspace. The name appears in the text editor in lowercase.

Note Be sure to click, rather than drag, an attribute or item name. Selecting text by sliding the cursor across it places the attribute or item name in the editor, but will not convert any spaces to dashes or change uppercase characters to lowercase.

You can also use a type of text item called a **text inserter** as a way of inserting text into the editor. Depending on the type of text inserter, you can insert a complete piece of text, word, character sequence, or character.

For information about how to create and use text inserters, see Using Text Inserters to Insert Text into the Text Editor.

Entering a Class Name

When a statement requires a class name, you have two options:

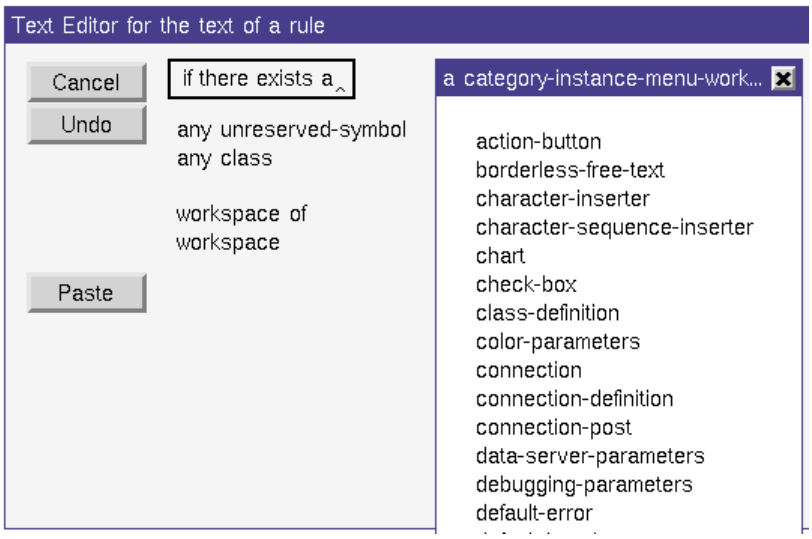
- You can display a list of available classes and select a class from the list.
- You can use the natural language prompts to display the classes whose name begins with the specified letters.

To enter a class name from a menu:

1 Click on one of the class-oriented prompts toward the top of the screen.

G2 displays a temporary workspace that includes all classes of the specified type defined in the current KB. For example, you might create a rule that tests

for the existence of a certain class. Clicking on any class displays a workspace such as the following:



2 Click on one of the classes in the list to insert it at the current cursor location and delete the temporary workspace.

   a If you find more classes than fit on the screen, move the workspace up to see the other classes.

   b If you want the workspace to remain after you select a class, name the workspace.

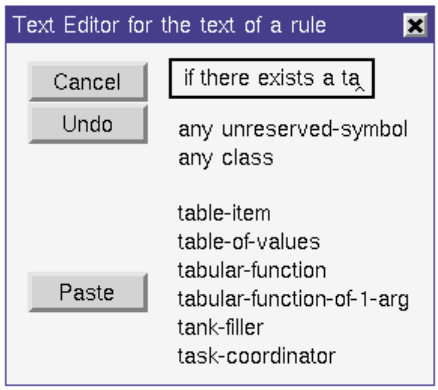
      You can then hide and retrieve the workspace like any other. The class names workspace is a non-KB workspace, which means G2 does not save it with the KB.

To enter a class name by using the natural language prompts:

1 Begin typing the name of a class in the editor.

   For example, when creating a rule that tests for the existence of a class, you might enter the letters "ta" in the editor after the expression.

G2 displays all the classes that begin with these letters, including system-defined classes and user-defined classes, as shown.



2 You can either continue typing the class name at the keyboard, or you can click on the name of a class that G2 displays in the workspace.

If you click on the name of the class, G2 inserts only the characters that are necessary to complete the name.

Controlling the Number of Classes that G2 Displays

By default, G2 displays a maximum of 50 classes in the natural language prompts that appear at the bottom of the editor workspace. Thus, when you type the first letter of a class, you will typically see a list of all system-defined and user-defined classes whose name begins with that letter. If you have defined more than fifty classes that begin with a particular letter, you will only see the first fifty.

You can control the number of classes that G2 displays in the natural language prompts when typing a class name into the editor.

To set the maximum number of class names that G2 displays:

➔ Edit the maximum-number-of-names-in-menus attribute in the Editor Parameters system table.

The default is 7.

Configuring the Grammar Prompts that G2 Displays

You can use the g2-ui-launch-editor system procedure to configure the prompts that appear in the text editor, both the classic G2 text editor and the Windows text editor available through Telewindows.

For details, see Editor Operations in User Interface Operations in the *G2 System Procedures Reference Manual*.

Using Text Editor Procedure and Function Signature Prompting

When you type the name of a procedure or function in the text editor, G2 prompts you with the signature of that procedure or function. In the case of multiple methods with the same name, G2 prompts you with the signatures of all methods with that name.

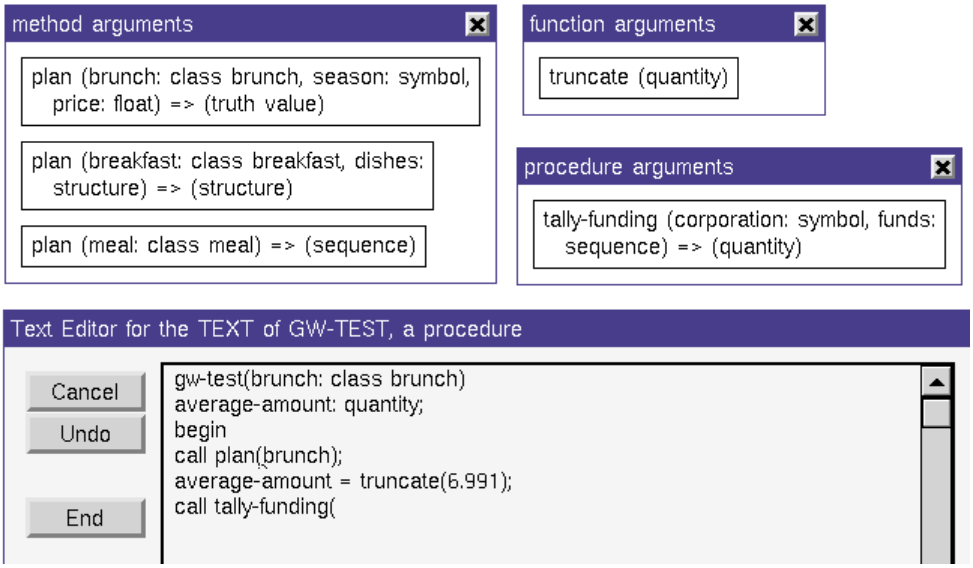
This facility can be disabled by specifying no for the show-procedure-signatures? of the Editor Parameters system table. The default value for this attribute is yes.

G2 prompts by putting up a workspace that displays the argument names for a function; and the argument names, argument types, and return values for a procedure. When you type a defined procedure or function name followed by a left parenthesis, the signature workspace appears in the upper right-hand corner of the G2 window. The workspace remains there as long as the cursor is within the opening and closing parentheses. It automatically disappears when you type the closing right parenthesis, and reappears when you relocate the cursor within the parentheses.

If the procedure or function has no arguments, G2 displays:

(no arguments)

Here is an example of the text editor open for editing a procedure. The argument workspaces display the arguments and types for a G2 system procedure and the argument for a user-defined function.



Undoing and Redoing the Last Edit

You can undo and redo the last edit by using the Undo and Redo button or the Alt + u and Alt + r commands, respectively.

For example, if you select a prompt from the bottom of the workspace to enter it, selecting Undo removes the word; if you type a character, Undo deletes the last character you typed. By repeatedly selecting Undo, you can undo all of the edits you have made in the editor.

If you delete a word and select Undo, selecting Redo deletes the word again. By repeatedly selecting this menu choice, you can redo all of the edits you undid. You can redo a series of edits only immediately after undoing them.

Controlling the Number of Edits You Can Undo

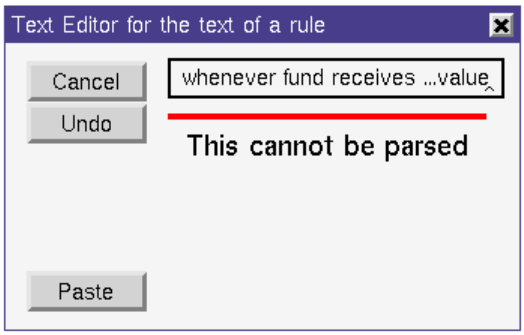
You can control the number of edits that G2 remembers when you use the Undo button in the editor. By default, G2 remembers the last 100 edits you perform.

To set the maximum number of undo's:

➔ Edit the maximum-number-of-undos-to-remember attribute in the Editor Parameters system table.

Correcting Errors in the Editor

If you enter text or select a prompt that makes the statement syntactically incorrect, the editor displays an ellipses (...) at the location of the error. While you can continue editing the text, G2 does not accept the text when you end the editing session, as described below. For example:



Ending the Editing Session

When you end an editing session, the edits you make are effective immediately. For example, if you change the scan interval for a rule, the rule starts scanning at the new rate immediately. G2 compiles or recompiles as needed to reflect any changes.

To end an editing session:

➔ Press the Return key or click the End button, depending on the context.

G2 accepts your edits and closes the editor.

When using the scrollable editor, you can click the End button or enter the Control + Return command to end the editing session (see Using the Scrollable Text Editor).

Note The End button only appears when you enter a valid statement or rule; the End button does not appear when entering simple attribute values.

To update the text without ending the editing session:

➔ Click the Update button.

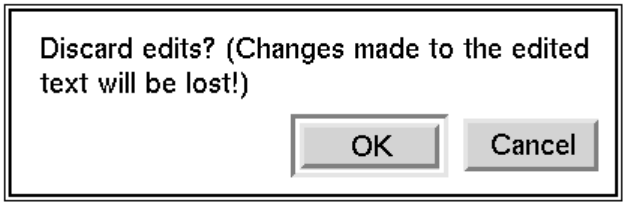
G2 accepts your edits without closing the editor.

The Update button appears whenever the End button appears.

To cancel an editing session:

1 Click the Cancel button, enter the Ctrl + a keystroke command, or press the Escape key.

G2 displays a confirmation dialog:



2 Click the OK button to discard your edits and to close the editor.

Using the Search Facility

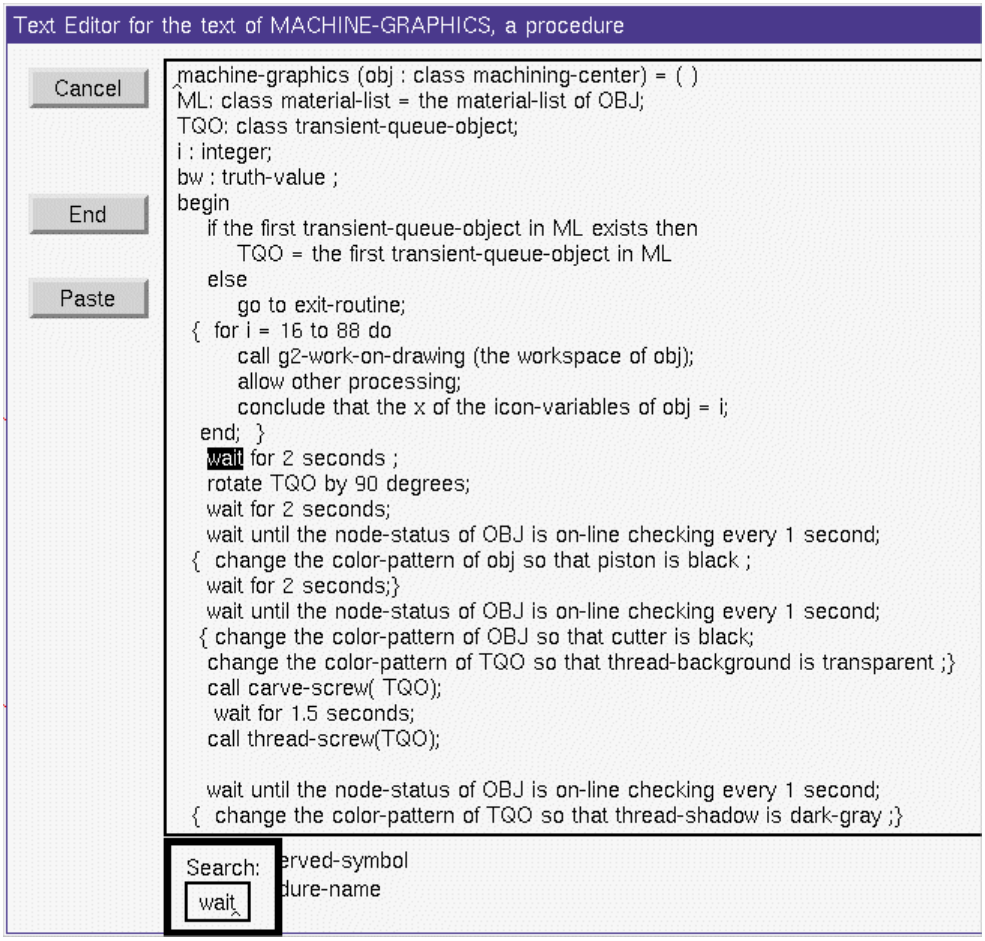
You can use the Search facility any time you are using the Text Editor. The facility lets you search for a character or string of characters anywhere in the text, then continue searching for additional instances of the search string.

To start searching in the Text Editor:

1 Position the cursor at the location in the code from which you want to start searching and enter Alt + s to display the Search type-in box. In the next example, the cursor is positioned at the beginning of the procedure.

2 Type the text you are searching for in the type-in box.

Searching for any character or string of characters is incremental. As you enter each character, the search facility moves to and highlights the first instance of that character it locates after the current position. Entering more characters highlights the first instance of the character string. The next example shows the search string, wait, being highlighted as the characters are entered:



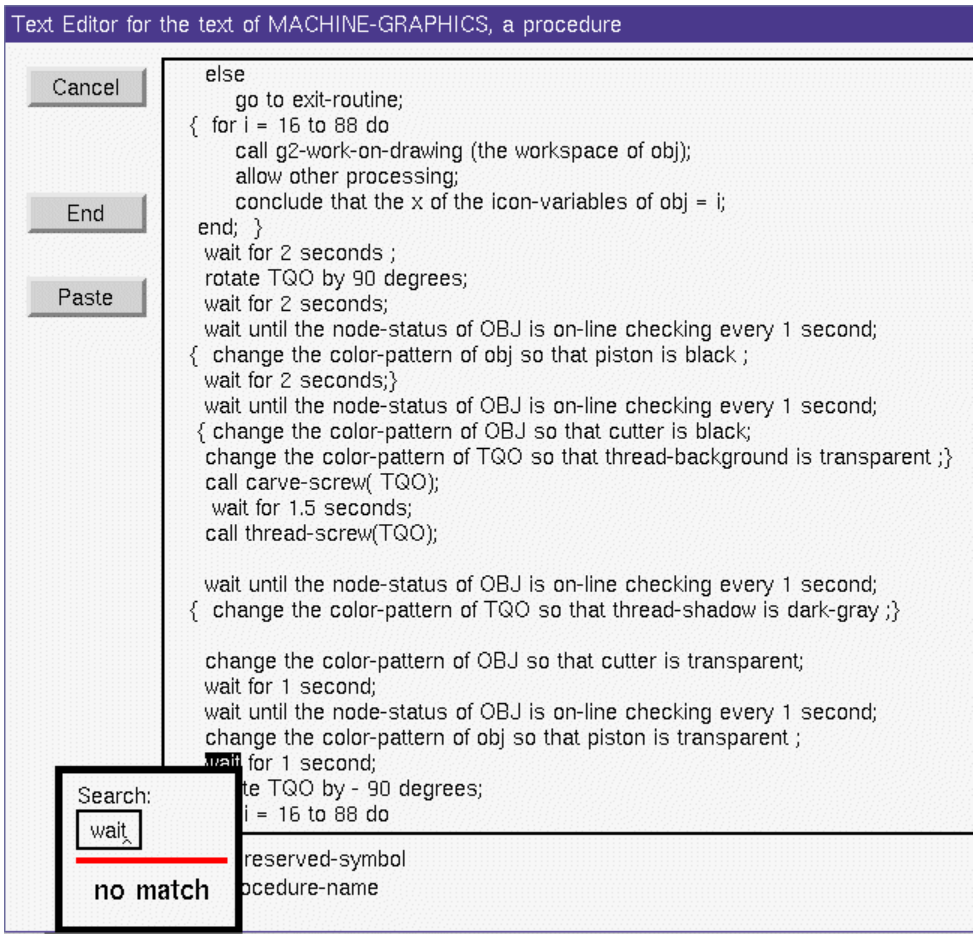
Note All Text Editor commands for cutting, pasting, and moving text work within the Search type-in box.

3 To continue searching for the same string, enter Alt + s again. If another instance is found, highlighting moves to that location in the text.

4 To search backwards, enter Alt + p (for previous).

You can enter Alt + p to search backwards at any time while the Search type-in box is displayed. If you have been searching forward over successive matching instances of your search string, using Alt + s, then decide to change direction, using Alt + p, the first string match will be the same one you have just found and the highlighted region will not move until you enter Alt + p a second time.

After the last match has been found, if you enter Alt + s again, the previous match remains highlighted and the message no match appears in the type-in box below the search string:

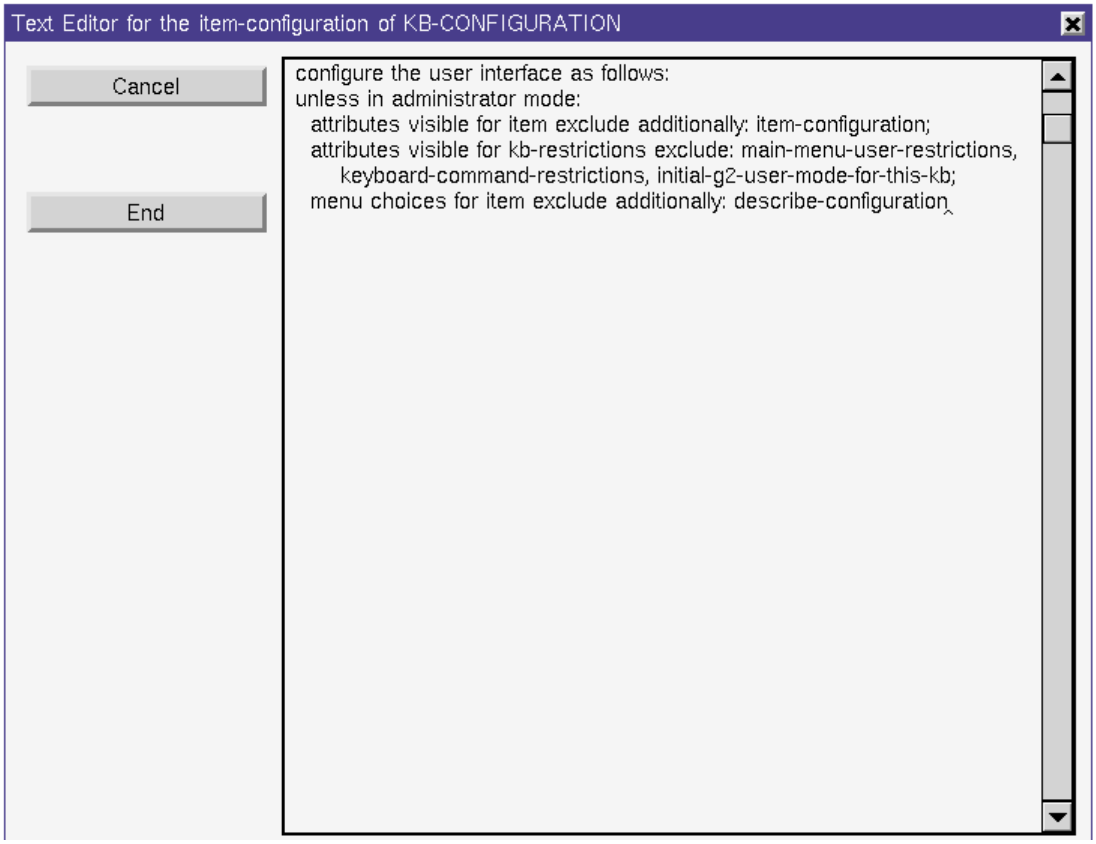


5 To end searching, enter Ctrl + a or press Return.

Using the Scrollable Text Editor

G2 provides a **scrollable text editor** for editing large amounts of text in particular attributes. A scrollable editor lets you scroll the contents of the text box and enter line feeds more easily.

The next figure shows how the scrollable editor appears as you edit an item configuration:



Editing the following attributes opens a scrollable editor:

- The text attribute of a procedure or method.
- Compiled attributes in definitions, such as:
  - item-configuration and instance-configuration
  - class-specific-attributes, attribute-initializations, attribute-displays, stubs, and icon-description in class, object, connection, and message definitions

The scrolling editor behaves exactly like the standard editor with two exceptions.

To enter a line feed in a scrollable editor:

➔ Use the Return key.

To end an editing session in a scrollable editor:

➔ Enter Control + Return key or click the End button, depending on the context.

Using the Clipboard and Scrapbook

You can cut and paste text in the editor by choosing either the popup menu choices or clicking buttons, depending on how you have configured your edit-options environment (see Configuring Editor Menu and Button Options). These options automatically appear whenever you select text in the editor by dragging the mouse over a sequence of characters.

Both Windows and X Windows systems have a **clipboard**, which serves as a holder for text and data that has been cut or copied from any clipboard-compliant applications.

The editor has a **scrapbook**, which is a special workspace that maintains text that you cut and copy from the editor. The scrapbook was used prior to the clipboard, but remains as part of the editor. G2 creates a scrapbook workspace the first time you cut or copy text in the editor. The scrapbook is a transient, non-KB workspace, which is not saved when you save the KB. The text items stored in the scrapbook are text inserters, which means that you can also use them to insert text directly into the editor.

Whenever you complete a cut or copy operation in the editor, or drag your cursor over text that is outside the editor, G2 copies the text both to the clipboard and the scrapbook. The clipboard text is then used as the pasting source, and the scrapbook text remains unused. You can access scrapbook text as described in Interacting with the Scrapbook Directly.

The amount of text you can cut or copy at one time is limited by the G2 text clipboard buffer, which can hold a maximum of 32766 characters, including invisible characters such as a linefeed. Attempting to copy more than the maximum number of characters to the clipboard truncates the copied text.

To cut and paste text in the editor:

1. Drag the mouse cursor over the sequence of characters you want to cut, and release the mouse.
2. Choose the cut menu option or click the Cut button to delete the text from the current cursor location and place it in the scrapbook.
3. Move the cursor to the new location where you want to paste the text.
4. Choose the paste menu option or click the Paste button to insert the text.

Note Clicking the Paste button a second time inserts the same text again.

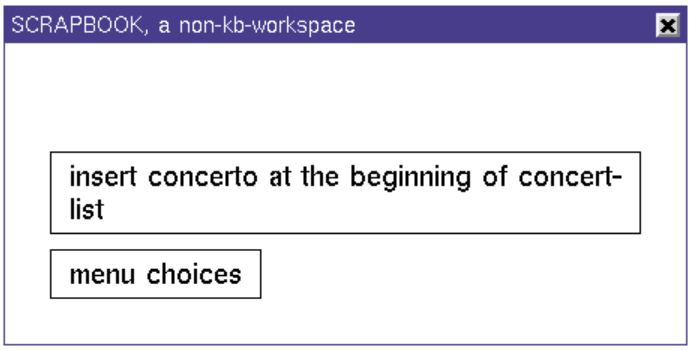
Interacting with the Scrapbook Directly

You can look at the contents of the scrapbook by displaying the Scrapbook workspace. The scrapbook contains text inserters, which allow you to insert the text item directly into the editor by clicking it. For detailed information about creating and using text inserters, see Using Text Inserters to Insert Text into the Text Editor.

To display the scrapbook:

➔ Choose Main Menu > Get Workspace > scrapbook.

For example, your scrapbook might look like this:



To insert text directly from the scrapbook:

1 Open the editor, and place your cursor in the location in the editor where you want to insert text from the scrapbook.

2 With the scrapbook, insert text in one of two ways:

- To insert the entire text item, click on a text inserter in the scrapbook.
- To insert part of the text item, drag the cursor over the sequence of characters you want to insert, starting from the left side.

Controlling the Amount of Text in the Scrapbook

By default, G2 keeps only 50 text items in the scrapbook. When you cut or copy more than ten items, G2 deletes the last item on the workspace and inserts the new text item at the top.

You can control the amount of text that the scrapbook holds, and you can delete individual text items in the scrapbook or the entire scrapbook.

To edit the maximum number of text items in the scrapbook:

➔ Edit the maximum-number-of-scraps-to-keep attribute in the Editor Parameters system table.

To delete a text item in the scrapbook:

➔ Click on the border of a text inserter in the scrapbook, and select delete from its menu.

To delete the entire scrapbook:

➔ Click on the background of the Scrapbook workspace, and select Delete Workspace.

G2 creates a new Scrapbook workspace the next time you cut or copy text.

Performing Other Edit Operations

To configure your editing options environment to have the full range of button *or* popup-menu choices described in this section, see Configuring Editor Menu and Button Options.

You can use the Edit Operations menu or buttons in the editor to:

- Delete text without placing it on the clipboard or scrapbook.
- Replace selected text by typing directly, without cutting.
- Insert selected text at the current cursor location.
- Move selected text to the current cursor location.
- Cut selected text and insert it immediately at the current cursor location.

To delete text by using the menu:

1 Drag the mouse cursor over the text you want to delete.

2 Click the Delete button or menu choice.

G2 does not place the deleted text on the clipboard or scrapbook.

To replace text, you can use the pending delete technique:

1 Select the text to be replaced by dragging the mouse cursor over it.

2 Without cutting, replace the selected text directly by typing one or more characters.

To insert the selected text at the cursor:

1 Move the mouse cursor to the new location where you want to insert text.

2 Drag the mouse cursor over the text to insert.

3 Choose the insert button or menu choice.

G2 inserts the selected text at the cursor location. If you have selected a word, G2 also inserts the following space.

To move the selected text to a new location:

1 Move the mouse cursor to the new location where you want to move the text.

2 Drag the mouse cursor over the text to move.

3 Click the move button or menu choice.

G2 moves the selected text to the cursor location. G2 only moves the selected text; it does not move any following spaces.

To cut the selected text and insert it at a new location:

1 Move the mouse cursor to the new location where you want to insert text.

2 Drag the mouse cursor over the text to cut.

3 Choose the cut and insert button or menu choice.

G2 cuts the selected text, placing it on the clipboard and scrapbook, and inserts the text at the cursor location. If you have selected a word, G2 inserts the following space.

Cutting/Pasting between G2 and Other Applications

You can cut and paste text, including international characters, between G2 and most other applications that include an edit menu with Cut, Copy, and Paste options.

Note Telewindows supports all of the G2 cut and paste functionality, but for simplicity, this section refers only to G2.

You can exchange text between two G2 processes, and all text, including international characters, can be copied between G2 processes. On Windows systems, you can copy international text to other applications that support the Unicode character set. On X Windows systems, you can copy international text to other applications that support Compound Text.

To cut or copy text from G2 to another application:

1 From the editor, select the text to cut or copy and choose the Cut or Copy menu selections or buttons.

2 With the destination application in focus, position the cursor and choose that application's Paste option.

To copy text from an external application to G2:

1 From the source application, select the text to copy and choose Cut or Copy from the application Edit menu.

2 With G2 in focus, click Paste in the editor. The new text appears in the editor.

Using the Clipboard for Text Exchange

G2 uses the clipboard to exchange text in both an X Windows and a Windows environment.

Exchanging text through the clipboard involves a:

- Source: the application from which text is cut or copied.
- Destination: the application into which the text is pasted.

Text Source

When placing text upon the clipboard, the source application has no knowledge of the destination application, and cannot know which text formats it supports. To accommodate various possibilities, the source application may copy text to the clipboard in multiple formats, depending on the platform:

Source Application	Windows	X Windows
G2/Telewindows	CF_TEXT	ISO Latin-1
	CF_UNICODETEXT	Compound Text
Non-G2/Telewindows	CF_TEXT	ISO Latin-1
	CF_UNICODETEXT	Compound Text

Note The CF_TEXT format is basically the same as ISO Latin-1. Not all applications support the CF_UNICODETEXT or Compound Text formats.

A major difference between the text formats is their support of international characters:

This text format...	Supports...
CF_TEXT	All keyboard characters and some European characters such as those for French, German, and so on.
ISO Latin-1	All keyboard characters and some European characters such as those for French, German, and so on.
CF_UNICODETEXT	All Unicode-supported international characters.
Compound Text	Most international characters.

When pasting clipboard text from G2 to a destination application, international character support in text is dependent on:

- Unicode support on Windows platforms.
- Compound Text support on X Windows platforms.
- Installed fonts on both platforms.

Displaying Unicode Characters

Internally, G2 uses and fully supports the entire Unicode character set. Such support, however, does not guarantee universal character display. You can enter any Unicode character in the editor, or import any character into a KB, but G2 may be unable to display the character. G2 represents any character that it cannot display as a solid block (■).

Displaying characters in a KB, both within the editor as a character entry facility, and in other contexts such as messages and name boxes, is largely dependent upon the languages that G2 supports and the available fonts.

In addition to English and European languages, G2 supports:

- Japanese
- Korean
- Russian

Note To display Japanese and Korean characters in outline fonts requires appropriate authorization.

By default, the fonts G2 uses to display characters reside in a `fonts` subdirectory of the G2 product directory. You can use an alternative location for the font files, but then must specify that location as described in Using G2 Fonts.

If the correct fonts are not installed on either G2 or another application, solid blocks (■) appear in place of the international characters.

When pasting text, these are the results you can expect, depending on what the destination application supports:

If the application...	Then Windows...	And X Windows...
Supports Unicode and has proper fonts installed	Displays all text and international characters that G2 copied to the clipboard.	
Does not support Unicode	Displays only supported characters.	
Supports Compound Text and has appropriate fonts installed		Displays most of the text and international characters that G2 copied to the clipboard.
Does not support Compound Text		Displays only supported characters.

No characters are lost when cutting or pasting between two G2 processes.

G2's intelligent cut and paste, where cutting a word and pasting it into existing text adds appropriate space characters, works as it normally does, as long as the text source and destination are the same G2. There is no intelligent cut and paste from another application into G2, or between two G2 processes.

On Windows platforms, text is null-terminated, as required by the Windows clipboard standard. On X Windows, text can contain embedded nulls.

Using Unicode and Special Characters

The default language of G2 is English. G2 also supports these other languages:

- Japanese
- Korean
- Russian

When the current language of G2 is set to one of these supported languages, and the appropriate fonts are available, the buttons of the editor and other G2 facilities are localized to the current language.

For more information about entering characters in supported languages, see Using the Natural Language Facilities.

From the editor, you can enter any character in the Unicode character set, including characters in languages for which G2 does not include fonts, and characters or symbols that are not available on your computer's keyboard, referred to here as *special characters*.

For more information about the Unicode character set, see G2 Character Support.

Entering Unicode Character Codes

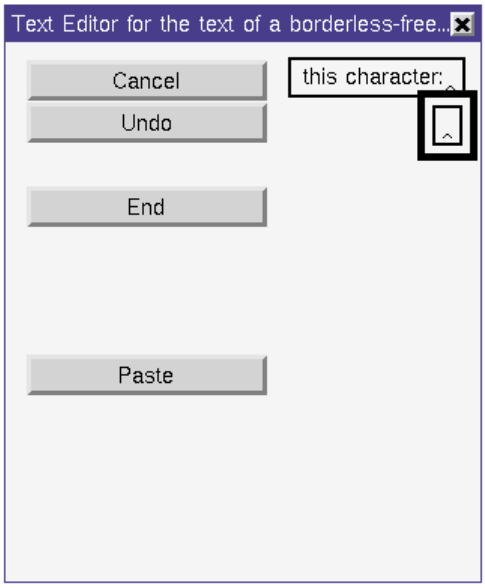
You can enter any Unicode character sequence in the editor. G2 interprets the character codes you enter in the context of the current language. For example, if the current language is japanese, and you enter a hexadecimal value, such as 2522, G2 interprets that value as a JIS code. Conversely, if the current language is english, and you enter the same hexadecimal value 2522, G2 interprets that as a Unicode character code, which is one of the box drawing symbols.

If G2 supports the font for the character you enter, the character appears. If not, G2 displays a solid block (■). For more information about character display, see Displaying Unicode Characters

To enter a Unicode character code:

1. Open the editor.
2. Enter Alt + i.

G2 displays a small type-in box for entering special characters:



3 Enter the four-digit hexadecimal Unicode code of the character you want to use. For example, in the Latin Extended-A block of the Unicode character set, the Latin capital letter K with cedilla is listed as U+0136. For this character, enter 0136.

Entering Special Characters

In addition to entering any Unicode character code, G2 includes some special characters that are common to English and other natural languages. You can enter one or more of these special characters by:

- Selecting a special character from the help screen
- Using an escape sequence

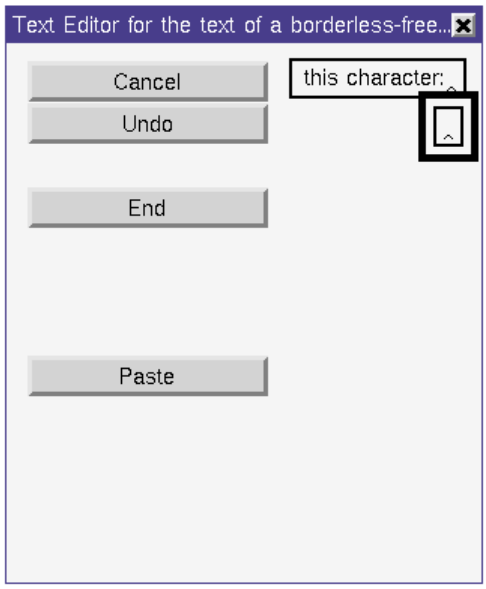
You can enter the G2 special characters by either selecting the character from the help screen, or by entering an escape sequence, described on the help screen.

To enter a special character by selecting it from the help screen:

1 Open the editor.

2 Enter Alt + i.

G2 displays a small type-in box for entering the escape sequence:



3 Enter Control + /.

G2 displays the following help screen for special characters:

a temporary-workspace

Special Characters with Diacritics

Special Characters	Base Characters	Type-In Sequence	Diacritic
à À è È ì Ì ò Ò ù Ù	a A e E i I o O u U	Alt+i `	Grave
á Á é É í Í ó Ó ú Ú	a A e E i I o O u U	Alt+i '	Acute
â Â ê Ê î Î ô Ô û Û	a A e E i I o O u U	Alt+i ^	Circumflex
ä Ä ë Ë ï Ï ö Ö ü Ü	a A e E i I o O u U	Alt+i : or Alt+i "	Umlaut
ã Ã ñ Ñ õ Õ	a A n N o O	Alt+i ~	Tilde
æ Æ œ Œ	a A o O	Alt+i e	Ligature
¢ ¢ ₣ ₣ ł Ł ø Ø ₪ ₪	c C f F l L o O s S	Alt+i /	Slash
ç Ç ş Ş	c C s S	Alt+i ,	Cedilla
å Å	a A	Alt+i O, Alt+i o, or Alt+i 0	Ring
ő Ő ű Ű	o O u U	Alt+i =	Double Acute
ğ Ğ	g G	Alt+i U or Alt+i u	Breve
ı İ	i I	Alt+i I or Alt+i i	Dot Above

Standalone Special Characters

Special Character	Type-In Sequence	Special Character	Type-In Sequence
ß	Alt+i s	©	Alt+i c
™	Alt+i t	®	Alt+i r
¿	Alt+i ?	¡	Alt+i !
«	Alt+i <	»	Alt+i >
£	Alt+i l	¥	Alt+i y
ƒ	Alt+i f	•	Alt+i *
€	Alt+i $	Tab	Alt+i Tab

For Unicode

Enter Alt+i followed by a 4-digit Unicode 2.0 hex code.
Preceed with x to force hex interpretation if the first digit is ambiguous. - e.g. "xC49A"

4 Drag your cursor over the special character you want to insert.

To enter a special character by using an escape sequence:

1 Open the editor.

2 Enter Alt + i to display the type-in box.

3 Enter Control + / to display the help screen for reference.

4 Enter a base character in the editor.

For example, to enter à, first enter the letter “a”.

5 Enter the escape sequence.

For example, to enter à, enter the Alt + i ` escape sequence.

When you enter Alt + i, G2 first displays the small type-in box. When you enter the additional character, in this case, a grave accent (`), G2 then inserts the special character into the editor and removes the small type-in box.

To enter a stand-alone special character:

➔ Open the editor, and enter the escape sequence directly.

For example, to enter the copyright character (©), enter Alt + i c.

Keystroke Commands

G2 includes a number of keystroke commands for performing a variety of editing tasks.

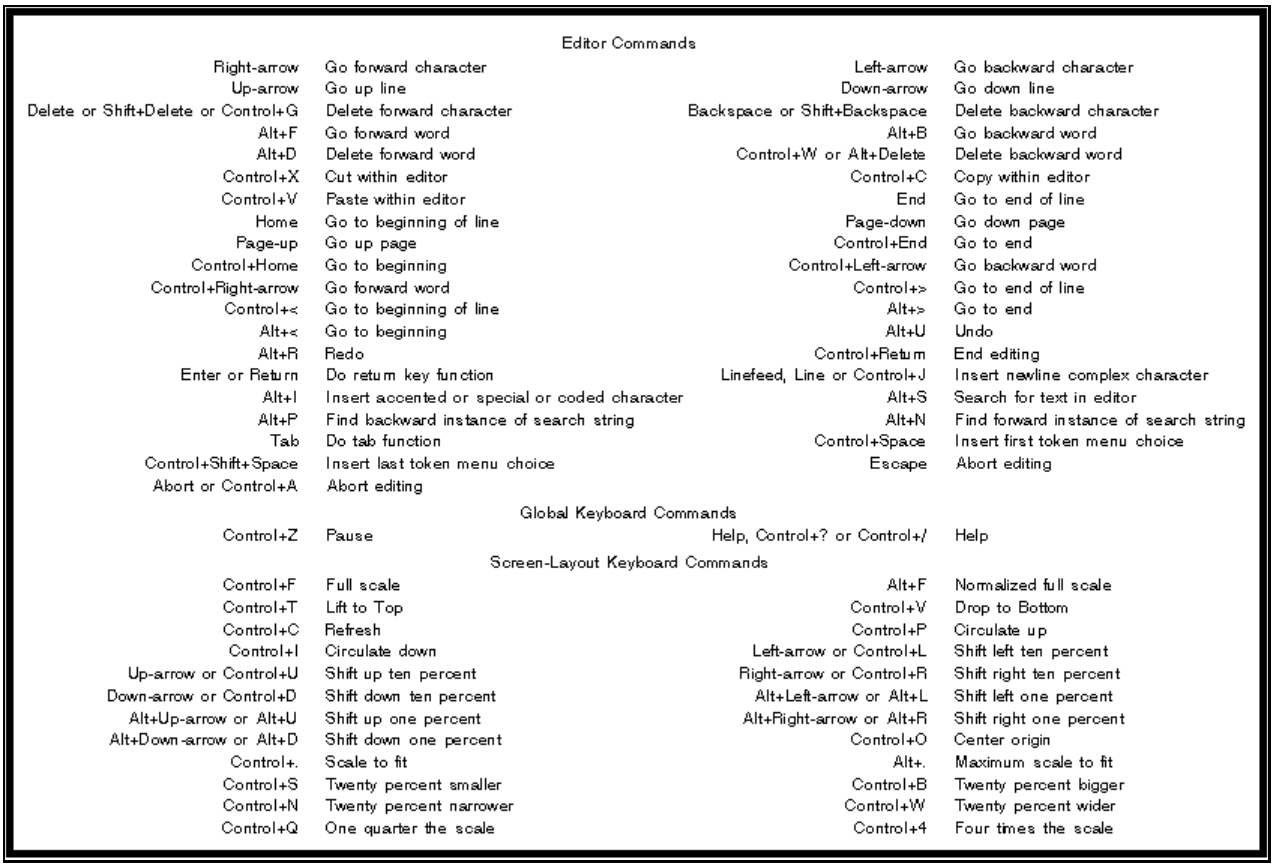
In the explanations that follow, a *word* means a sequence of characters delimited by the beginning of the text, the end of the text, a punctuation mark, or one or more spaces, tabs, or line breaks.

Displaying Help

To display a help screen of all keystroke commands:

1. Open the editor.
2. Press the Help key or use the Control + / command.

The following figure shows the portion of the help screen related to the editor:



Editor Commands

Right-arrow	Go forward character	Left-arrow	Go backward character
Up-arrow	Go up line	Down-arrow	Go down line
Delete or Shift+Delete or Control+G	Delete forward character	Backspace or Shift+Backspace	Delete backward character
Alt+F	Go forward word	Alt+B	Go backward word
Alt+D	Delete forward word	Control+W or Alt+Delete	Delete backward word
Control+X	Cut within editor	Control+C	Copy within editor
Control+V	Paste within editor	End	Go to end of line
Home	Go to beginning of line	Page-down	Go down page
Page-up	Go up page	Control+End	Go to end
Control+Home	Go to beginning	Control+Left-arrow	Go backward word
Control+Right-arrow	Go forward word	Control+>	Go to end of line
Control+<	Go to beginning of line	Alt+>	Go to end
Alt+<	Go to beginning	Alt+U	Undo
Alt+R	Redo	Control+Return	End editing
Enter or Return	Do return key function	Linefeed, Line or Control+J	Insert newline complex character
Alt+I	Insert accented or special or coded character	Alt+S	Search for text in editor
Alt+P	Find backward instance of search string	Alt+N	Find forward instance of search string
Tab	Do tab function	Control+Space	Insert first token menu choice
Control+Shift+Space	Insert last token menu choice	Escape	Abort editing
Abort or Control+A	Abort editing		

Global Keyboard Commands

Control+Z	Pause	Help, Control+? or Control+/	Help

Screen-Layout Keyboard Commands

Control+F	Full scale	Alt+F	Normalized full scale
Control+T	Lift to Top	Control+V	Drop to Bottom
Control+C	Refresh	Control+P	Circulate up
Control+I	Circulate down	Left-arrow or Control+L	Shift left ten percent
Up-arrow or Control+U	Shift up ten percent	Right-arrow or Control+R	Shift right ten percent
Down-arrow or Control+D	Shift down ten percent	Alt+Left-arrow or Alt+L	Shift left one percent
Alt+Up-arrow or Alt+U	Shift up one percent	Alt+Right-arrow or Alt+R	Shift right one percent
Alt+Down-arrow or Alt+D	Shift down one percent	Control+O	Center origin
Control+.	Scale to fit	Alt+.	Maximum scale to fit
Control+S	Twenty percent smaller	Control+B	Twenty percent bigger
Control+N	Twenty percent narrower	Control+W	Twenty percent wider
Control+Q	One quarter the scale	Control+4	Four times the scale

Moving the Cursor

These are the keystrokes for moving the cursor from its current position.

Press...	To move...
Left arrow	One character to the left
Right arrow	One character to the right
Ctrl + left arrow	One word to the left
Ctrl + right arrow	One word to the right
Up arrow	Up one line
Down arrow	Down one line
End	To the end of a line

Press...	To move...
Home	To the beginning of a line
Page Up *	Up one screen
Page Down *	Down one screen
Ctrl + End	To the end of a document
Ctrl + Home	To the beginning of a document

* Effective only within an editor with scroll bars.

Note Note that the arrow keys have a different meaning in the context of the editor than in other G2 contexts.

Cutting, Copying, and Pasting Text

These are the keystrokes for copying, pasting, and cutting text in the Editor:

Press...	To...
Ctrl + x	Cut selected text to the clipboard. If no text is selected, Ctrl + x deletes text backwards from the current cursor position.
Ctrl + c	Copy selected text to the clipboard.
Ctrl + v	Paste any text from the clipboard to the current cursor position.

Using Ctrl + c and Ctrl + v Outside of the Editor

The Ctrl + c and Ctrl + v keystrokes also work outside of the Editor, as follows:

Pressing...	In this context....	Has this effect...
Ctrl + c	With no Editor active	Refreshes the window display.
Ctrl + v	Over a workspace	Drops the workspace to the bottom.
	Over the background	Does nothing.

Selecting Text

These are the keystrokes for selecting text. Select text and then extend the selection by holding down the indicated keys to move the insertion point as follows:

Press...	To...
Shift + right arrow	Extend selection one character to right.
Shift + left arrow	Extend selection one character to left.
Ctrl + Shift + right arrow	Extend selection to end of word.
Ctrl + Shift + left arrow	Extend selection to beginning of word.
Shift + End	To the end of a line.
Shift + Home	To the beginning of a line.
Shift + down arrow	One line down.
Shift + up arrow	One line up.
Ctrl + Shift + End	To the end of a document.
Ctrl + Shift + Home	To the beginning of a document.

Deleting Text

To delete...	Use this command...
Character left	Backspace or Rubout keys.
Character right	Delete, Shift + Delete, or Control + g.
Word left	Alt + Delete key or Control + w.
Word right	Alt + d.
Backward from cursor to beginning of text	Control + x.

Note Note that the Control + g command has a different meaning in the context of the editor than in other G2 contexts.

Inserting Tabs and Line Breaks

You enter a Tab character by pressing the Tab key.

You use one of two commands to insert a line break, depending on the type of editor you are using:

To insert a...	Use this command in the standard editor...	Use this command in the scrolling editor...
Line break	Control + j or Linefeed key	Return key
Tab	Tab key or Alt + i Tab	Tab key or Alt + i Tab

When you enter a tab in the editor, G2 inserts the number of spaces specified by the number-of-spaces-to-insert-on-a-tab attribute; G2 does not use preset tab stops unless the value of this attribute is 0. Otherwise, use Alt + i Tab to enter an actual tab character.

To control the number of spaces that G2 inserts when using a tab:

➔ Edit the number-of-spaces-to-insert-on-a-tab attribute in the Editor Parameters system table.

Controlling the Editing Session

You use one of two commands to end the editing session, depending on the type of editor that you are using. You can also abort the editing session, and undo and redo by using a keystroke.

To...	Use this command in the standard editor...	Use this command in the scrolling editor...
Accept the text and close the editor	End button or Return key	End button or Control + Return key
Abort the editing session	Cancel button, Control + a, or Escape key	same

To...	Use this command in the standard editor...	Use this command in the scrolling editor...
Undo	Alt + u	same
Redo	Alt + r	same
Search forwards	Alt + s	same
Search for next occurrence	Alt + n	same
Search backwards	Alt + p	same

Inserting Prompts by using the Keyboard

You can insert text directly from the natural language prompts at the bottom of the workspace by using a keystroke:

To insert...	Use this command...
The first prompt	Control + Space Bar
The last prompt	Control + Shift + Space Bar

Text Editor Buttons

Depending on the context, the editor contains the following buttons along the left side of the workspace.

This button...	Does this...
Cancel	Closes the editor without making any changes to the text. If you have made changes to the text, G2 displays a confirmation message. Cancel is always visible.
End	Accepts the text and closes the editor. End is only visible when the current text is a valid, complete statement. The End button does not appear when you enter a value into a simple attribute, or when the statement is not valid or complete.
Paste	Inserts the top-most piece of text from the scrapbook at the current cursor location. Paste is only visible when text is in the scrapbook.

This button...	Does this...
Redo	Redoes the last edit you undid using Undo. (You can also use the keystroke command Alt + r.) Redo is only visible when you have undone one or more edits by using Undo.
Undo	Undoes the last edit you made in the editor. (You can also use the keystroke command Alt + u.) Undo is only visible when you have made text edits.
Update	Accepts the current text in the editor without closing the editor. Update is particularly useful for editing procedures. Update is not available when creating new statements or rules, when saving and loading KBs, or when using the Inspect facility.
Cut	Cuts the selected text to the clipboard.
Copy	Copies the selected text to the clipboard.
Delete	Deletes the selected text without placing it on the clipboard or scrapbook.
Insert	Inserts the selected text at the cursor location. If you have selected a word, the following space is also inserted.
Move	Moves the selected text to the cursor location. Only the selected text is moved, not any following spaces.
Cut and insert	Cuts the selected text, placing it on the clipboard and scrapbook, and inserts the text at the cursor location. If you have selected a word, the following space is also inserted.

46

The Icon Editor and Icon Management

Describes the G2 Icon Editor and its icon-description language.

Introduction

The Icon Editor allows you to define a class icon with graphic tools. The Icon Editor converts the resulting graphical description into G2 code, and sets this code as the value of the icon-description attribute of the class definition.

If you prefer, you can define an icon with the Text Editor, by editing the text of the icon-description attribute of the class definition. You can also use the Icon Editor and Text Editor together to define icons:

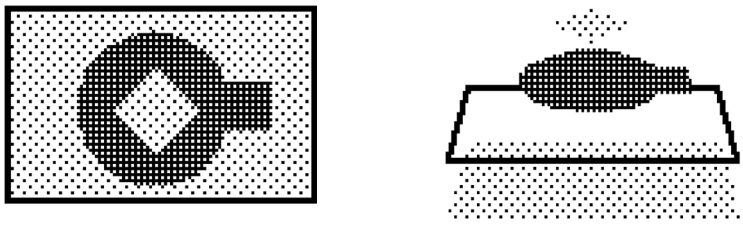
- You can use the Icon Editor to define the general appearance of an icon and the Text Editor to fine-tune the definition.
- You can use the Text Editor to extend the capabilities of the Icon Editor in various ways.

This chapter shows you how to use the Icon Editor to define icons, discusses textual icon editing, and shows you how to use the Icon and Text Editors together for icon definition. For information about the Text Editor, see Chapter 45, The Text Editor on page 1599.

Composition of an Icon

An object icon is made up of one or more overlapping layers. A **layer** is like a piece of transparent film with a one-color image on it. A layer contains one or more graphic elements (circles, lines, etc.) defined with the Icon Editor and/or the Text Editor.

For example, the following illustration shows an icon that has four layers, and a conceptual view of the layers from the side:



Layers can be assigned to regions. A **region** is a named collection of one or more layers. All layers in a region have the same color or metacolor. For information about metacolors, see Identifying the G2 Color Palette on page 417.

You can change the color of a region for a particular instance of an object class with a change action:

change the *region* icon-color of *object* to *color*

This action changes the color of all layers of *region* of the icon of *object* to *color*. The change does not affect the class default color of *region*; it affects only the instance. If an icon does not have any regions, the *region* specifier can be omitted. The action then changes the color of all layers of the icon to *color*.

Starting the Icon Editor

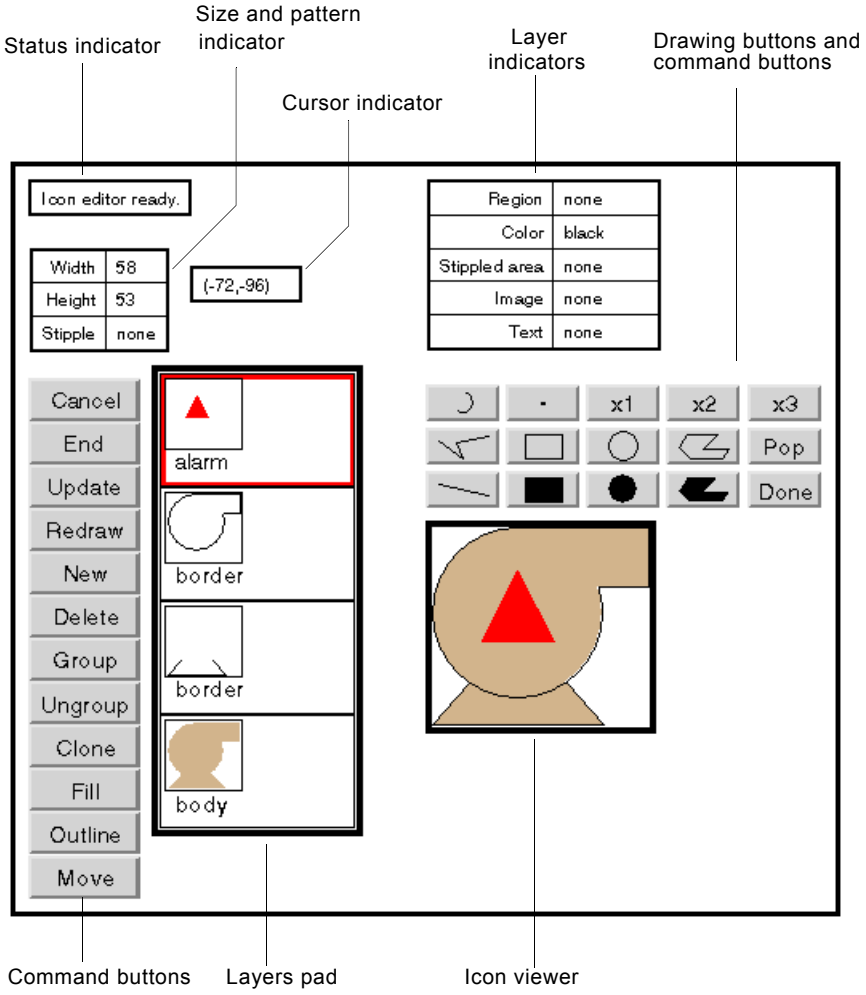
To create or modify the icon for an object class:

➔ Click the left side of the icon-description attribute in the class definition, then select edit icon from the menu that appears.

or

➔ Select edit icon from the menu of the class description.

The following figure shows the Icon Editor open on an icon that represents a pump:



Parts of the Icon Editor

This section briefly describes each component of the Icon Editor. Subsequent sections show you how to use the editor's capabilities to create icons.

Layers Pad

The **layers pad** shows all layers of the icon. Each layer is accompanied by a symbol that names the region that the layer belongs to, or the color of the layer if it does not belong to a region. The example shows four layers in three regions: alarm, border, and body.

One layer in the layers pad is always selected. This layer is called the **current layer**. The current layer is surrounded by a box whose color is the layer color. In the previous example, the first layer is selected. Icon Editor operations that apply to a particular layer apply to the current layer in the layers pad.

You can use the layers pad, in conjunction with various commands, to create, reorder, clone, redefine, and delete layers.

Icon Viewer

The icon viewer shows the complete icon. The view consists of all layers, superimposed on one another in the order shown in the layers pad. The backgrounds of the layers are transparent, but the graphics on the layers are opaque. The first layer occludes all others; the second occludes all but the first; and so on.

The size and shape of the icon viewer are the same as the size and shape of the icon itself. An outline indicating these dimensions appears on each layer in the layers pad. The icon size indicator shows the icon's dimensions numerically in workspace units.

You can reshape the icon viewer with the mouse to change the size and/or shape of the icon, and you can use the viewer as a drawing pad to draw new graphics on the current layer. When you make any change that affects the appearance of the icon, the icon viewer updates immediately to reflect the change.

Layer Indicators

The four layer indicators show four properties of the selected layer:

- **Region Indicator**: Names the region (if any) to which the layer belongs. This name also appears at the bottom of the layer in the layers pad. You can edit this indicator to specify a layer's region.
- **Color Indicator**: Names the color of the layer. The layer's selection box and graphics appear in this color. You can edit this indicator to specify a layer's color.
- **Stipple Area Indicator**: Names the stipple pattern that is applied to the layer. You can edit this indicator to define a stipple pattern for the entire area of the layer, or for a rectangular area of the layer.
- **Image Indicator**: For a layer that contains an externally defined image, the name of the image. You can edit this indicator to specify an image name.
- **Text Indicator**: For a layer that contains a text element, the text of the element; or if more than one text element exists, the text of the first element. You can edit this indicator to specify a text element.

Other Indicators

The Icon Editor includes three other indicators. These are:

- **Status Indicator**: Displays various messages that name the activity currently underway in the editor, prompt for user actions, and describe conditions that prevent the editor from carrying out a command as requested. Check the status indicator if you are ever unsure of what the editor is doing, or of what you should do next.
- **Size and Stipple Pattern Indicator**: Shows the size of the icon in workspace units and the stipple pattern applied to the icon.

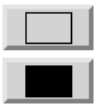
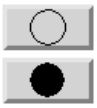
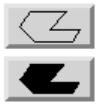
  The first number gives the width, the second the height. You can edit these numbers to change the size and/or shape of the icon; the icon viewer reshapes accordingly. If you reshape the icon viewer with the mouse, the size indicators change to reflect the new shape.

  The stipple pattern defines a global stipple. This pattern applies to the entire icon, including the entire area on each of its layers. You can edit this indicator to define a stipple pattern for the icon.
- **Cursor Indicator**: Shows the current position of the cursor, in workspace units, measured from the upper left corner of the icon viewer, whose coordinates are (0,0).

Drawing Buttons

These buttons control the creation of graphic elements in the icon viewer. You can use them to add graphics to the current layer. Each button contains a glyph that indicates the type of element that the button draws. The following table summarizes the drawing buttons and their actions.

Button	Command	Description
	point	Draws a dot that occupies one workspace nit.
	line	Draws a line between any two coordinates in the icon.
	segmented line	Draws a set of lines that form an open polygon.
	arc	Draws a circular arc.

Button	Command	Description
	rectangle	Draws either an outlined or a filled rectangle.
	circle	Draws either an outlined or a filled circle.
	polygon	Draws either an outlined or a filled polygon.

Command Buttons

These buttons control all Icon Editor actions except the drawing of graphic elements. You can use them as needed to invoke editor commands. Each button contains a label that names its function. The following table summarizes the command buttons and their actions:

Button	Description
	Closes the editor without saving changes.
	Saves changes and closes the editor.
	Saves changes without closing the editor.
	Refreshes the icon viewer.
	Creates a new layer in the layers pad.
	Deletes the current layer.
	Groups two layers, making them a single layer.

Button	Description
Ungroup	Ungroups the current layer, placing each constituent graphical element on a separate layer.
Clone	Clones the current layer, creating another layer that contains all of the same graphic elements.
Fill	Changes any outlined elements in the current layer into filled elements.
Outline	Changes any filled elements in the current layer into outlined elements.
Move	Moves graphics within a layer.
x1 x2 x3	Displays the icon viewer at normal size (x1), or magnifies it to two (x2) or three (x3) times normal size.
Pop	Removes the most recently created graphic element in the current layer.
Done	Ends a drawing sequence.

Defining Icons

Creating and modifying icons with the Icon Editor entails some or all of the following:

- Creating, reordering, cloning, and deleting layers.
- Assigning colors to layers.
- Drawing graphical elements.
- Defining text elements.
- Importing external images.
- Defining regions and groups.

You must create a layer before you can use it, but otherwise you can perform these operations in any desired order. You can also use the Text Editor in

conjunction with the Icon Editor to give an icon extended capabilities that the Icon Editor alone does not provide:

- A background layer, as described under Specifying an Icon Background Layer on page 1668.
- The ability to animate icons, as described in Animated Icons on page 1670 and subsequent sections.

Starting an Icon Definition

You cannot invoke the icon editor on a class-definition until you have specified a superior class that has an iconic representation.

When you invoke the Icon Editor on an object-definition that has no superior class, the editor creates by default a single layer whose color is the metacolor foreground. You can use this layer as a starting point to create an icon for the class definition. When you later specify a superior class, the icon you have defined will override the inherited icon.

When you invoke the editor on a class definition that already has an icon, either locally defined or inherited from a superior class, the editor displays the layers of the icon in the layers pad. You can edit this icon as desired. Changes will affect only the icon definition of the class itself; the definition in the superior class will not be affected.

Controlling Icon Size and Shape

The default icon size is a square whose side is 100 workspace units. You can reshape an icon to have different dimensions. The length and width need not be equal.

To reshape an icon:

➔ Use the mouse to drag the right side, the bottom, and/or the lower right corner of the icon viewer as needed to give the viewer the desired size and shape. As you drag the corner, the icon size indicator changes accordingly.

or

➔ Edit the length and/or width shown in the icon size indicator to specify the desired values.

The corner of the icon viewer moves to reflect the specified shape.

You can reshape an icon for which graphics already exist. Such changes do not affect the existing graphics. If you shrink the viewer so that it is smaller than existing graphics, the graphics are truncated on the right and/or bottom. However, the obscured information is not lost: it can be recovered at any time by increasing the size of the icon.

Tip If you intend to rotate an icon, make both the length and the width even numbers. Then the item-x-position and item-y-position of the icon will not change when it is rotated.

Controlling Icon Viewer Magnification

By default, the icon viewer shows an icon as it would appear on a normal sized workspace. You can magnify the icon viewer without affecting the icon itself. Such magnification can be convenient for drawing finely detailed icons.

To magnify the icon viewer:

➔ Click the x2 button to double the display size of the viewer, or the x3 button to triple it.

The actual size of the icon, measured in workspace units, is unaffected.

To demagnify the icon viewer:

➔ Click the x1 button to restore the viewer to unmagnified size.

The actual size of the icon, measured in workspace units, is unaffected.

You can obtain magnifications much larger than the x3 button provides by using Control + b to enlarge the Icon Editor workspace. As the editor becomes larger, the icon viewer grows with it, increasing the effective magnification. Use Control + s to shrink the editor back to normal size.

Working with Layers

You can create, clone, move, or delete a layer at any time. Before you can do any of these, you must select one of the existing layers.

To select a layer:

➔ Click the mouse on the layer.

The layer becomes the current layer.

Creating Layers

To create a new layer:

1 Select the existing layer above which the new layer is to appear.

2 Click the New button.

The editor creates a new empty layer above the selected layer, then selects the new layer. The new layer has the same color as the previously selected layer, but does not belong to any region.

Moving Layers

To move a layer:

1 Select the layer that is to be moved.

2 Drag the layer to the desired position in the layers pad.

The editor shifts layers up or down as needed to make room for the moved layer, and places that layer in the indicated position. The icon in the icon viewer changes as appropriate to reflect the new ordering of the layers.

You can move a layer to the right of the layers pad while you are dragging it, so that you can see the other layers during the move.

Cloning Layers

To clone a layer:

1 Select the layer that is to be cloned.

2 Click the Clone button.

The editor creates a clone of the selected layer immediately below the layer, then selects the clone. The cloned layer is an exact duplicate of the original, except that it does not belong to any region.

Deleting Layers

To delete a layer:

1 Select the layer that is to be deleted.

2 Click the Delete button.

The editor deletes the layer, then shifts other layers up as needed to fill the space.

Specifying Colors

Every layer has an associated color or metacolor. All graphics on the layer have this color. You can change the color of a layer at any time.

To set the color of a layer:

1 Select the layer.

The current color appears in the Color Indicator.

2 Click on the name of the current color.

The color selection workspace appears.

3 Select the desired color or metacolor.

The color of the layer changes to be the selected color.

The color transparent looks opaque in the Icon Editor, so you can see the graphics in a transparent layer, but is transparent when the icon appears on a workspace, allowing the workspace background to show through.

Creating Graphics

To create graphics:

1. Select the layer on which the graphics are to appear.
2. Click the drawing button for the desired type of graphic.

   Once you click a drawing button, you can draw as many graphics of its type as you like. The button remains on until you click Done or do something other than draw additional graphics of the type.
3. Draw the graphic on the icon viewer (*not* the layer itself).

If you try to draw in the current layer, the Icon Editor will think you are trying to move the layer. If this happens:

- Restore the layer to its original position (if needed).
- Retry the drawing operation in the icon viewer.

As you draw, the cursor indicator updates continuously to indicate the position of the cursor in workspace units. You can use this information to draw very precise graphics. All points are one workspace unit in size; all lines are one workspace unit thick.

Drawing Points

To draw a point:

1. Click the Point button.
2. Click the mouse at the location of the point.

Drawing Lines

To draw a line:

1. Click the Line button.
2. Click at the location of the beginning of the line.
3. Click again at the location of the end of the line.

Drawing Segmented Lines

To draw a segmented line:

1 Click the Segmented Line button.

2 Click at the location of the beginning of the line.

3 Click again at the location of each vertex.

At each click, a line segment appears connecting the vertex to the previous vertex. When you have created the last vertex:

4 Click the Done button.

Drawing Arcs

To draw an arc:

1 Click the Arc button.

2 Click at the location of the beginning of the arc.

3 Click a second point.

A line appears between the two points. As you move the mouse, this line flexes and extends so that it always forms a circular section running from the first point, through the second, to the mouse position. When the arc has the desired shape:

4 Click a third point.

The arc remains as it was when you clicked the third point.

Drawing Rectangles

To draw a rectangle:

1 Click the Rectangle or Filled Rectangle button.

2 Click at the location of one corner of the rectangle.

3 Click at the location of the diagonally opposite corner of the rectangle.

Drawing Circles

To draw a circle:

1. Click the Circle or Filled Circle button.
2. Click at the position of the center of the circle.

   A circle appears. The center remains at the first point. As you move the mouse, the circle expands and contracts so that its edge is always at the mouse position. When the circle has the desired radius:
3. Click a second point.

The circle remains as it was when you clicked the second point.

Drawing Polygons

A filled polygon must be a simple polygon (its edges must not cross) or the editor will delete it after it is complete. An unfilled polygon need not be simple.

To draw a polygon:

1. Click the Polygon or Filled Polygon button.
2. Click at the location of a vertex.
3. Click again at the location of each additional vertex.

   At each click, a line segment appears connecting the vertex to the previous vertex. When you have created the last vertex:
4. Click the Done button.

The editor automatically connects the last vertex to the first.

Toggling Filled and Outlined Graphics

You can change all closed graphics on a layer to be either filled or unfilled (outlined). The change does not affect any graphics that already have the chosen appearance.

To set all closed graphics to be filled:

1. Select the layer to be changed.
2. Click the Fill button.

The editor redraws all closed graphics in the layer to be filled. Exception: any nonsimple polygon will remain unchanged.

To change all closed graphics to be outlined:

1 Select the layer to be changed.

2 Click the Outline button.

The editor redraws all closed graphics in the layer to be outlined.

Deleting Graphics

You can delete graphical elements in a layer in the reverse order of their creation.

To delete the most recent graphical element in a layer:

1 Select the relevant layer.

2 Click the Pop button.

The editor deletes the most recently created graphical element in the selected layer. To delete additional elements, continue clicking Pop.

If you click Pop on an empty layer, the editor deletes the layer itself. The next layer below the deleted layer, or the bottom-most layer if there was no next layer, becomes the current layer.

Moving Graphics

You can use the mouse to change the positions of the graphical elements in a layer. All of the elements move together to the new position.

To move the graphics in a layer:

1 Select the layer whose graphics are to be moved.

2 Click the Move button.

3 Drag the graphics to the desired position.

The editor clips the graphics as needed if you move them outside the borders of the icon. If you move an element far enough, it may disappear entirely, but it still exists: it will reappear if you move it back into the viewing area, or expand the area to expose the graphic.

If you want to change the relative positions of the graphical elements in a layer, ungroup and group the layer as needed to provide separate access to its elements, as described under Creating Groups on page 1660.

Reshaping Graphics

You cannot use the mouse to reshape a graphical element after you have drawn it. If the shape of the element is not satisfactory, you have two options:

- Delete and replace the element.
- Use the Text Editor to modify the icon description.

For information on using the Text Editor to modify an icon, see Editing Icons Textually on page 1662.

Defining Text Components

You can use the Icon Editor to define a single text string as a component of a layer, and to combine such layers into a single layer with multiple text components.

The background of a text component is transparent. The text of a text component has the color of the component's layer, and occludes components in lower layers, just as a graphical component does.

The Layer Indicators include a Text Indicator for including a text component in a layer. When a layer has multiple text components, only the first shows in the Text Indicator. All text components appear in the icon description in the class definition's table.

To specify a text component in an icon layer using the Icon Editor:

1 Make the layer the current layer.

2 Edit the value of the layer's Text Indicator. The grammar is:

string at (*x-position*, *y-position*) in *font-size*

where:

string: The text of the component.

x-position: The position of the left edge of the text relative to the left edge of the icon.

y-position: The position of the baseline of the text relative to the top of the icon.

font-size: Any standard G2 font size: small, large, or extra-large.

If the text is larger than the icon that displays it, G2 clips it to fit the available area, both in the Icon Viewer and on a workspace. If necessary, G2 also truncates the text displayed in the Text Indicator. Neither clipping nor truncation affect the text itself as defined in the icon description.

To put more than one text component into a layer, specify each in a separate layer, then combine the layers using the Group button, as described under Creating

Groups on page 1660. Only the first text component in the combined layer appears in the layer's Text Indicator. Use Ungroup to regain separate access to the components of the combined layer.

Applying a Stipple Pattern

A *stipple* generally is an application of some color as a pattern appearing as dots, flecks, or short strokes, i.e., as opposed to a solid color. To stipple an area is to apply a stipple.

In G2, the word stipple refers to one of the named stipple patterns, which are represented as any of the following symbols:

- light-stipple
- medium-stipple
- dark-stipple

A `*stippled icon* is one whose icon description contains either a stipple header or one or more stippled-area elements.

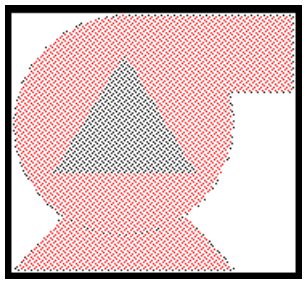
Stippled Header

A *stipple header* defines a global stipple that applies to the entire icon, including the complete area of all layers. A stipple header in an icon description appears at the beginning of the icon description as:

stipple: <*stipple pattern*>

where *stipple pattern* is one of the G2 stipple patterns.

For example, the following pump has a light-stipple pattern header defined for the icon. Notice that the pattern applies to all layers of the icon.



Icon description

```
width 58; height 53;
body = red, border = black;
stipple: light-stipple;
body:
  filled circle (0, 23) (23, 0) (46, 23);
  filled polygon (36, 42) (46, 53) (0, 53)
    (10, 42);
  filled polygon (23, 0) (58, 0) (58, 16)
    (45, 16);
border:
  outline (23, 0) arc (23, 46) (45, 16) (58,
    16) (58, 0);
  lines (10, 42) (0, 53) (46, 53) (36, 42);
black:
  filled polygon (38, 33) (8, 33) (23, 8)
```

Stipple header

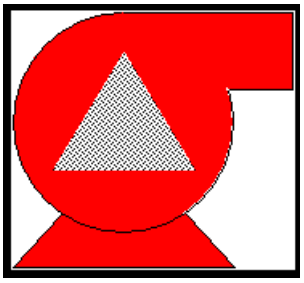
Stippled-Area Elements

A *stippled-area element* defines a stipple that applies to a particular layer of the icon. This can include the entire layer or a rectangular portion of a layer. A stipple area in an icon description appears among the elements of a given icon layer as:

stippled-area <*stipple pattern*> [(<*x1*>, <*y1*>) (<*x2*>, <*y2*>)]

where *stipple pattern* is one of the G2 stipple patterns and *x2*, *y1*, *x2*, and *y2* are points that define the top-left and bottom-right corners, respectively, of the rectangular area. If the points are undefined, the entire rectangular area of the icon layer is stippled.

For example, the following pump has a light-stipple-area drawing element defined for the triangle layer of the icon. Notice that the pattern applies to the triangle layer only, and that the pump itself is a solid color.



Icon description

```
width 58; height 53;
body = red, border = black;
body:
  filled circle (0, 23) (23, 0) (46, 23);
  filled polygon (36, 42) (46, 53) (0, 53)
    (10, 42);
  filled polygon (23, 0) (58, 0) (58, 16)
    (45, 16);
border:
  outline (23, 0) arc (23, 46) (45, 16) (58,
    16) (58, 0);
  lines (10, 42) (0, 53) (46, 53) (36, 42);
black:
  stippled area light-stipple;
  filled polygon (38, 33) (8, 33) (23, 8)
```

Stipple-area element

Only one stippled-area element per layer in an icon description is supported at this time. However, the grammar allows you to define any number of such stippled-area elements. If there are more than one element defined, G2 will use only one of the stippled-area elements.

Displaying and Printing Stippled Icons

Special limitations apply to stippled areas on different layers drawn in the same color:

- If a stippled area appears on a layer of a given color, and there is another layer of the same color further below it, both layers will be drawn with stippling, just as though both layers were given the identical stipple.

  If the two layers differ in colors, this does not occur.

- Also, if the layer without the stipple appears above the layer with the stipple, rather than below it, this does not occur.

Note The noncolored dots, or "holes", in stippled areas of icon layers show through to whatever is underneath the icon, not simply to whatever is on the next layer down in the icon. It is as though these elements were drawn in the metacolor "transparent", in that elements of an icon layer drawn in the transparent metacolor similarly show "all the way through".

Stippled icons are rendered using solid colors when printed using G2's print facility, for example, using G2's print command on workspaces, the print action, and so on.

Programmatic Access to Stipples

The only access to stipples in icons is through the attribute access facility. A stipple header is an optional field in an icon description structure of the form:

stipple: the symbol <*stipple pattern*>

where <*stipple pattern*> is one of the G2 stipple patterns.

A stippled-area element is represented as a structure of the form:

```
structure(
    type: the symbol stippled-area,
    stipple: the symbol <stipple pattern>,
    points: sequence ( [<point> <point>] )
)
```

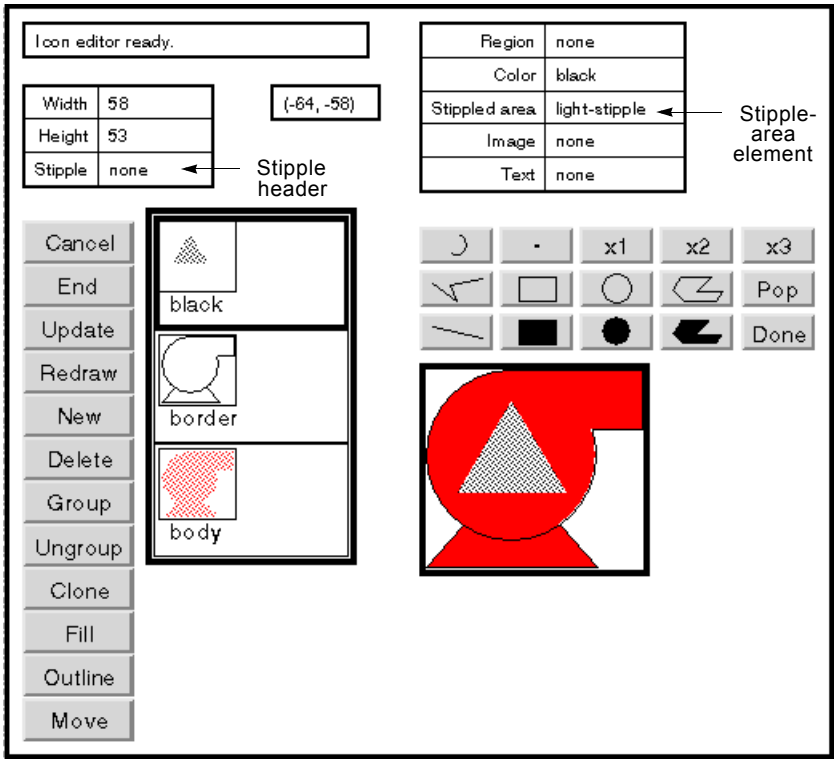
where <*stipple*> is as defined above, and where <*point*> is a structure of the form:

sequence (x: <*integer*>, y: <*integer*>)

The first and second points represent the top-left and bottom-right corners, respectively, of the rectangular area for the stipple. If the sequence is empty, the entire rectangular area of the icon layer is stippled.

Stipples in the Icon Editor

Most developers see stipples through the icon editor, where developers generally view and edit the detailed graphical makeup of an icon. The icon editor provides a way to view and edit both the stipple header and the stipples-area elements of an icon.



The grammar accepted for the stipple header is:

{none | light-stipple | medium-stipple | dark-stipple}

The grammar accepted for the stipple-area element is:

{ none | {light-stipple | medium-stipple | dark-stipple} [<point> <point>] }

where the first and second <point> elements have the grammar:

(<*integer*>, <*integer*>)

and represent the top-left and bottom-right corners, respectively, of the rectangular area to be stippled. If they are not supplied the entire area of the layer is stippled.

Including Externally Created Images

An **image** is a JPEG, GIF, or XBM file that has been made available within G2 via an image definition, as described in Chapter 34, External Images on page 1219. Images typically contain screen captures, scanned-in photographs, or other complex graphics.

You can use the Icon Editor to define a single image as a component of a layer, and to combine such layers into one layer with multiple images. A layer that includes an image can also contain other graphics and text defined with the Icon Editor, just as if no image were present.

An image used in an icon appears in monochrome, even though the original bitmap may be polychrome. An image behaves like any other graphic with respect to occlusion, color, grouping, and region definition. The same image can be included in more than one layer. In each layer, it takes on that layer's color.

The Layer Indicators include an Image Indicator for including an image in a layer. When a layer has multiple images, only the first shows in the Image Indicator. All image components appear in the icon description in the class definition's table.

Various GIFs that can be used as external images in icons are available in the G2 `demos` directory, as described under GIF Files on page 2163.

To include an image in an icon:

1. Select the layer that is to contain the image.
2. Click on the value of the Image Indicator to invoke the Text Editor.

   The editor lists all currently defined images.
3. Edit the Image Indicator to contain the name of the image.

The image appears in the layer as soon as you close the Text Editor.

To put more than one image into a layer, specify each in a separate layer, then combine the layers using the Group button, as described under Creating Groups on page 1660. Only the first image in the combined layer appears in the layer's Image Indicator. Use Ungroup to regain separate access to the components of the combined layer.

Image Size and Icon Size

The size of an image does not affect the icon size. When the icon's width and/or height is larger than the image, the image occupies only a portion of the icon. When the icon's width and/or height is smaller than the image, the image is cropped at the boundary of the icon.

To have an image determine the icon size:

➔ Edit the Icon Editor width and height indicators to have the values of the width-of-image and height-of-image attributes of the image-definition table.

Image definition tables are described under Creating an Image Definition on page 1221.

Image Position

By default, G2 positions an image with its top left corner (the image's (0, 0)) at the top left corner of its layer (the layer's (0,0)). To position an image elsewhere in its layer, you can enter an (X, Y) offset after the image name.

The numbers in the offset represent workspace units, and can be negative. The editor offsets the image horizontally by the amount of the X offset, and vertically by the amount of the Y offset. A negative X offsets the image to the left, and a negative Y offsets the image upwards. Such an offset crops the image on the left and/or top.

To position an image in a layer:

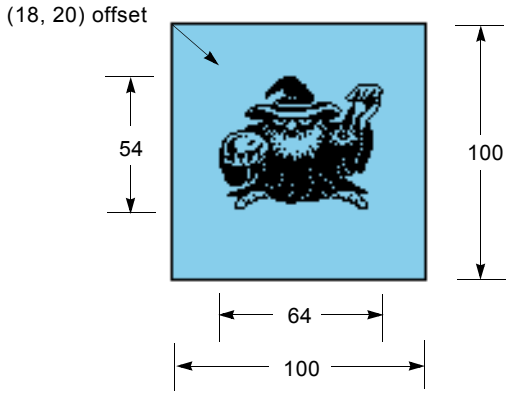
1 Click in the image value to edit the image name.

After the name of the image, the editor prompts you with at.

2 Enter an (X, Y) offset, for example:

```
wizard at (18, 20);
```

The image wizard now appears in its layer with its upper left corner 18 units to the right and 20 units down from the upper left corner of the icon. If the size of the icon is 100x100, and the size of the image is 64x54, this offset positions the image just above the center of the icon:



Defining Regions

A region is a collection of one or more layers that have the same region name. The constituent layers all have the same color, but appear separately in the layers pad.

The layers of a region can be interspersed freely with single layers, groups, and layers of other regions. You can use this property in combination with the change color action to define complex alterations of icon appearance.

The members of a region are included in the region by the fact that they have the same region name. Region names are arbitrary symbols: a region exists as soon as you name it as the region of one or more layers. A layer's region name (if any) appears at the bottom of each layer in the region. It also appears in the Region Indicator when the layer is selected.

To add a layer to a region:

1 Select the layer.

The first line of the Region Indicator shows the layer's region, or none if the layer does not belong to a region.

2 Edit the region name in the Region Indicator to specify the desired region.

You can add additional layers to a region at any time. When you add a layer to a region, the layer takes on the color of the region. If you change the color of any layer in a region, the color of all other constituent layers changes to the new color.

Creating Groups

Any layer that contains two or more graphical elements constitutes a **group** of elements. You can group the elements on different layers onto a single layer, or ungroup a layer into its constituent elements.

To combine two layers into a group:

1 Position the layers consecutively in the layers pad.

2 Select the upper layer of the intended group.

3 Click the Group button.

The editor combines the selected layer and the layer immediately below it into a single layer. The layer appears in the position of the upper constituent layer, and shows all graphics that were on either layer. If the layers differed in color, the color of the upper layer predominates.

You can use Group to add the elements on additional layers to a group at any time. Grouping does not preserve the individual identities of the grouped layers: it produces a flat collection of graphical elements, indistinguishable from a single layer on which all constituent elements were drawn without using Group.

To decompose a group:

1 Select the group.

2 Click the Ungroup button.

The editor creates a separate layer for every graphical element in the group that is not a point, and another (if needed) that contains all points (if any). Each layer has the color and region of the decomposed group. The topmost layer is in the position previously occupied by the group; the other layers are strung out below it.

To ungroup a layer that contains points only:

1 Select the group.

2 Click the Ungroup button.

The editor notes that the layer contains only points, and ungroups each point onto a separate layer. Each layer has the color and region of the ungrouped layer.

Saving and Canceling Changes

After you have edited an icon, you can do one of three things:

- Save changes and remain in the Icon Editor.
- Save changes and leave the Icon Editor.
- Discard changes and leave the Icon Editor.

To save changes and remain in the editor:

➔ Click the Update button.

The editor saves all changes, but remains active. You can now make further changes if desired.

To save changes and leave the editor:

➔ Click the End button.

The editor saves all changes and exits.

To discard changes and leave the editor:

➔ Click the Cancel button.

The editor discards any changes, then exits. Any changes previously saved using the Update button remain in effect.

Tips for Working with Icons

To facilitate creating a complex icon, first design the icon completely using graph paper on which each gridline represents one workspace unit.

To make working with a complex icon easier, temporarily change the colors of some of its layers to transparent. This makes the icon's appearance less cluttered, so that you can see other graphic elements in the icon.

Many icons look better when their filled elements are surrounded by a border in a contrasting color. You can use Clone and Outline to create such a border conveniently.

To create a contrasting border:

1. Select the layer that you want to outline.
2. Clone the layer.
3. Change the color of the cloned layer to be the outline color.
4. Use the Outline button to convert the layer's graphics to outlined graphics.

Editing Icons Textually

An icon exists as a textual description of its appearance. This description is the value of the icon-description attribute of the icon's class definition. The Icon Editor translates an icon's description into graphical form, and allows you to edit the icon definition graphically, as described earlier in this chapter.

Alternatively, you can edit an icon's definition textually, either creating it from scratch, or modifying it as desired. Textual icon editing can be useful, because the Icon Editor does not allow you to modify graphical elements once you have created them; textual editing allows any type of modification.

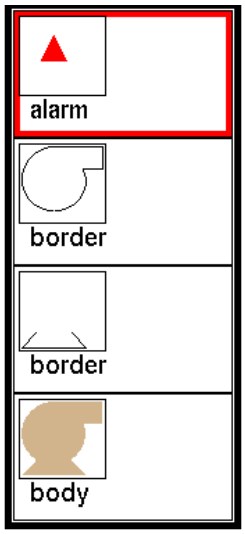
Thus you can use the Icon and Text Editors together to define an icon. The Icon Editor establishes the icon's overall properties and shows its current appearance; the Text Editor allows modifications that fine-tune the icon to have precisely the desired look.

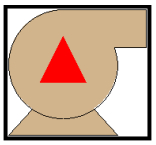
Icon Description Language Example

A simple icon description consists of:

- The width and height of the icon.
- For each region, the region's color.
- For each layer, the layer's region, followed by a description of the graphical elements of the layer. The layers are listed from the bottom up.

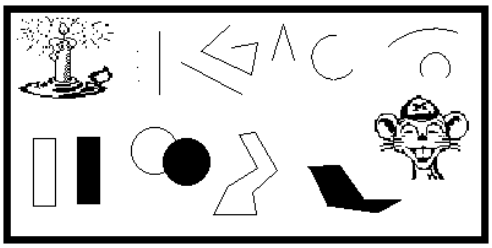
For example, the following figure shows the layers pad, icon viewer, and text description of the icon shown in the Icon Editor in the beginning of this chapter:

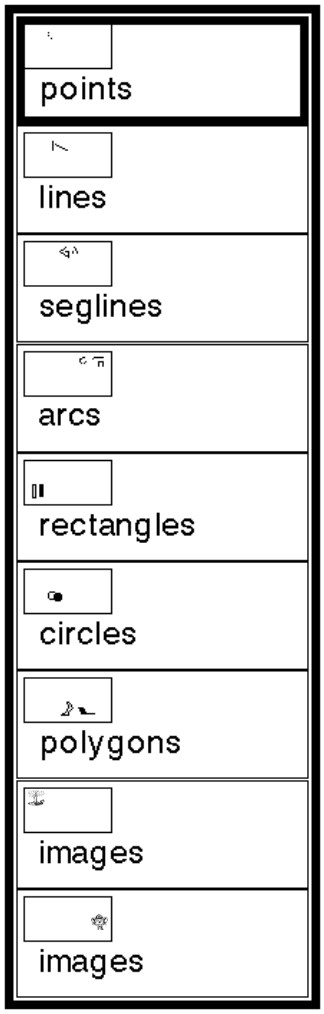




```
width 58; height 53;
body = tan, border = black, alarm = red;
body:
    filled circle (0, 23) (23, 0) (46, 23);
    filled polygon (36, 42) (46, 53) (0, 53) (10, 42);
    filled polygon (23, 0) (58, 0) (58, 16) (45, 16);
border:
    lines (10, 42) (0, 53) (46, 53) (36, 42);
border:
    outline (23, 0) arc (23, 46) (45, 16) (58, 16) (58, 0);
alarm:
    filled polygon (33, 31) (13, 31) (23, 11)
```

The following figure shows examples of many of the elements that an icon description can contain. Each element appears in both Icon Editor and textual form. Each layer belongs to a region whose name is the type of graphics drawn on the layer.






```
width 293; height 137;
images = black, polygons = magenta,
   circles = coral, rectangles = yellow,
   arcs = red, seglines = green, lines =
   cyan, points = black;
images:
   image gopher at  (225, 50);
images:
   image candle;
polygons:
   outline (152, 74) (166, 98) (151, 112)
      (153, 126) (126, 126) (135, 108)
      (154, 97) (142, 76);
   filled polygon (186, 96) (206, 94)
      (221, 116) (245, 117) (230, 125)
      (198, 121) (188, 99);
circles:
   circle (74, 86) (89, 71) (104, 86);
   filled circle (94, 93) (109, 78) (124,
      93);
rectangles:
   outline (12, 78) (12, 121) (26, 121)
      (26, 78);
   filled rectangle (40, 77) (54, 119);
arcs:
   lines (205, 13) arc (191, 34) (214,
      35);
   lines (236, 20) arc (252, 11) (281,
      12);
   lines (259, 40) arc (258, 30) (274,
      41);
seglines:
   lines (138, 21) (154, 18) (150, 38)
      (118, 23) (136, 7);
   lines (177, 29) (170, 6) (163, 29);
lines:
   lines (106, 30) (144, 50);
   lines (92, 11) (92, 50);
points:
   point (79, 42);
   point (78, 33);
   point (78, 23)
```

Icon Description Language Grammar

This section describes the grammar of the icon description language. You don't need to study this grammar unless you intend to edit icons textually. When studying the grammar, you can skip over any sections that do not relate specifically to your needs. See the preceding figure for help in interpreting this grammar.

The optional icon-background-layer section is described under Specifying an Icon Background Layer on page 1668. The optional variables section is described in Animated Icons on page 1670 and subsequent sections.

```
width: integer; height: integer;
    [{region-name = color-name} [, ...] ; ]
[variables: {variable-name = variable-value} [, ...] ;]
[icon-background-layer:
    {color image image-name at (x-pos, y-pos) | region-name | color-name} ]
    {region-name: {graphic-definition;} [...] } [...]

graphic-definition :=
    point-definition |
    line-definition |
    segmented-line-definition |
    arc-definition |
    outlined-rectangle-definition | filled-rectangle-definition |
    outlined-circle-definition | filled-circle-definition |
    outlined-polygon-definition | filled-polygon-definition |
    text-definition |
    image-definition
```

The width and height can each be set to a maximum of 100,000, subject to a maximum area of 40,000,000. This means, for example, that you can create a 100,000x400 icon.

Note Very large icons may quickly exhaust available graphics memory on Windows platforms, which will lead to an abort of the G2 or TW process displaying the icons. G2 and Telewindows may draw very large icons incorrectly on UNIX platforms, because X Windows restricts drawing coordinates to 16-bit signed integers (-32768 through +32767). You can also experience drawing problems with very long diagonal connections.

You can provide a symbol of the form RGB*rrggbb* as a valid color name, where *rr*, *gg*, *bb*, are the 8-bit hex values for red, green, and blue. For details, see Other Literal Terms on page 2155.

The various *graphic-definition*s are based on a small number of primitives which they use in ways that are sometimes not obvious. The rest of this section gives details. All numeric coordinates (*x*, *y*) are integers (*not* integer expressions) in

workspace units measured from the upper left corner of the icon, whose coordinates are (0,0).

Specifying Points

point-definition := point (*x*, *y*);

The coordinates specify the location of the point.

Specifying Lines

line-definition := lines (*x*, *y*) (*x*, *y*);

The two pairs of coordinates specify the beginning and end of the line.

Specifying Segmented Lines

segmented-line-definition := lines (*x*, *y*) (*x*, *y*) [(*x*, *y*)]...;

Each coordinate pair specifies the location of one vertex of the segmented line.

Specifying Arcs

arc-definition := lines (*x*, *y*) arc (*x*, *y*) (*x*, *y*);

The first, second, and third coordinate pairs denote the first, second, and third points that you click to define an arc with the mouse. The arc is a circular section that extends from the first point through the second point to the third point.

Specifying Outlined Rectangles

outlined-rectangle-definition := outline (*x*, *y*) (*x*, *y*) (*x*, *y*) (*x*, *y*);

The four coordinate pairs specify the four corners of the rectangle.

Specifying Filled Rectangles

filled-rectangle-definition := filled rectangle (*x*, *y*) (*x*, *y*);

The two coordinate pairs specify the upper left and lower right corners of the rectangle.

Specifying Outlined Circles

outlined-circle-definition := circle (*x*, *y*) (*x*, *y*) (*x*, *y*);

The three points denote an arc that, when completed, forms the circle. The same three points, given in order to an arc definition, would produce the underlying arc.

When you specify an outlined circle definition textually, you can give any three points on the circumference of the circle.

When the Icon Editor generates the three points of an outlined circle definition from graphical input (via the Circle or Filled Circle button), the points specify a semicircular arc that opens downwards. If the center of the circle is at (XC, YC) and the radius is R, the three points are (XC-R, YC) (XC, YC-R) (XC+R, YC). Inversely, if the three points are (X1, Y1), (X2, Y2), and (X3, Y3), the center of the circle is (X2, Y1) and the radius of the circle is (X3-X1)/2.

Specifying Filled Circles

filled-circle-definition := filled circle (*x*, *y*) (*x*, *y*) (*x*, *y*);

The three points are the same as those in an *outlined-circle-definition*.

Specifying Outlined Polygons

outlined-polygon-definition := outline (*x*, *y*) (*x*, *y*) (*x*, *y*) [(*x*, *y*)]...;

Each point specifies the location of one vertex of the polygon.

Specifying Filled Polygons

filled-polygon-definition := filled polygon (*x*, *y*) (*x*, *y*) (*x*, *y*) [(*x*, *y*)]...;

Each point specifies the location of one vertex of the polygon.

Specifying Text Components

text-definition := text *string* at (*x-position*, *y-position*) in *font-size*

The elements of the definition are:

string: The text of the component.

x-position: The position of the left edge of the text relative to the left edge of the icon.

y-position: The position of the baseline of the text relative to the top of the icon.

font-size: Any standard G2 font size: small, large, or extra-large.

Including External Images

image-definition := *image-name* (*x*, *y*);

The *image-name* is a symbol that is the name of an image definition object. The coordinates give the offset of the upper left corner of the image from the upper left corner of the icon.

Using the Icon and Text Editors Together

To use the Icon and Text Editors together, invoke them both on the same icon description, and position them so that both are fully visible. You can now use

either editor to examine and change the icon, and the other editor to see the effect of the changes.

The two editors function independently even when you use them on the same icon. Neither editor knows what happens in the other, or is aware of changes saved by the other. After you save changes made in either editor, you must *close and reopen* the other editor to show the changes there also.

Be careful not to make concurrent changes in both editors, or the changes saved by one editor will be overwritten when you save the changes made in the other.

Specifying an Icon Background Layer

G2 provides an optional icon layer called the **icon-background-layer**, which appears behind all other components of an icon. This layer can contain a monochrome or polychrome image, or can be of any G2 color. Unlike imported images in other layers, a polychrome image in an icon background layer appears in polychrome on the screen.

An image in a background layer is called a **background image**, and the color of a colored background layer is called the **background color**. An icon background layer cannot simultaneously have both an background image and a background color.

Specifying a Background Image

To specify an icon background image for a class:

1. Use the Text Editor to give the class definition's icon description an icon-background-layer section.
2. Specify the name and location of the image using the grammar:

   color image *image-name* at (*x-position*, *y-position*)

   where *image-name* is the name of an image-definition, and *x-position* and *y-position* give the position of the upper left corner of the image relative to the upper left corner of the icon.

For example, if corporate-logo is an image, the definition:

```
icon-background-layer: color image corporate-logo at (0,0)
```

displays the image in the background layer at the indicated coordinates.

Notes on Background Images

When you use the Icon Editor to edit an icon that includes a background image, the image appears in the Icon Viewer, but cannot be edited: it appears only to facilitate designing the rest of the icon.

If a background image is larger than the icon that displays it, G2 clips it to fit the available area, both in the Icon Viewer and on a workspace.

If you specify a background image that does not exist, G2 puts a warning in the class definition's notes attribute, but does not signal an error. The icon looks just as it would if the unresolved definition did not exist.

A background image definition applies to all instances of a class: you cannot change background images per-instance.

Various GIFs that can be used as icon background images are available in the G2 `demos` directory, as described under GIF Files on page 2163.

Specifying a Background Color

A background color can be specified explicitly or by naming a region, in which case the background layer has the color of that region. When a region gives the background color, the color can be changed programmatically. A background color completely fills the background of an icon, irrespective of the icon's size.

To specify an icon background color for a class:

1. Use the Text Editor to give the class definition's icon description an icon-background-layer section.
2. Specify the value of the layer to be either:
   - Any G2 color.
   - A *region-name*.

For example:

```
icon-background-layer: red
```

or:

```
icon-background-layer: my-region
```

where my-region is any defined region.

To change the icon background color of an instance:

1. Specify the icon background color in the class icon description by giving a *region-name* rather than a specific color.
2. Execute this action:

   change the *region-name* icon-color of *instance* to *color*

where:

region-name: the name of the region that specifies the icon background color

instance: the instance whose background color is to change

color: any G2 color

For example:

```
change the my-region icon-color of my-instance to green
```

Animated Icons

G2 allows you to:

- Specify almost any element of a class's icon description as an **icon variable**.
- Use the conclude action to change the value of the icon variable in an instance.

The instance's icon immediately changes to reflect the new value. Other instances are unaffected. Changes to icon variable values are permanent: resetting the KB does not reset the value.

Note Do not confuse icon variables with G2 variables. An icon variable is not an object: it is just a symbol that has a value.

You can use icon variables to specify:

- The x-position and/or y-position in any coordinate pair *except* the position of a background image.
- The text of a text component.
- The font size of a text component.
- The name of the image in any image component *except* a background image.

You *cannot* use icon variables to specify:

- The width or height of an icon.
- A region name or color.
- The essential type of a graphical component.
- Any property of a background image.

You cannot use icon variables to specify width and height because the width and height in an icon description are functionally icon variables already, as described under Changing Width and Height on page 1676.

If you change icon appearance so that an icon component extends outside the icon border, G2 clips the component. If the entire component is outside the border, it does not appear at all.

Defining and Using Icon Variables

All icon variables (except width and height) are defined in the variables section of the icon description. The syntax of that section is:

[variables: {*variable-name* = *variable-value*} [, ...] ;]

For example:

```
variables: my-variable = 20, your-variable = 30;
```

defines the icon variables my-variable and your-variable, and gives each the default value shown. The values are default values rather than initial values because they are set once when a class is instantiated, and are not reset when the KB is reset, as initial values of variables and parameters are.

Once an icon variable is defined, you can use it in place of any component of the icon description that can be given by an icon variable. For example, given the above definitions, the graphic definition:

```
lines (10, my-variable) (your-variable, 40);
```

defines a line that runs from (10, 20) to (30, 40).

You can define and use icon variables with the Text Editor, the Icon Editor, or both in combination. The most convenient technique varies with the purpose of the icon variables, as described in this section.

You can also specify any *x-position* and/or *y-position* as an integer expression, as described under Specifying Locations with Expressions on page 1675. For simplicity, the following instructions assume that locations are given by single icon variables.

If you follow the guidelines described under Using the Icon and Text Editors Together on page 1667, you can use the Icon and Text Editors in parallel. This technique allows you to see the icon's visual and textual representations simultaneously, which can be helpful in designing complex icons.

Specifying Graphical Positions with Icon Variables

The simplest use of icon variables is to specify the positions of the defining points of graphical icon components: lines, rectangles, circles, and the like. You can then cause the icon components to move and reshape by changing the values of the icon variables.

To specify graphical positions using icon variables:

1. Create the icon component(s) whose positions will be given by icon variables, in approximately their initial locations, as described under Creating Graphics on page 1648.
2. Use the End button to save the icon description.
3. Use the Text Editor to edit the icon description.
4. Create a variables section (or update an existing one) that defines all needed *x-position* and *y-position* icon variables. Give each icon variable the desired default value.
5. Substitute icon variables for *x-position*s and *y-position*s in the icon description as desired.
6. Exit the Text Editor to save the icon description.

For example:

```
width 100; height 100;
variables: x = 89, y = 74, label = "George";
blue: filled rectangle (0,0) (100, 100)
linen: filled rectangle (13, 18) (X, Y)
forest-green: text label at (20, 50) in large
```

Specifying Text Components with Icon Variables

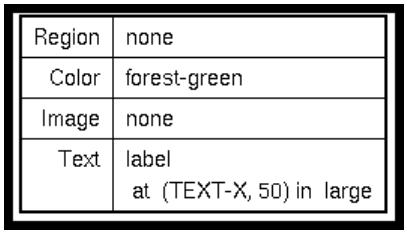
Defining a text component using icon variables is similar to defining a graphical component using them. The differences are:

- A text component includes a string and a font, which can be given by icon variables.
- When a layer contains only one text component, you can use the Icon Editor to partially automate the work of specifying the icon variables.

To define a single text component that uses icon variables:

1. Use the Icon Editor to define the text component, as described under Defining Text Components on page 1652.
2. In the Text Indicator, specify any available symbol for any or all of:
   - The *string* itself.
   - The *x-position* of the text string.
   - The *y-position* of the text string.
   - The *font-size* in which the string appears.

For example:



3 Use the End button to save the icon description and close the Icon Editor.

G2 adds a variables section to the icon description if one did not already exist, and defines in the icon description's variables section any icon variables used in the text description that were not already defined there. Each such definition specifies a default value for the icon variable:

- "undefined-text" for a *string*.
- large for a *font-size*.
- 0 for an *x-position* or *y-position*.

The Status Indicator reports each icon variable that the editor adds to the icon description's variables section.

4 Use the Text Editor to assign a new value to any icon variable that does not have the desired default value.

5 Close the Text Editor to save the modified icon description.

To define multiple text components that use icon variables:

1 Use the Icon Editor to specify each component in a separate layer.

2 Use the Text Editor to complete the definitions as needed.

3 Use the Icon Editor Group button to combine the layers into one layer.

Use Ungroup to regain separate access to the components of the combined layer.

To define text components textually:

1 Use the Text Editor to enter the text definitions into the icon description, as described under Specifying Text Components on page 1667.

2 Use icon variables to specify parts of the definition as needed.

3 Create and/or edit the variables section of the icon description to define the icon variables and give them default values.

4 Close the Text Editor to save the modified icon description.

Specifying Image Components with Icon Variables

Defining image components is similar to defining text components.

To define a single image component that uses icon variables:

1. Use the Icon Editor to specify the image component, as described under Including Externally Created Images on page 1658.
2. In the Image Indicator, specify any available symbol for any or all of:
   - The *image-name* itself.
   - The *x-position* of the image.
   - The *y-position* of the image.
3. Use the Update or End button to save the icon description.

   At this point, G2 has no way to know whether you intended the symbol that specifies the *image-name* as a literal image name or as an icon variable, so the name does not appear in a variables section. If the icon description contains nothing that G2 knows to be an icon variable, the definition has no variables section.
4. If you want *image-name* to be an icon variable, use the Text Editor to either add a variables section that defines *image-name*, or to add the definition of *image-name* to the existing variables section. The needed definition gives *image-name* a value that names an image-definition: the same value that you would have put directly into the Image Indicator if you were not using an icon variable.

   If you used icon variables to specify the *x-position* and/or *y-position*, the icon description contains a variables section that defines the icon variable(s) with a default value of 0.
5. Use the Text Editor to assign a value to any *x-position* and *y-position* icon variable that does not have the desired default value.
6. Close the Text Editor to save the modified icon description.

To define multiple image components that use icon variables:

1. Specify each component in a separate layer.
2. Use the Text Editor to complete the definitions as needed.
3. Combine the layers into one layer using Group.
4. Use Ungroup to regain separate access to the components.

To define image components textually:

1. Use the Text Editor to enter the image definitions into the icon description.
2. Use icon variables to specify parts of the definition as needed.
3. Create and/or edit the variables section of the icon description to define the icon variables and give them default values.
4. Close the Text Editor to save the modified icon description.

Specifying Locations with Expressions

Any *x-position* and/or *y-position* in an icon description can be specified as an integer expression consisting of any number of symbols and integers combined by addition and subtraction. You can use the Icon Editor to specify such expressions in the Text and Image Indicators, and in the Text Editor to specify them in any context.

Any symbol that appears in an integer expression specified in the Icon Editor, and was not already defined in the icon description's variables section, automatically appears in the variables section with a default value of 0, just as a single icon variable does. Edit the default value as needed to provide a different value.

When an expression in an icon description contains more than one constant, G2 collapses all of them into a single constant when it saves the icon description.

Manual Layer Positioning and Icon Variables

If you use the Icon Editor to manually change any position given by an icon variable or an expression, the editor adds to the icon variable or expression a constant that reflects the change.

- For a location specified by an icon variable, the effect is to convert the specification to an expression.
- In an expression that already contained a constant, the editor combines the new and existing constants into a single value.

If you move a text component or an imported image whose position is defined using icon variables and appears in the Text Indicator or Image Indicator, the constants that the editor adds to the icon variables appear in the relevant indicator.

Errors in Icon Variable Specifications

Whenever you use an icon variable, both the Icon Editor and the Text Editor check that the usage is syntactically correct. If an error exists, the editor indicates it, and will not save the icon description until the error is corrected.

Animating Icons

When any part of a class's icon description is given by an icon variable, you can use the conclude action to change the value of that icon variable for an instance of the class whenever G2 is running. The appearance of that instance changes to reflect the new value(s). Other instances of the class are unaffected. Changes to icon variable values are permanent: resetting the KB does not reset the value.

G2 performs type checking on all changes to icon variables, and signals an error if you attempt to conclude a value that is incompatible with the icon variable's usage.

Changing Width and Height

You cannot use icon variables to specify width and height because the width and height in an icon description are functionally icon variables already. Therefore, G2 lets you conclude values to them just as you can to icon variables that appear explicitly. This section hereafter refers to width and height as if they were ordinary icon variables.

You change icon size by concluding integer values into width and height. The maximum allowable value for each dimension is 10,000, subject to a maximum area of 40,000,000. This means, for example, that you can create a 100,000x400 icon. The minimum allowable value is 1.

Referencing Icon Variables

Suppose that the icon description of the class tank specifies:

- A width and height of 50.
- Three icon variables, x, y, and z, whose values are all 0.
- No other icon variables.

If an instance of tank named tank-1 exists, and none of these values has been changed by a conclude, the value of:

```
the icon-variables of tank-1
```

is:

```
structure (width: 50, height: 50, x: 0, y: 0, z: 0)
```

Replacing Icon Variable Values

You can replace all icon variable values in a single operation by concluding a structure to them that specifies the new values. Thus:

```
conclude that the icon-variables of tank-1 =
structure (width:100, height: 100, x: 0, y: 50, z: 75)
```

sets tank-1's icon variables to have the values shown.

Replacing Icon Variable Text

You can replace icon variable text by concluding its value. For example, suppose you have defined an icon variable named text-var. The first example below concludes the value of the text-var icon variable, using a text parameter. The second example concludes the text-var icon variable subattribute of the icon-variables structure.

```
conclude that the text-var of the icon-variables of test-obj-1 = text-parameter-1

conclude that the icon-variables of test-obj-1 = structure (text-var: "new-text",
width: 100, height 100)
```

Merging Icon Variable Values

You can change some icon variable values while preserving the existing values of others. The technique is:

1. Obtain the existing icon values
2. Change values in the returned structure as desired.
3. Conclude the changed structure back into the icon variables.

For example, if tank-1's icon variable values are:

```
structure (width:100, height: 100, x: 0, y: 50, z: 75)
```

then executing:

```
current-icon-variables: structure;
new-icon-variables: structure;
begin
    current-icon-variables = the icon-variables of tank-1;
    new-icon-variables =
        change-attribute (current-icon-variables, Z, 30);
    conclude that the icon-variables of tank-1 = new-icon-variables; end
```

results in the icon variables of tank-1 being:

```
structure (width:100, height: 100, x: 0, y: 50, z: 30)
```

The same logic could be expressed more compactly as:

```
conclude that the icon-variables of tank-1 =
change-attribute (the icon-variables of tank-1, z, 30)
```

Alternatively, you can use a subattribute reference to conclude a single variable in the icon variable structure. Thus the effect of the preceding code could also be produced by executing:

```
conclude that the z of the icon-variables of tank-1 = 30;
```

This technique eliminates the overhead of explicitly retrieving, setting, and concluding values. However, a sequence of concludes that use subattribute references executes more slowly than a single conclude that accomplishes the effect of them all, and might cause icon appearance to pass through unwanted intermediate stages as the concludes execute one by one.

Conveniently Merging New and Default Values

When you want every icon variable to have either its class default value or a value to be set in a conclude action, you can conclude a structure that references only the icon variables that are to have non-default values.

For example, if the tank class is as described under Referencing Icon Variables on page 1676, and tank-2 has default values for all icon variables (width: 50, height: 50, x: 0, y: 0, z: 0), executing:

```
conclude that the icon-variables of tank-2 = structure ( z: 75)
```

sets tank-2's icon variables to:

```
structure (width: 50, height: 50, x: 0, y: 0, z: 75)
```

This technique eliminates the overhead of specifying icon variable values that are to be left unchanged, but it resets any icon variables not referenced in the concluded structure to have the default values specified for them in the class definition. For example, executing:

```
conclude that the icon-variables of tank-2 = structure (width: 100)
```

on the modified tank-2 sets tank-2's icon attributes to:

```
structure (width: 100, height: 50, x: 0, y: 0, z: 0)
```

The value of z, being unspecified in the concluded structure, has reverted from 75 to the class default value.

47

The Inspect Facility

Describes how to use the Inspect facility to search for items.

Introduction

You use the Inspect facility to search a knowledge base (KB) for items based on their type, class, attributes, and location. You can use the Inspect facility to:

- Show short representations of items on the Inspect workspace.
- Display the class, module, workspace, invocation, and method hierarchies.
- Create a file containing the definitions of items.
- Go directly to a particular item in the knowledge base.
- Display a representation of an item's table on a workspace.
- Replace and highlight text in the knowledge base.
- Check for consistent modularization.
- Recompile specific items.

You can also apply a variety of filter expressions to limit the search when inspecting classes of items.

Once the Inspect facility has located the item(s), you can interact with the item in a variety of ways, including going to the item, displaying the item's table, and describing the item.

The Inspect facility runs in the background, which means you can launch multiple searches simultaneously, and users can access the KB while the search is underway.

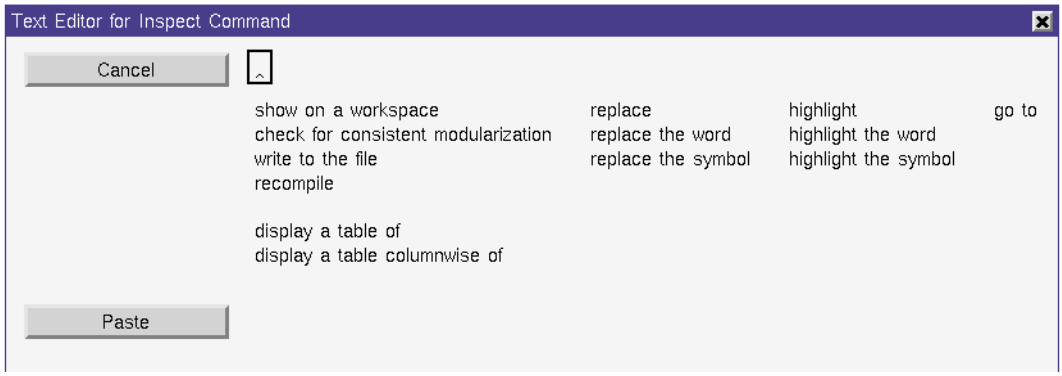
A number of system procedures exist for performing Inspect commands programmatically. For details, see the *G2 System Procedures Reference Manual*.

Using the Inspect Facility

To use the Inspect facility:

1 Select Main Menu > Inspect.

G2 displays the following edit workspace with the Inspect commands.

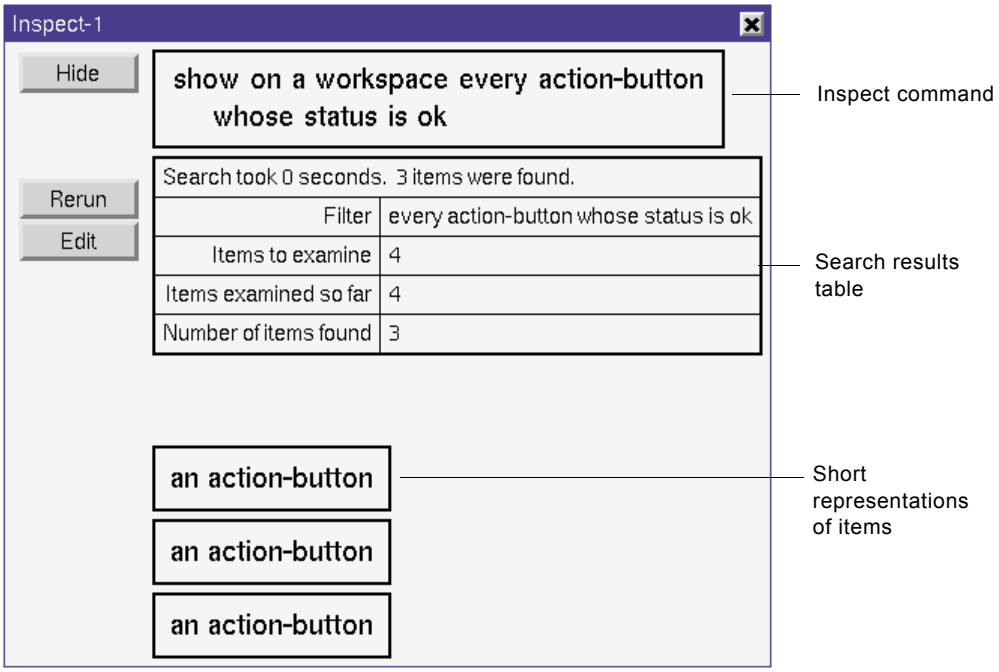


2 Enter an Inspect command from the list of commands shown above.

For example, to display every action button whose status is OK, enter the following command:

```
show on a workspace every action-button whose status is ok
```

G2 displays an Inspect workspace such as this:



Note: There may be more items than can be contained on the workspace.

The Inspect workspace displays the progress and results of an inspection in its search results table. If Inspect is searching for items, the workspace also shows a short representation of each item that it finds.

3 To hide the Inspect workspace, click the Hide button on the workspace.

The Inspect workspace contains the following buttons:

Button	Description
Hide	Hides the temporary workspace by deleting it.
Suspend	Suspends the current search. This button only appears while the search is in progress.
Continue	Continues searching after suspending a search.
Rerun	Reruns the current Inspect command.

Button	Description
Edit	Opens the editor for editing the current Inspect command.
End	Executes the current Inspect command when the cursor is in the text editor.

Interacting with Items on the Inspect Workspace

Each item in the Inspect workspace behaves in many ways like a normal item; you can display its table in the workspace and describe the item. You can also place your cursor on the actual item.

To display an item's table:

➔ Click on the item to display its menu and select table.

To describe the item:

➔ Click on the item to display its menu and select describe.

To go directly to the item:

➔ Click on the item to display its menu and select go to original.

Tip You can accomplish the same thing by using the go to command described in Locating Items in Your KB.

To transfer the item to another workspace:

➔ Click on the item to display its menu and select transfer.

Showing Items on a Workspace

You can use the Inspect facility to search the KB for particular types of items by using the show on a workspace command as described in Using the Inspect Facility. G2 can show specific items or classes of items, and graphical representations of various hierarchies in the KB.

Syntax

show on a workspace
{ *item* |
  the *class-name* named *item-name*} |
  every *class-name* [*filter*] |
  the workspace hierarchy [of { *kb-workspace-name* | *object-name* }] |
  the class hierarchy [of *class-name*] |
  the module hierarchy [of *module-name*] |
  the procedure {caller | calling} hierarchy of *procedure-name* |
  the method hierarchy of *method-name* |
  method inheritance path for *class-name* [and the method *method-name*] }

Showing Items and Classes

You can use the show on a workspace Inspect command to show a particular item, or a named item of a particular class. Here are some examples:

```
show on a workspace valve-1
```

```
show on a workspace the node named server-node
```

You can also show classes of items and class of items that meet a particular criteria. When searching for classes of items, you can specify a filter to qualify the search. For information about filters, see Filtering Classes of Items. For example:

```
show on a workspace every rule
```

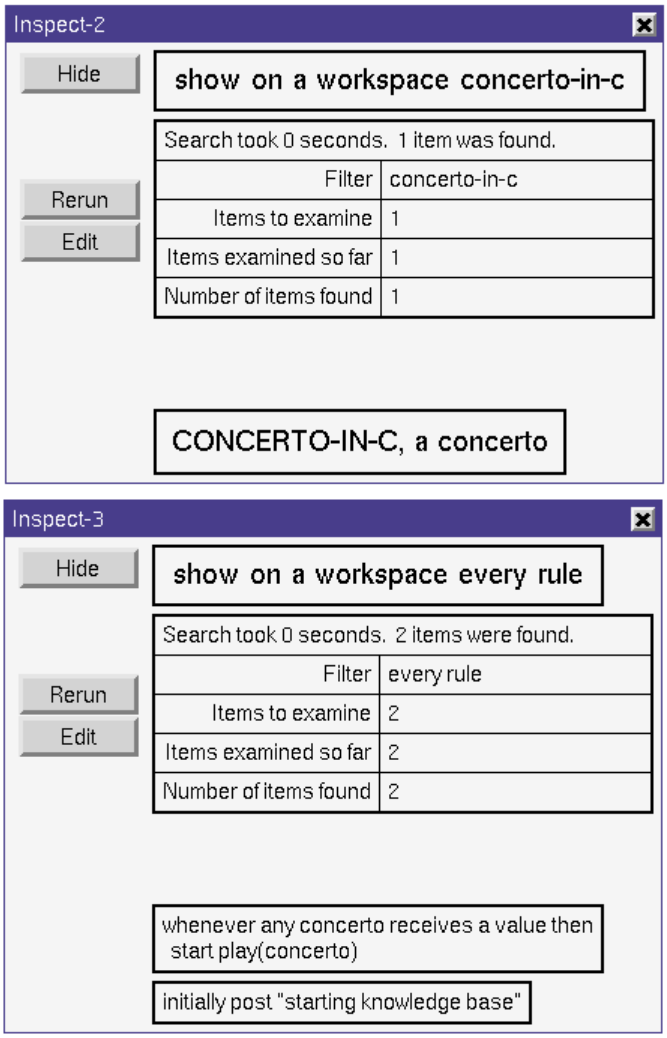
```
show on a workspace every class-definition containing pressure
```

The following command shows every item with permanent changes to attribute values and newly created items:

```
show on a workspace every item with unsaved changes
```

Tip When inspecting classes of items using the every syntax, specify a class as low as possible in the class hierarchy to uniquely identify the item(s). If you specify a class that is very high in the hierarchy, such as the object class, the Inspect facility must search through all items of the specified class, which can be time consuming.

G2 displays a short representation of each item that meets the criteria, or the item itself. This figure shows two Inspect workspaces, one with a short representation of an item, and the other with two actual rules.



Note If more than one item has the same name, G2 shows you all of them.

Showing Items with Unsaved Permanent Changes

To show items with unsaved permanent changes:

➔ Enter one of these commands in the edit box:

show on a workspace every *class-name* with unsaved changes

display a table of every *class-name* with unsaved changes

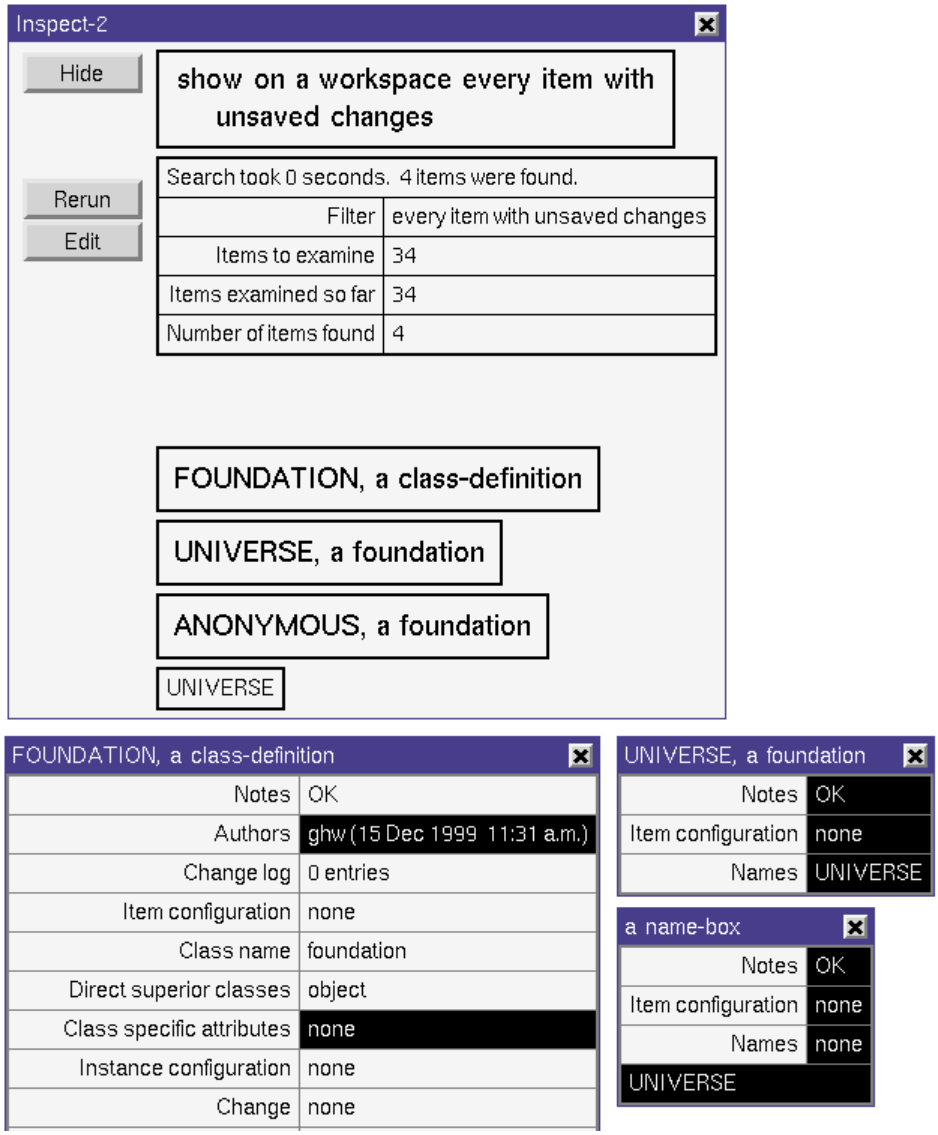
To access permanent change information from an item's short representation:

➔ Select show unsaved attributes from the item menu.

The item's permanently changed attributes are highlighted in the attribute table that appears. All attributes are highlighted for a new item. An item that has no unsaved permanent attribute changes has no show unsaved attributes menu option.

The next example shows unsaved permanent-change information accessed through the Inspect facility and from item menus. The only changes made to the KB were:

- The class-specific-attributes attribute of foundation class was changed from a one-attribute value to none.
- The item universe was created and named.



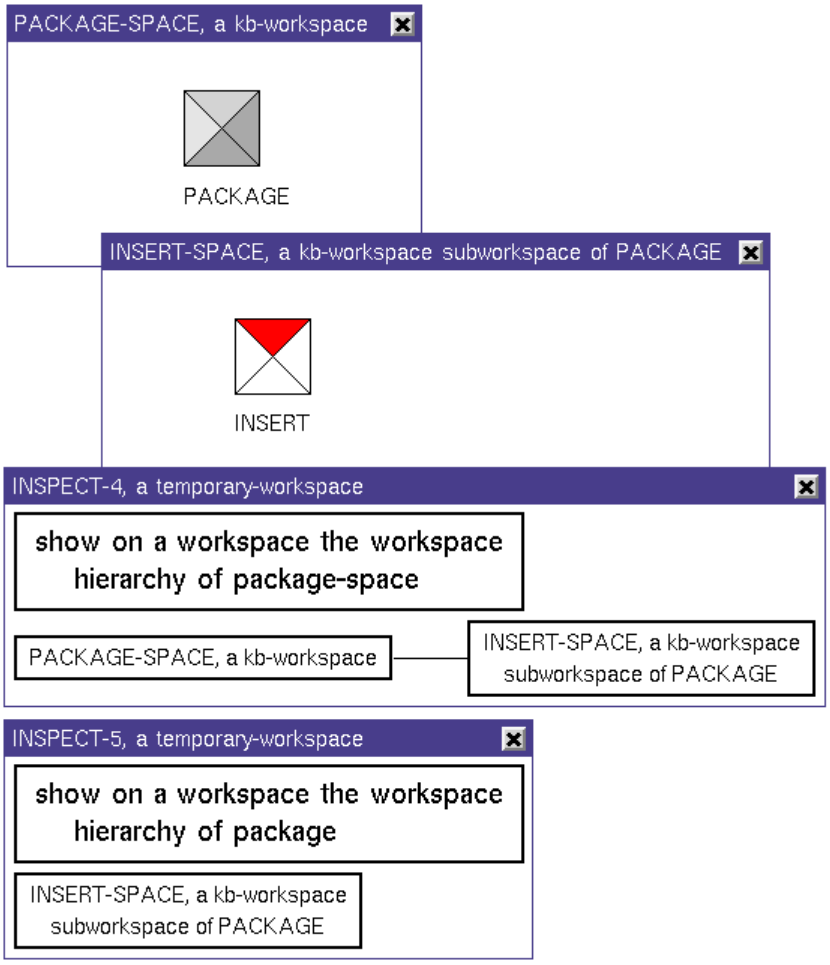
The class-definition and the item-representation and name box for universe have a show unsaved attributes menu choice. The class-definition has its two changed attributes highlighted, and all the attributes are highlighted for universe item and its name-box because they are new items. Because anonymous has lost an attribute, it is shown on the Inspect workspace, but the item does not have a show unsaved attributes menu choice.

Note Executing a show on a workspace every item with unsaved changes Inspect command immediately after launching your G2 process shows changed items. They are the system tables that have been created during G2 initialization.

Showing the Workspace Hierarchy

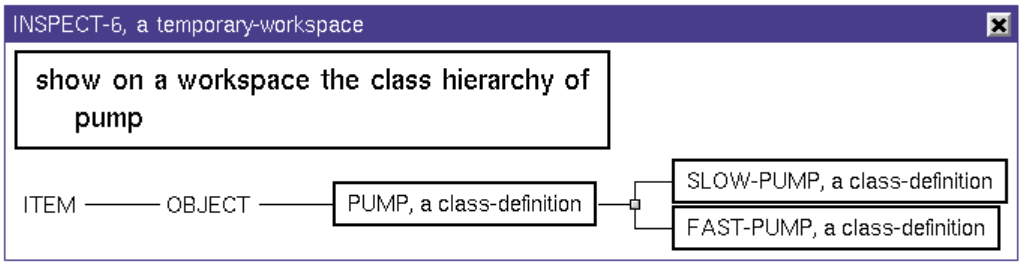
You can show the workspace hierarchy of a particular workspace or object, and you can show the overall workspace hierarchy. G2 displays a graphical representation of the specified workspace hierarchy on a temporary workspace.

The following example shows the workspace hierarchy of a workspace and an item:

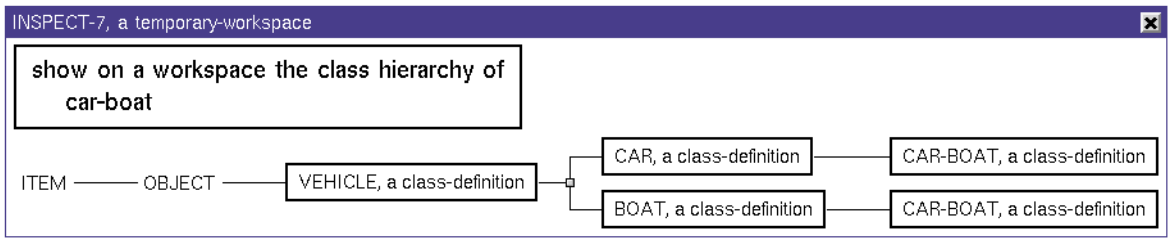


Showing the Class Hierarchy

You can show the class hierarchy for a particular class or all classes. G2 displays a temporary workspace that shows a graphical representation of the class hierarchy. When showing the class hierarchy for a particular class, G2 also displays all the superior classes. For example:

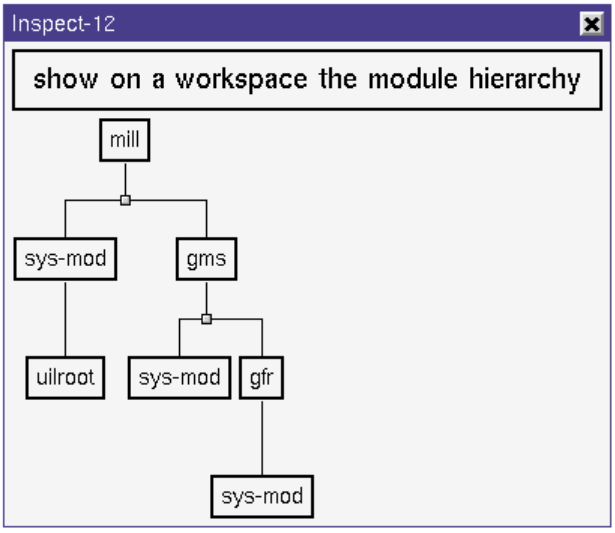


When displaying the class hierarchy of a class that uses multiple inheritance, the graphical representation shows the specified class as a subclass of each superior class, as shown in this example where the class car-boat has two direct-superior classes: car and boat.



Showing the Module Hierarchy

You can show the module hierarchy for the entire knowledge base, or for a particular module. G2 displays a temporary workspace that shows a graphical representation of the module hierarchy as shown in the following example:



If more than one module in the hierarchy directly requires the same module, the module appears multiple times in the hierarchy. However, if the directly required module also has directly required modules, the submodules appear only once. An example is the sys-mod module shown in the previous example.

Showing Procedure Caller and Calling Hierarchies

You can show the hierarchy of all procedures that call, or are called by, a specified procedure. The invocation can be either a call or a start. The grammar is:

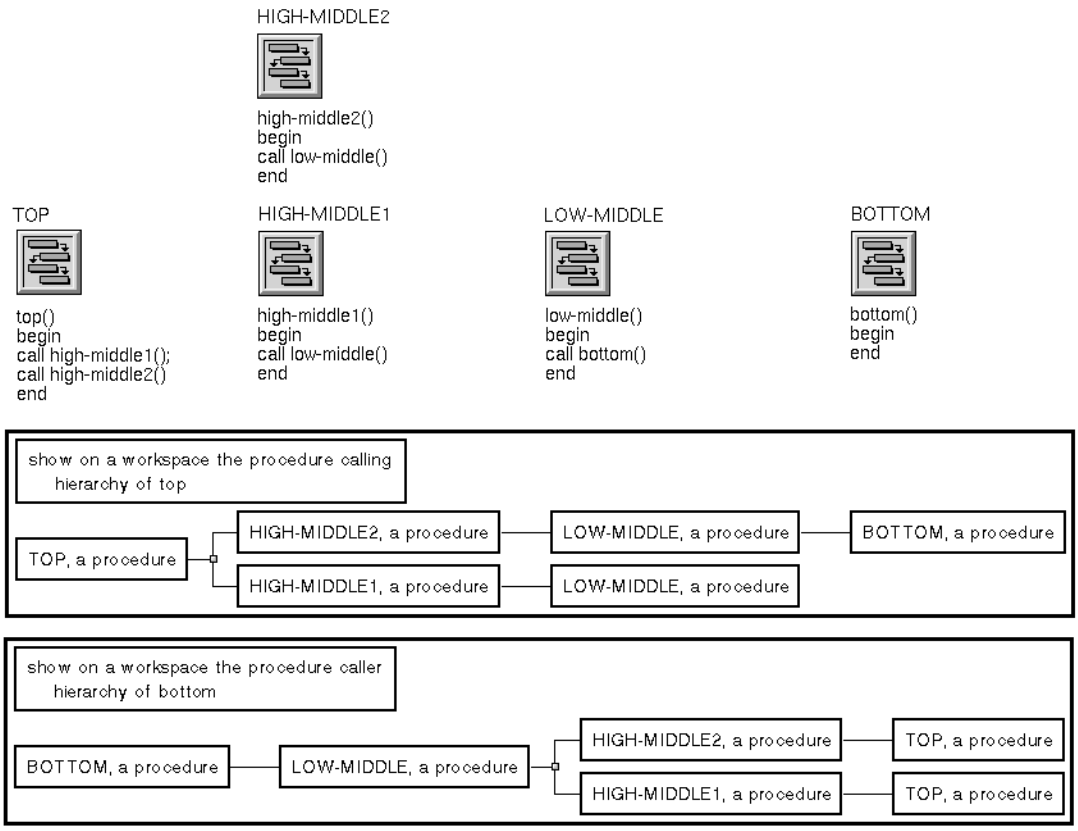
```
show on a workspace
  the procedure {caller | calling} hierarchy of procedure-name
```

The *procedure-name* can be a method name, and the method name can be class-qualified. If an unqualified method name is given, and more than one method exists with that name, Inspect displays information for all of the methods.

Inspect can show only invocation hierarchies that are statically defined in the compiled code. Invocations that are computed at run time, such as a call to a procedure passed as an argument to the calling procedure, do not appear.

The system procedures g2-get-procedure-caller-hierarchy and g2-get-procedure-calling-hierarchy give you programmatic access to these hierarchies. These procedure and other programmatic inspect procedures are documented in the *G2 System Procedures Reference Manual*.

Here is an example of the caller and calling hierarchies:



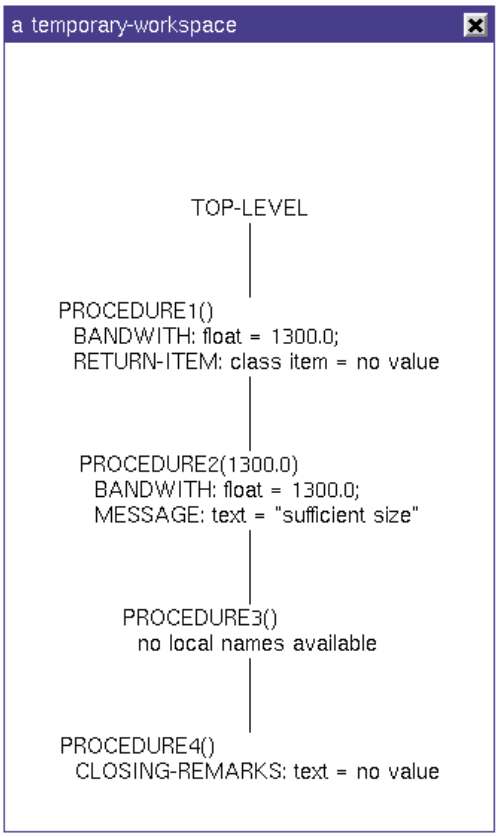
Showing the Procedure Invocation Hierarchy

You can show the procedure invocation heirarchy that is on the current runtime stack. The hierarchy is displayed as a tree with the procedure that was invoked first at the top root with subsequent invocations cascading downwards. The values displayed are the arguments to the procedure, the local variables, and the return arguments.

The syntax is:

```
show on a workspace the procedure invocation heirarchy
```

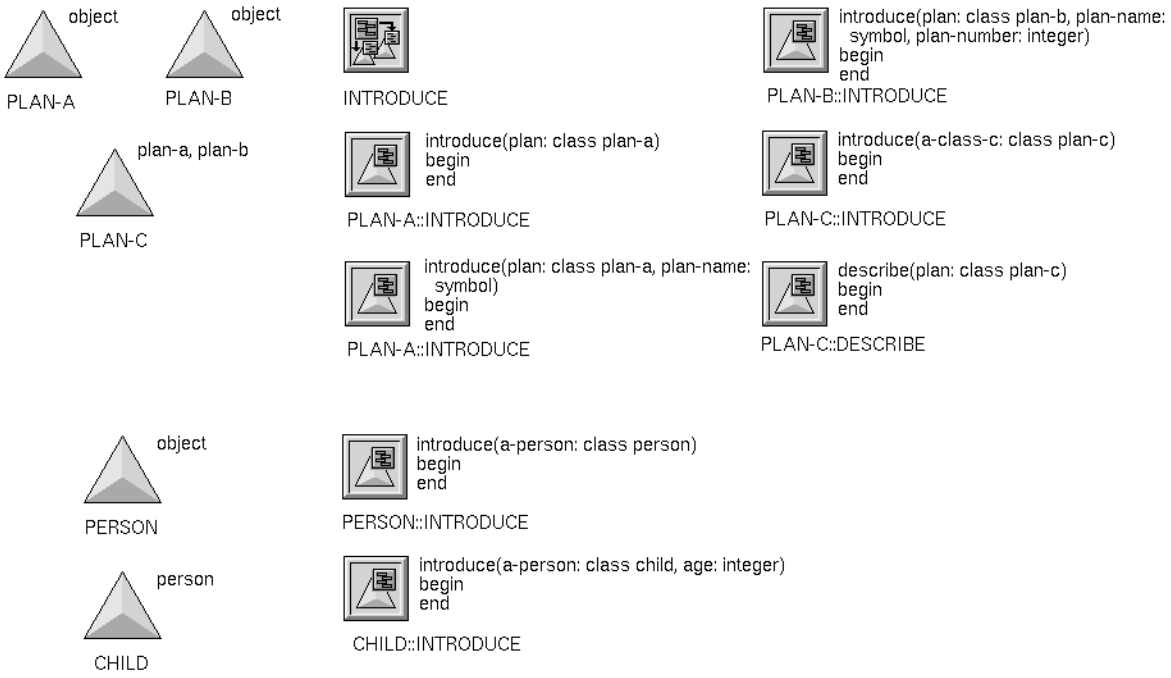
Here is an example invocation hierarchy as displayed on an Inspect temporary workspace:



The system procedure g2-get-procedure-invocation-hierarchy gives you programmatic access to the invocation heirarchy. This procedure and other programmatic inspect procedures are documented in the *G2 System Procedures Reference Manual*.

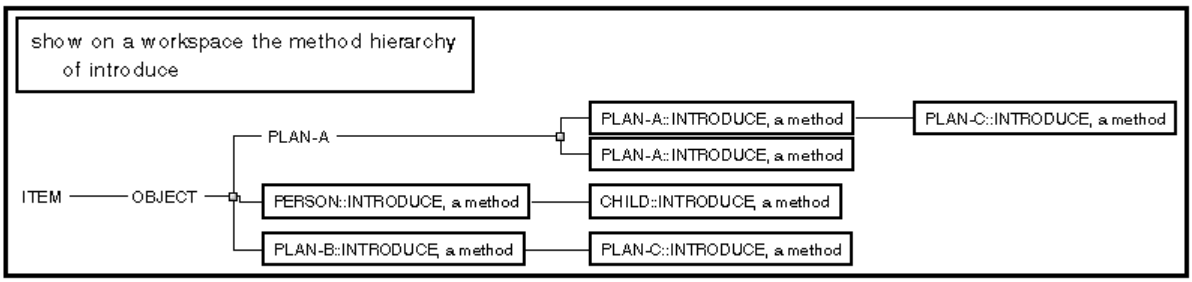
Showing Method Definition Hierarchies

The three method hierarchy examples shown in this section are based on the two class hierarchies and their related methods and method declaration shown in the following figure. For example purposes, there is no method declaration defined for the describe method.



You can show the hierarchy for all classes that define methods with a particular name. G2 displays a temporary workspace that shows a tree representation of the method hierarchies, using short representations of the methods defined for each class.

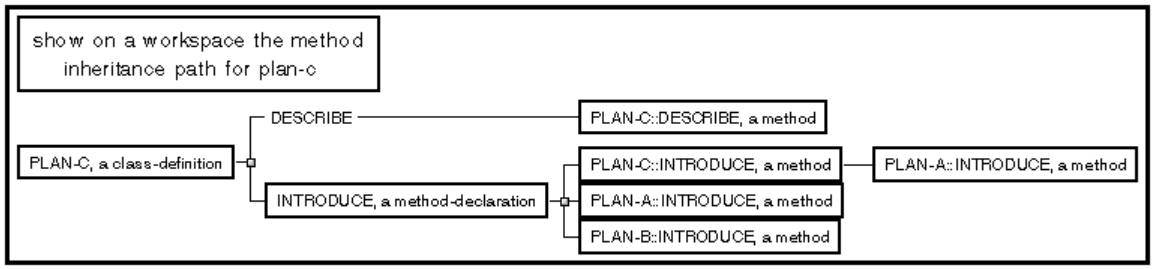
The following example shows the result of showing the method hierarchies of all the classes that define the introduce method in the KB:



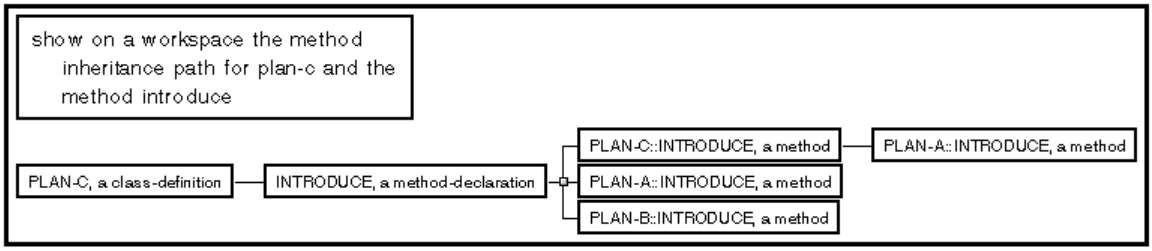
You can also show the method inheritance path for a particular class, which is useful for determining all the methods defined specifically for the class and the

methods it inherits from its superior classes. This Inspect command is also useful for determining what method G2 calls methods when evaluating the call next method statement.

The following example shows the method inheritance path for plan-c. The methods are arranged from left-to-right, with the most specific methods on the left and the least specific methods on the right. There is no method declaration for the describe method so G2 displays the method name instead of a method-declaration short representation.



Finally, you can show the method inheritance path for a particular class and a particular method name, as this example shows:



Writing Items to a File

You can use the Inspect facility to write a report to a file by using the write to file command. Write to the file allows you to inspect most of the same things that you can inspect with show on a workspace, except that G2 writes the results to a file instead of showing them on a temporary workspace.

The generated file contains a text heading indicating what knowledge base you inspected and the time when the report was generated. The contents of the file depend on what you are inspecting, as the following sections discuss.

The filename you specify can be a symbol if it just names a file. If it names a complete pathname, it must be a string. If you do not specify a file extension, G2 adds the suffix .qp. If you do not specify a directory, it writes to the current directory that is specified by the Load KB command.

Syntax

write to the file *filename*
{ *item* |
  the *class-name* named *item-name*} |
  every *class-name* [*filter*] |
  the class hierarchy [of *class-name*] }

Writing Items

You can write the contents of any item, class of items, or item of a particular class with a particular name to a file. You can use any expression to identify the item or class.

The file contains the file header, the name of the item and its class, where relevant, the name, notes, and item-configuration attributes for the item, and any user-defined attributes or specifications for the item.

For example, you might want to write a file of a particular item or a class of items:

```
write to the file "user/log/pump-1" pump-1

write to the file "user/log/pump-classes" every pump
```

The output file would be an ASCII text file that might look like this:

```
** Gensym G2 Knowledge Base Inspection Output
** From KB:            pump-text.kb
** File:      /home/abc/pump-test.text
** Written at: 26 Jul 9912:39:14 p.m.

** Command:
write to the file "/home/nrs/pump-test.text" every pump

** Results follow this line:

FUEL-PUMP-1, a fuel-pump
Notes             OK
Item configuration  none
Names             FUEL-PUMP-1
Pressure                    8

WATER-PUMP-1, a water-pump
Notes             OK
Item configuration  none
Names             WATER-PUMP-1
Pressure                    5
```

You might also issue the following command to write a file of all messages containing a certain word:

```
write to the file "/user/log/pump-status" every message
     containing the word pump
```

Writing a Class Hierarchy

You can write a class hierarchy to a file. The contents of the file includes the file header, a list of classes, appropriately indented to indicate the hierarchy, and the number of instances of each class.

For example, you could write a file that showed the class hierarchy of a particular class and all its instances:

```
write to the file "user/log/pump-classes" the class hierarchy of pump
```

Locating Items in Your KB

You can use the go to command to locate particular items in a KB. Issuing this command causes G2 to display the specified item at full scale in the center of the workspace with your cursor on the item.

The syntax is:

go to *symbol*

For example, the following command moves the cursor directly to valve-5:

```
go to valve-5
```

Tip You can accomplish the same thing by using the show on a workspace command, and then by using the go to original command on the item.

Note If the item is already on the screen when you go to that item, G2 moves the workspace so that the item is in the center of the screen.

Displaying Item Tables

You can use the Inspect facility to display attribute tables for an item directly on a temporary workspace. You can display all attributes of an item or particular attributes, and you can display the items row by row, or column by column.

The cells of the table serve as items, with which you can interact. For example, you can display the actual table for the item, go to the original item, and describe the item. You can also edit attribute values of the item directly in the table.

Note Displaying tables for large numbers of items can require significant amounts of memory, because each table cell requires memory to display.

Syntax

display a table
[columnwise] of [the *attribute-name* [, *attribute-name*]... of]
{ *item* |
 the *class-name* named *item-name* |
 every *class-name* [*filter*] }

Determining How to Display the Table

You can display a table of attributes where each item is displayed in a row or in a column. The table includes all user-defined attributes of the item, as well as the notes and names attributes.

If there are more items that are visible on the workspace, you will have to move the workspace to see all the cells in the table. Here are two examples:

Table header	Notes	Names	Pressure
VALVE-2, a valve	OK	VALVE-2	1000
VALVE-1, a valve	OK	VALVE-1	1500

Display a table of every valve

Table header	VALVE-2, a valve	VALVE-1, a valve
Notes	OK	OK
Names	VALVE-2	VALVE-1
Pressure	1000	1500

Display a table columnwise of every valve

Specifying Which Attributes to Display in the Table

You can specify particular attributes to display in the display a table command.

The following command displays just the input-signal and output-signal attributes of every Digital Component object on the schematic workspace:

```
display a table of the input-signal and output-signal of every
  digital-component found on the workspace schematic
```

When specifying more than two items in the list of attributes to display, the items should be separated by commas, and the last two items should be separated by the word and.

The following command displays the name of every digital component, in addition to the input signal and output signal:

```
display a table of the names, input-signal, and output-signal of every
  digital-component found on the workspace schematic
```

Interacting with the Table

You can interact with the cells in the table as if they were items. For example, you can display the table for the item, describe the item, go to the original item, as well as edit attribute values of the item directly in the table.

To interact with the item:

1. Click on a cell that represents the name of the item or its notes to display a menu for the item representation.
2. To display the table for the item, select table.
3. To place the cursor on the original item, select go to original.
4. To use the Describe facility on the item, select describe.

To edit the attributes of the item:

➔ Click on a cell that represents the value of an attribute, and enter a new value.

To hide the table:

➔ Click on any cell that represents the name of the item or its notes, or click on any header cell, and select hide table.

Replacing Text in Items

You can use the Inspect facility to change every occurrence of a piece of text within all or part of a knowledge base. You can replace numerous types of text, for example:

- Attribute values of items.
- Symbols and strings in rules and procedures.
- Text strings in messages, labels, free text, and so on.

Note You cannot replace text when using a deployment license.

Syntax

```
replace [ the { word | symbol } ] {string | symbol}
with {string | symbol}
in  {item |
      the class-name named item-name |
      every class-name [ filter ] }
```

Replacing Text

In general, the format of the replace command is:

replace *text-to-find* with *text-to-replace* in *item*

Some examples are:

```
replace person-class with people-class in every item
```

```
replace "connected at the input-port-1" with "connected at the input-port-a"
  in every rule
```

When you follow the replace command with the word or the symbol to specify the type of text to replace, G2 will not replace subtext. It will only replace text preceded and followed by white space.

For example, given this text:

There are nine classes that are subclasses of alligator.

if you use this command to replace text:

```
replace classes with species in text203
```

the text becomes:

There are nine species that are subspecies of alligator.

If you replace the original text with this command:

```
replace the word classes with species in text203
```

the text becomes:

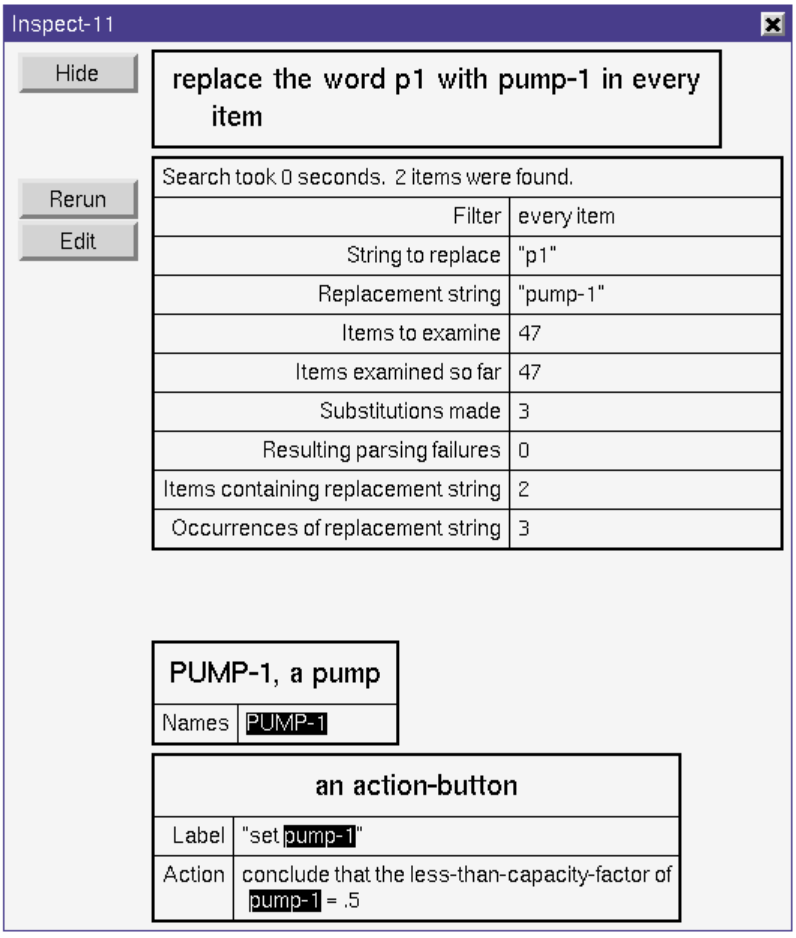
There are nine species that are subclasses of alligator.

As with show on a workspace, you can replace text in specific items, named items of a specific class, or classes of items that meet a particular criteria. For example:

```
replace "pump-1" with "p-1" in every procedure found on the workspace pump-ws
```

The Inspect workspace reports on various information as the replacement is occurring, and it displays a short representation of the items in which replacements have occurred, highlighting the replaced text. If there are more replacements than can fit on one workspace, you will need to move the workspace to see all items.

This example replaces a word in every item:



Note G2 ignores case when searching for and replacing text.

Replacing Text That is Not Grammatically Correct

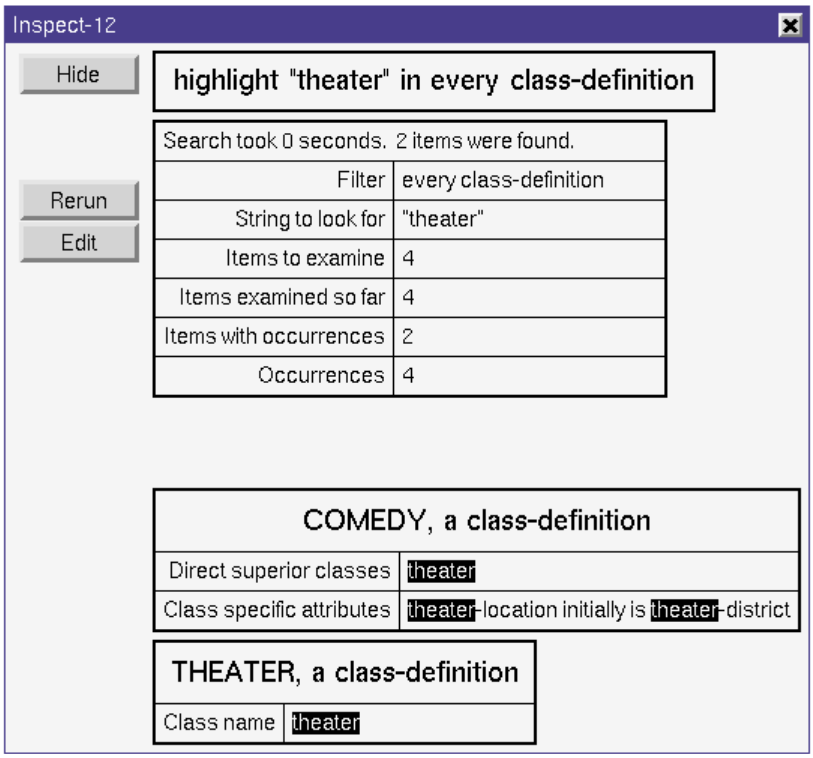
G2 does not replace a string if the operation is grammatically incorrect. When G2 attempts a replacement that results in incorrect syntax, it displays a temporary edit workspace, which shows the item that it could not change. You can edit the item or hide the workspace.

Highlighting Text

If you do not want the Inspect facility to change text immediately, you can use the highlight command to find every occurrence of a piece of text, and display it on a temporary workspace. Once it is found, you can edit the text manually, if desired. G2 highlights text only in editable attributes.

The highlight command has the same syntax as the replace command. Using the word and the symbol syntax highlights text preceded and followed by whitespace. sIt does not highlight subtext. For examples, see Replacing Text in Items.

For example:



Checking for Consistent Modularization

You can use the Inspect facility to check that your KB is consistently modularized. You do this by using the following command:

```
check for consistent modularization
```

Executing this command causes G2 to validate that the current KB's modules conform to G2's rules for consistent modularization. For information about these rules, see Rules for Consistent Modularization.

Recompiling Items

You can use the Inspect facility to recompile individual items or classes of items. You recompile items when upgrading to new versions of G2 or after declaring a KB to be stable.

For example, when upgrading to a new version of G2, you can recompile every item like this to make use of compiler optimization:

```
recompile every item
```

Syntax

```
recompile
{ item |
  the class-name named item-name |
  every class [ filter ] |
```

Filtering Classes of Items

When inspecting classes of items by using every in the command, you can restrict the search to include only those items that meet certain criteria. This is a powerful way of identifying particular classes of items to inspect.

This filter applies when you inspect classes of items by using the every syntax with the following Inspect commands:

- show on a workspace
- write to the file
- display a table
- replace
- highlight

You can provide any number of filters, connected by and and or, and optionally grouped in parentheses for clarity. For example, to see all the pipes that fall within an acceptable diameter range, enter the following:

```
show on a workspace every pipe where diameter >= 5 inches and
  where diameter <= 18 inches
```

Note When combining filters in an Inspect command, it is up to you to insure that the filter expressions are not mutually exclusive.

Filtering Items Based on a Truth-Value Expression

You can filter items based on a condition that you specify as a truth-value expression. Use the such that phrase as the filter. The syntax for the filter expression is:

such that *truth-value-expression*

For example, the following command finds every variable with a current value.

```
show on a workspace every g2-variable
  such that the g2-variable has a current value
```

For information about truth-value expressions, see Expressions.

Note You cannot use such that to test the value of an attribute; use where instead (see Filtering Items Based on the Value of an Attribute).

Filtering Items That Contain Specific Text

You can filter items based on the existence of a particular piece of text by using containing the as the filter. The item can contain a word, a string, or a symbol.

The format for the filter expression is:

containing [the { word | symbol }] *text-expression*

For example, this command finds every statement containing the specified text.

```
show on a workspace every statement containing "tank is overflowing"
```

If you use the word or the symbol in the command, the command only finds text that exactly matches the specified word or symbol; it does not find partial strings. For example, the following command only finds statements containing the word inform; it would not find strings that contained the word information, for example.

```
show on a workspace every statement containing the word inform
```

Filtering Items That Contain Notes

You can find items based on whether or not they contain a value other than OK in the notes attribute. This command allows you to locate all items with unspecified information. For example:

```
show on a workspace every tank with notes
```

Filtering Items Based on the Item Status

Every item has a status that indicates various information about the item:

- The status ok, bad, or incomplete indicates the value of the item's notes attribute.
- The status active and inactive indicates whether the workspace on which the item is located is active or inactive.
- The status enabled and disabled indicates whether the item itself is enabled or disabled.

You can find items based on their status by using the following as the filter:

whose status is *status*

For example:

```
show on a workspace every class-definition whose status is incomplete
```

Filtering Items Based on the Value of an Attribute

You can filter items based on the value of a numeric or symbolic attribute:

- If the attribute value is numeric, you use one of these relational operators to test the value: =, /=, >, <, >=, and <=.
- If the attribute value is symbolic, you use the word is with a truth value (true or false) or any symbol to test the value.

The format for the filter expression is either of these expressions:

where *attribute-name relational-operator numeric-expression*

where *attribute-name* is { *truth-value-expression* | *symbolic-expression* }

Two examples are:

```
show on a workspace every tank where inflow >= 14
```

```
show on a workspace every bin where recently-emptied is true
```

Filtering Items Based on Their Category or Focal Class

Every rule has the attributes categories, focal-classes, and focal-objects. You can filter rules based on their category, focal class, or focal object by using the following filter expressions:

in the category *symbolic-expression*

which has the focal { class | object } *symbolic-expression*

Two examples are:

```
show on a show on a workspace every rule in the category safety-rules

show on a workspace every rule which has the focal class automobile
```

Filtering Items Based on Their Workspace

You can filter objects based on the workspace on which the object is located by using found on the workspace in the command.

```
show on a workspace every rule found on the workspace
  machine-schematic-rules
```

Filtering Items Based on Their Module

You can filter items based on their assigned module as follows:

- To filter items based on their location in a particular module or a list of modules, use assigned to module in the filter expression.
- To filter items based on their location in any module in a particular module hierarchy, use assigned to the hierarchy of module in the filter expression.

For example:

```
show on a workspace every tracked-vehicle
    assigned to module plant-floor-schematic
```

You can filter items based on a list of modules, by separating the modules names with or, as follows:

```
show on a workspace every tracked-vehicle
    assigned to module plant-floor-module or
    assigned to module vehicle-status-module
```

If there are more than two modules in the list, separate the module names with a comma, and separate the last two module names with or.

This example shows how you filter items based on any module in a hierarchy, as follows:

```
show on a workspace every tracked-vehicle
    assigned to the hierarchy of module plant-floor-module
```

Filtering Items That Do Not Meet Specified Criteria

You can use the word not in conjunction with any of the previously discussed filter expressions to restrict the search to those items that do not meet the criteria.

The word not precedes the filter expression.

For example, the following command finds all class definitions whose status is not OK:

```
show on a workspace every class-definition whose status is not ok
```

The following command finds every variable that does not have a current value:

```
show on a workspace every G2-variable
  not such that the G2-variable has a current value
```

Version Control

You can use the following Inspect command to show the change log for items, tag change log entries, revert change log entries, delete change log entries, and enable/disable change logging for items.

show on a workspace the change log entry of the *attribute* of *item*
{as of *timestamp*} | {with revision *num*} | {with tag *tag*}

Shows the list of change log entries for the given *attribute* of *item*, as of *timestamp*, with revision *num*, or tagged with *tag*.

show on a workspace the differences between
the change log entry of the *attribute* of *item*
{as of *timestamp*} | {with revision *num*} | {with tag *tag*}
and the change log entry of the *attribute* of *item*
{as of *timestamp*} | {with revision *num*} | {with tag *tag*}

Shows the differences between two change log entries for the *attribute* of *item*, as of *timestamp*, with revision *num*, or tagged with *tag*. This command uses the external "diff" program specified by g2-set-external-diff-specification.

use version control to tag the change log entry of every logged attribute of every item in module *module-name* [as of *timestamp*] using tag *tag*

Tags the change log entry of every item for which change logging is enabled in *module-name*, using *tag*, optionally as of *timestamp*.

use version control to tag the change log entry of the *attribute* of *item*
[as of *timestamp* | with revision *num*] using tag *tag*

Tags the change log entry of the *attribute* of *item*, using *tag*, optionally as of *timestamp* or with revision *num*.

use version control to revert the text of every logged attribute of every item in module *module-name* to the change log entry
{as of *timestamp*} | {using tag *tag*}

Reverts the text of every item for which change logging is enabled in *module-name* to the change log entry as of *timestamp* or tagged with *tag*.

use version control to revert the text of the *attribute* of *item* to the change log entry {as of *timestamp*} | {with revision *num*} | {with tag *tag*}

Reverts the text of the *attribute* of *item* to the change log entry as of *timestamp*, with revision *num*, or tagged with *tag*.

use version control to delete the change log entry of the *attribute* of *item* {as of *timestamp*} | {with revision *num*} | {with tag *tag*}

Deletes the change log entry for the *attribute* of *item*, as of *timestamp*, with revision *num*, or tagged with *tag*.

use version control to enable change logging on *item*

Enables change logging on *item*. If change logging is already disabled on the module, this command has no effect.

use version control to disable change logging on *item*

Disables change logging on *item*. If change logging is already disabled on the module, this command has no effect.

Inspect Command History (Enterprise only)

This is a Enterprise only feature. Now you can use the following Inspect command to show the inspect command history. Notice that currently G2 Server only keep 50 history commands, which is shared between server and all clients.

show on a workspace the inspect command history

Shows the list of inspect command history. By double-clicking on historical commands, it's possible to modify historical inspect commands and re-execute, or just re-execute them.

48

Natural Language Facilities

Describes the facilities for using non-English languages in a KB.

Introduction

G2 supports natural languages through these facilities:

- Fonts for working with various European and non-European languages.
- Language-definition items, which provide a means to localize G2 menu choices and certain G2 facilities.
- The `language.kl` knowledge library (KL), which provides pre-defined menu translations in several European languages.
- Japanese language facilities.
- Korean language facilities.

- Chinese language facilities.
- Russian language facilities.

On Windows, when connecting Telewindows in standard mode, your Windows system must have fonts installed to display all natural language text. For example, to properly display Japanese text in menus, your system must have Japanese-capable fonts. Fonts for Japanese and many other languages are included with the Microsoft Windows 2000 and Windows XP operating system, but may need to be installed in some cases. Refer to Windows documentation and support for further information.

Using G2 Fonts

G2 and Telewindows have built-in fonts for Roman alphabet languages. To display Japanese, Korean, and Chinese characters, G2 requires other fonts. The fonts for these languages are available in the following subdirectory of your G2 product directory, by default:

`g2\fonts` (Windows)

`g2/fonts` (UNIX)

Fonts that reside in this directory are loaded automatically for customers that are authorized for Japanese, Korean, or Chinese fonts.

You can store font files in a directory other than the default, and then specify that location when starting G2 by using:

- The `-fonts` command-line option:

  `g2 -fonts` *\directory\my-fonts*
  `g2 -fonts` */directory/my-fonts/*

  where *\directory\my-fonts* (Windows) or *\directory\my-fonts* (UNIX) is the directory location of the fonts. The command-line option requires the slash character following the directory name.

- An environment variable or a logical to specify the location. For a description of using environment variables and logicals, see Using Environment Variables.

Once an environment variable or a logical exists, you can override that location with the `-fonts` command-line option as described previously.

Using the Natural Language Facilities

To use the natural language facilities that G2 provides, you can:

- Set the current language to be any available language.
- Use more than one language in the same KB.
- Extend the set of available localizations by creating language translation definitions.
- Localize one or more properties of G2 facilities.

G2 provides system-defined localizations for some languages. To use these capabilities, you can:

- Load the `language.kl`, described in Using Language Translations for Localization.
- Load `japanese.kl`, described in Using the Japanese Language Facilities.
- Load `korean.kl`, described in Using the Korean Language Facilities.
- Use Cyrillic characters, described in Using the Russian Language Facilities.

Setting the Current Language

The current language is part of a KB's knowledge and is set through the current-language attribute of the Language Parameters system table. The default is english.

To set the current language:

➔ Enter a language other than english in the current-language attribute of the Language Parameters system table, or conclude a new value programmatically.

If language translations for the new language exist in the KB, localized menus and facilities are displayed.

The current language does *not* affect existing symbol and text values entered in supported languages. For example, if you set the current language to russian, and create messages in cyrillic, or name items using cyrillic characters, changing the current language to japanese does not alter the cyrillic messages and item names that you entered.

For system-defined localizations, the effect of changing the current language in a KB depends on:

- The level of support G2 provides.
- The available language KB being loaded.

Changing the current language to one of these languages immediately localizes the editor and Inspect buttons:

- japanese
- korean
- russian

Changing the current language does not change the system-defined menu choices unless the corresponding KB is loaded. For example, changing the current-language attribute to korean immediately localizes the facility buttons, but does not translate the menu choices until you load the `korean.kl` file.

Loading the appropriate KB file localizes the G2 menu choices as follows:

Language	KB to Load	Menu Localizations
japanese	`japanese.kl`	Yes
korean	`korean.kl`	Yes
chinese	Not available	No
russian	Not available	No
french	`language.kl`	No
german	`language.kl`	Yes
dutch	`language.kl`	Yes
italian	`language.kl`	Yes
spanish	`language.kl`	Yes
swedish	`language.kl`	Yes

User-defined localizations can be added through the use of the language translation facilities as described in Localizing Menu Choices and G2 Facilities.

Two command-line options can affect the current language:

- `-default-language`
- `-language`

Setting a Default Language for a G2 Session

The `-default-language` command-line option and its corresponding environment variable, `G2_DEFAULT_LANGUAGE`, is only for G2. You can set a default language only by starting G2 with the `-default-language` command-line option.

Note If you start G2 with the `-default-language` command when an environment variable exists for a different language, the command-line option overrides the environment variable.

Once you start G2 with a default language, that language persists for the entire G2 session with any KB whose current language is english. Loading a KB with a different current language or changing the system table's current-language attribute overrides the default language.

The purpose of the default language command-line option is to set a common language for the entire G2 session, which can be overridden as necessary. For example, if multiple developers accessing a KB through Telewindows are French, you could start G2 with the default-language francais.

If one Telewindows user wanted to use the German menu options, that user could change the language to German in his or her window in the login dialog. Changing the language of a Telewindows connection is described in Supporting a Window-Specific Language.

Setting a Language for the Current Window

The `-language` command-line option, applicable to both G2 and Telewindows, affects only the current window, which is the local window for G2. It does not change G2's default language or the KB's current language.

One way to use this option is in conjunction with the default language option. You could start G2 with a default language. Any Telewindows user who wanted an alternative language could then connect to G2, using the `-language` option with the language they chose.

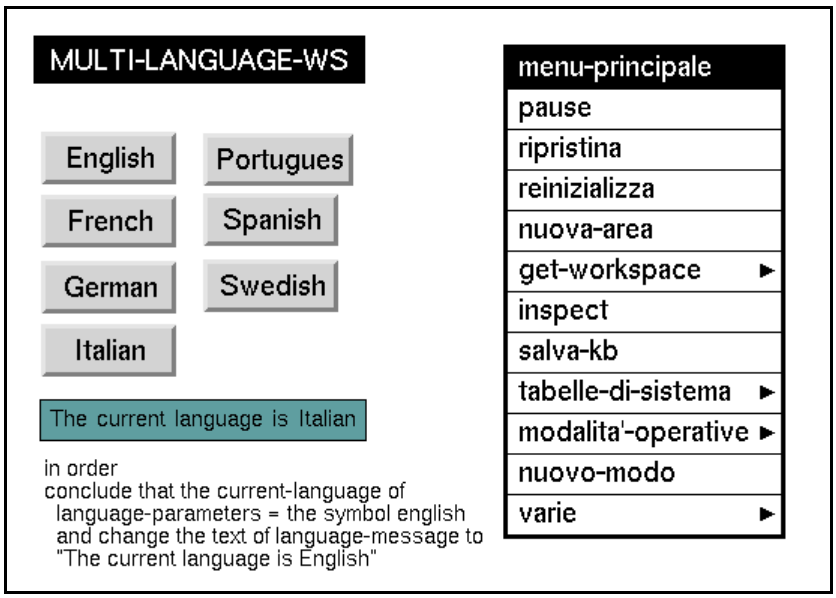
Connecting to G2 this way would alleviate the need for a Telewindows user to change the current language.

Supporting Multiple Languages in a KB

If your KB requires multiple language support, you can provide it by changing the current-language attribute each time you need to enter text in that language.

As an example, the next diagram shows one way to provide access to several languages programmatically through the use of action buttons. Each of the buttons uses a conclude action to change the value of the current-language attribute to a different language. The text of the Italian action button is shown,

along with the sample Main Menu, demonstrating how a menu appears when the current language is Italiano.



Localizing Menu Choices and G2 Facilities

You can localize menu choices and G2 facilities by using a language-translation item.

For menu choices, a language translation lets you substitute one symbolic name for another. For G2 facilities, you can also localize one or more properties of:

- Text Editor buttons.
- Icon Editor buttons.
- All elements of the login dialog, including buttons, messages, and the text of its attributes.

Creating language translation definitions affects whatever choices you include in the definitions. Item names, user-defined classes, and class-specific attributes that include special characters, or characters of another alphabet such as Japanese and Korean, appear on workspaces and in class and attribute lists as you enter them.

Using Language Translations for Localization

You can use one or more language translation definitions to localize system-defined and user menu choices and facilities.

To create a language translation definition:

1 Select KB Workspace > New Definition > language-translation.

G2 invokes the Text Editor immediately for you to enter the translation.

2 Enter the localizations you require for the language you are using.

The language translation definition grammar is:

in *language* [, *context*] : *symbol-to-localize* = *localized-symbol-or-text*

Definition Element	Description
language	The symbolic name for the language. For example, in `language.kl`, these language names exist: • francais • german • dutch • italiano • spanish • swedish
context	See Specifying a Context.
symbol-to-localize	The menu choice or facility to localize.
localized-symbol-or-text	The localized symbol or text value of what should appear for the *symbol-to-localize* argument.

For example, tabella is the localization for table when Italiano is the current language. The language translation defined for that choice is:

```
in italiano : table = tabella
```

The same localization could be expressed as:

```
in italiano : table = "tabella"
```

Note There is currently no restriction for the *language* symbol, and you can enter virtually any valid symbol of your choice. The translations you enter will be in effect any time the current language uses that symbol. Future G2 releases may restrict this element to a predefined set of accepted language symbols.

Specifying a Context

You localize G2 facilities by specifying a context for the G2 element you wish to localize into *language*. A context appears between the specification of the language and the set of symbols and translations. The notation for a context varies depending on what you are specifying. You can:

- Specify button labels of G2 facilities.
- Localize each button, label, message, and attribute name in the G2 Login dialog.

The *context* can be expressed as:

as {a | an} {attribute of {a | an} | *facility-element* in the} *g2-facility*:
symbol-to-localize = *localized-symbol-or-text*

Context Element	Description
facility-element	The button or menu choice to localize. For example, you could specify any of the Icon Editor buttons when specifying *g2-facility* as Icon Editor.
g2-facility	The G2 facility whose buttons, attributes, or other elements you are localizing.
symbol-to-localize	The menu choice or facility to localize.
localized-symbol-or-text	The localized symbol or text value of what should appear for the *symbol-to-localize* argument.

Using a context in this way permits you to use different words when, for example, a symbol is the name of a menu choice rather than the name of an attribute, or when the symbol names a button in an editor or the login dialog.

Only certain combinations of attributes or *facility-elements* and *g2-facility* specifications make sense. Combinations are checked when you close the edit. For example, for the Text Editor, you can specify a button label as a valid *facility-element*, but not any attributes.

Localizing the Text and Icon Editor Buttons

To specify the label on a button in a G2 facility:

➔ as a button-label in the {text-editor | icon-editor}

For example:

in my-world, as a button label in the text-editor: end = Complete

in my-world, as a button label in the icon-editor: Update = "Finalize"

The buttons available for localizing in the editors are:

Text Editor Buttons	Icon Editor Buttons
Cancel	
End	
Update	
Paste	
Redo	
Undo	
	
	
	
	
	
	

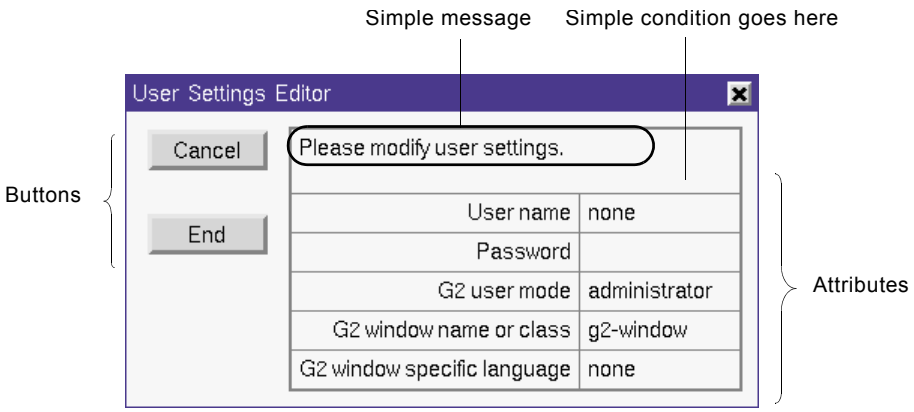
Text Editor Buttons	Icon Editor Buttons
	x1 x2 x3
	Pop
	Done

Localizing the Login Dialog

You can localize each element of the login dialog, including:

- Every button.
- Each attribute.
- The instructional message.
- One or more conditions.

These are the elements of a login dialog:



Specifying the Dialog Buttons

You can localize these buttons:

- Cancel
- End
- Disconnect

Note The buttons on the login dialog do not include the Text Editor buttons Paste, Undo, and Update that appear when a user edits one of the dialog attributes.

To specify a login dialog button:

➔ as a button-label in the g2-login-dialog : *button* = *local-name*

For example:

```
as a button-label in the g2-login-dialog: end = complete
```

Specifying or Localizing the Dialog Message

You can change or localize the message that appears as a directive at the top of the login dialog. The grammar identifies this message as simple-message. By default in G2, this message reads:

Please modify user settings.

To specify a simple message in the dialog:

➔ as a simple-message in the g2-login-dialog : *g2-login-message* = "*local-text*"

The symbol that denotes this particular message on the login dialog is g2-login-prompt-message, so to modify it, enter a definition such as:

```
in my-language, as a simple-message in the g2-login-dialog:
    g2-login-prompt-message = "Please login with your user name:"
```

Note You can enter messages such as this as a text value using quotation marks (") or as a symbol using hyphens to separate words.

Localizing Dialog Attributes

You can localize all attributes of the login dialog, which is an instance of a g2-login item.

To localize the login dialog attributes:

➔ as an attribute of a g2-login : {*g2-login-attribute*} = "*local-text*"

For example:

```
in my-language, as an attribute of a g2-login :
password = "Your password:"
```

Localizing Condition Messages

You can localize the various condition messages that appear in the login dialog when, for example, the user enters an incorrect password. The grammar identifies this message as simple-condition.

To localize the condition messages that can appear:

➔ as a simple-condition in the g2-login-dialog : {*g2-login-condition* } = "*local-text*"

For example:

in my-language, as a simple-condition in the g2-login-dialog :
unknown-user-or-bad-password = "G2 does not recognize your User name or password. Please reenter both and try again. "

Using European Languages

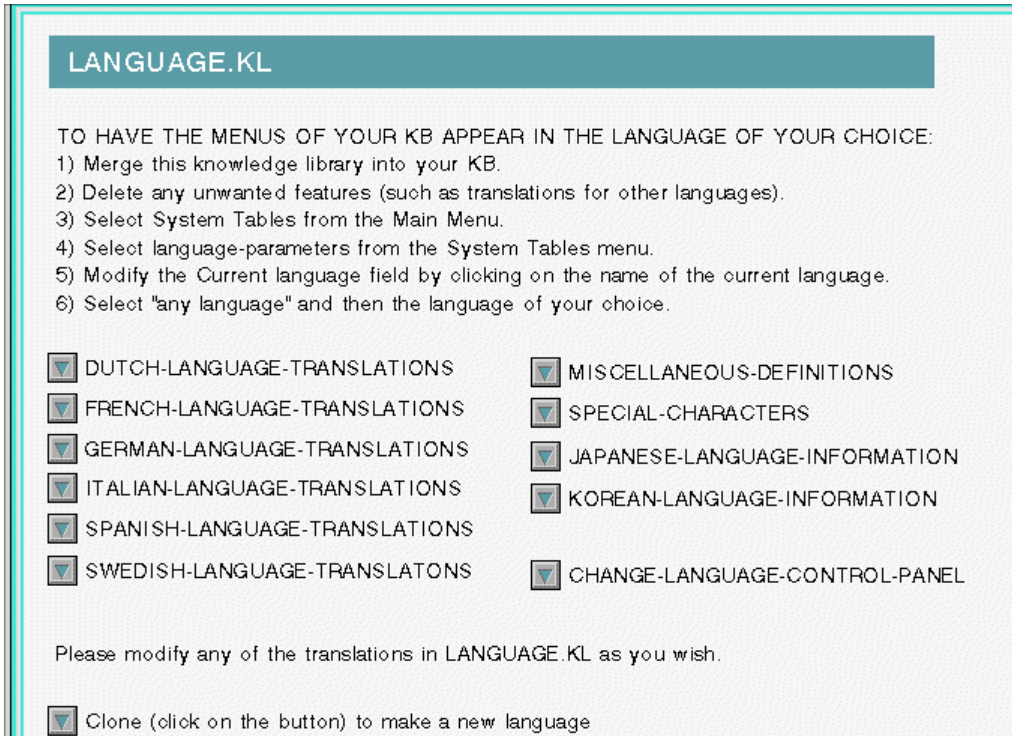
To access the European language translations provided with G2, you need to access the `language.kl` KB.

To merge the KB into your own KB:

1 Select Main Menu > Merge KB.

2 Enter the name of the KL you wish to merge into your KB, `language.kl`.

The `language.kl` file is located in the `g2\kbs\utils` (Windows) or `/g2/kb/utils` (UNIX) directory of your G2 product directory. The workspace that appears after you merge the KL into your own KB is:



Available Translations

Menu translations are available in these languages:

- French
- German
- Dutch
- Italian
- Spanish
- Swedish

Click on any of the go-to-subworkspace buttons to see the specific language translation workspaces.

In addition, the KL provides a set of character inserters for the special characters shown here:

The KL also includes a generic language facilities workspace, containing all of the menu choices available for translation. Use this as a template for translating other menu options of your choice.

The menu choice translations that `language.kl` provides are created by using language definition items (described in Localizing Menu Choices and G2 Facilities).

All language definitions are arranged in alphabetical order upon their workspace, each providing translations for the menu choices, beginning with that particular letter. Alphabetizing the choices is not mandatory, but provides an easy way to locate G2 menu selections.

Once you merge `language.kl` into your KB, you can delete any workspaces containing unused languages.

Using the Japanese, Korean, Chinese, and Thai Language Facilities

G2 provides these special facilities for the Japanese, Korean, Chinese, and Thai languages:

- High-quality fonts for characters specified in Unicode Version 2.0.
- Support for the character-input methods built into Windows platforms.
- The ability to specify your preferred style for the display and printing of Han characters.

Using Windows Character-Input Methods

In G2, you can use the character input methods Windows supplies for platforms that are specifically configured for Chinese, Japanese, or Korean. The supported Windows versions are 2000 Professional, 2003 Professional, and XP Professional.

You can use the character-input methods on a G2 local window running on Windows, and also on Telewindows running on Windows and connecting to a G2 on any platform.

You do not need to set the locale to a specific country in order to enter Asian language characters in G2 and Telewindows.

See your Windows documentation on how to use these character-input methods.

Specifying a Han Character-Style Preference

The Chinese, Japanese, and Korean writing systems share a set of ideographic, historically Chinese, Han characters, but each of these languages has developed its own preference for character style.

The G2 Chinese-Japanese-Korean (CJK) language preference facility determines what style will be used for displaying Han characters in your G2 local window or Telewindows, and for printing workspaces. For Chinese, G2 supports the simplified Chinese characters used mainly in the Peoples Republic of China, rather than the traditional Chinese characters used mainly in Taiwan.

G2 has a default CJK language preference ranking for Han characters which is: Japanese first, Korean second, and Chinese third. This ranking corresponds strictly to the order in which support has been added to G2 for these three languages, and is preserved for the compatibility of existing applications.

You can alter this ranking by specifying your CJK language preference:

- Specifying Japanese leaves the default ordering.
- Specifying Korean moves Korean to the front of the ordering.
- Specifying Chinese moves Chinese to the front of the ordering.

If a character is not available in the first-ranking language font, G2 will use the second-ranking font; or, if necessary, the third-ranking font.

To specify your CJK language preference:

➔ Use this command-line option when starting your G2 or Telewindows process:

```
-cjk-language {chinese | japanese | korean}
```

If you do not include the `-cjk-language` command-line option, G2 determines your CJK language preference by evaluating the following items of information and choosing the first one that supplies a preference:

1. You specify either Chinese, Japanese, or Korean for the g2-window-specific-language attribute of the login dialog.
2. You specify either Chinese, Japanese, or Korean in the `-language` command-line option when starting your G2 or Telewindows process:

   ```
   -language {chinese | japanese | korean}
   ```
3. Your G2 or Telewindows is authorized for Japanese.
4. Your G2 or Telewindows is authorized for Korean.
5. Your G2 or Telewindows is authorized for Chinese.

The following illustration demonstrates a few stylistic differences between Han characters in different CJK language-preference fonts. The characters in each column have the same unicode designation, but each row represents a different CJK preference, as indicated in the left-hand column.

From left to right, an English translation of the characters is: dark, mind, institution, world, shadow, well.

Note When you are printing workspaces from Telewindows, the CJK language preference of G2, not Telewindows, determines character style.

Using the Japanese Language Facilities

As a part of G2, the Japanese language facilities support all system-defined G2 features, plus the ability to develop, run, and view applications in Japanese. KBs developed in Japanese are platform independent and can run without modification when the current language is not Japanese. The Japanese language facilities provide this support for developing Japanese KBs:

Feature	Description
Text transliteration and conversion	Converts Romaji input to Hiragana or Katakana automatically.
Customized menus and other facilities	G2 facilities, such as the editor, have localized option buttons. Loading `japanese.kl` provides localized versions of the G2 system-defined menu options. Developers can localize any facility element, as described in Using Language Translations for Localization.
Knowledge engineering	Can assign Japanese names to all the items you define, including objects, workspaces, rules, procedures, and methods.
Character input	Support for the character-input methods built into Windows platforms configured for Japanese.
Text editing	Provides the system-defined G2 editing features, as well as special modes added to support Hiragana, Katakana, and Kanji characters.

Support for the Japanese language within G2 consists of the Kanji Front-End Processor (KFEP), which handles all Japanese keyboard input, and the knowledge library, `japanese.kl`, which includes translations of all G2 menu choices. Japanese outline fonts for screen display are available if you have appropriate authorization. Outline fonts are currently not supported for printing.

Accessing Japanese Menus

The Japanese menus are in G2's Japanese KL.

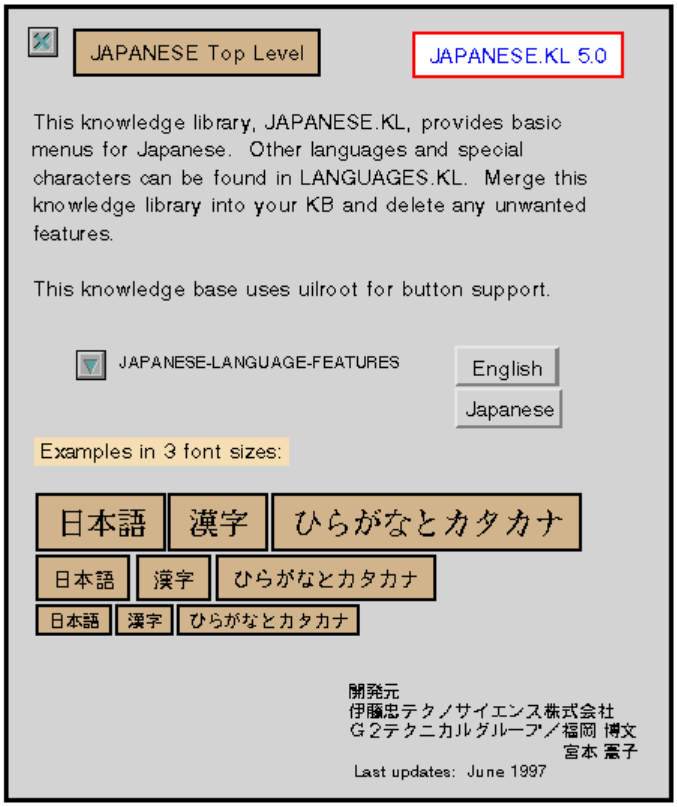
To merge `japanese.kl` into your KB:

1 Select Main Menu > Merge KB.

2 Enter the name of the KL you wish to merge into your KB, `japanese.kl`.

The `japanese.kl` file is located in the `\g2\kbs\utils` (Windows) or `/g2/kbs/utils` (UNIX) directory of your G2 product directory. Check with your system administrator if you are unable to locate the sample KBs that ship with G2.

After merging the KL into your own KB, this workspace is displayed:



The current user mode is developer. Starting the KB activates all go-to-subworkspace and other buttons.

To start the KB:

➔ Select Main Menu > Start.

The G2 menu translations and the translated Text Editor buttons do not appear unless the current language is Japanese.

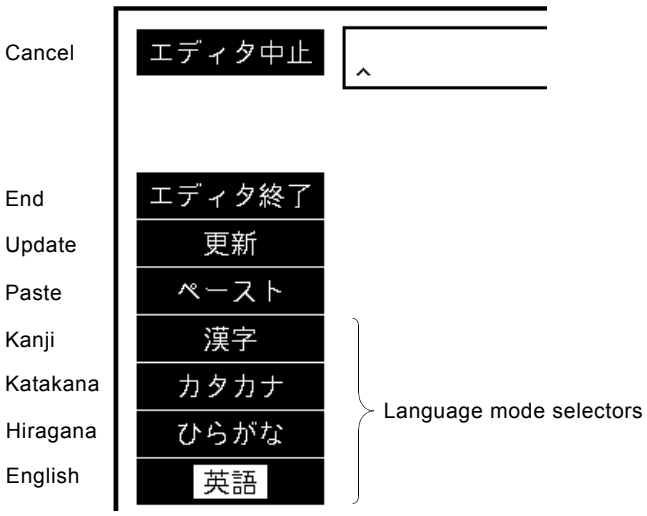
To change the current language to Japanese:

1 Select Main Menu > System Tables > Language Parameters.

2 Edit the current-language attribute and select japanese.

Entering Japanese Text

When japanese is the current language in your KB, the editor includes four language mode selectors to help you entering Japanese text, as shown in the following figure:



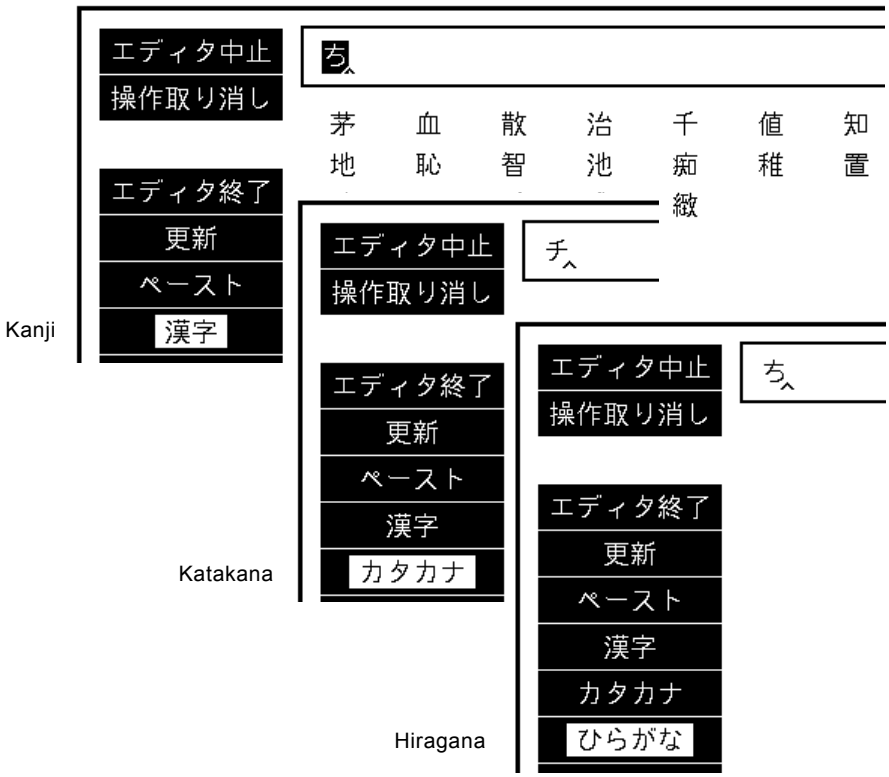
The language modes are:

Mode	Description	Character Interpretation
Kanji	The Japanese writing system using characters borrowed from Chinese.	Hiragana, and displays Kanji choices whenever a convertible sequence occurs.
Katakana	The form of Japanese writing used for scientific and technical terms, official documents, and words adopted from other languages.	Romaji, and transliterates the characters to Hiragana.
Hiragana	A Japanese script, the second of two forms of Japanese writing.	Romaji, and transliterates the characters to Hiragana.
English	American English language.	Latin characters. This is the default mode.

Keyboard input is phonetic. The Text Editor accepts several phonetic variations for Japanese characters, including *si* and *shi*; *ti* and *chi*, and *tu* and *tsu*.

The KFEP interprets character input based on the language mode you select, as noted in the previous table. You can choose a mode by clicking an activity button directly with the mouse, or by using Control + k to cycle through the choices.

To illustrate the effect of each language mode, the next diagram shows what happens when you enter the characters chi in Kanji, Katakana, and Hiragana. Notice that, while in Kanji mode, the characters are interpreted as Hiragana, and the editor displays various Kanji choices appropriate for that input.

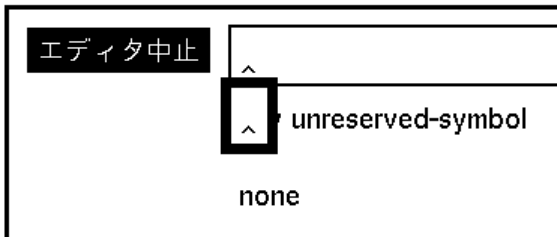


When the current language is japanese, you can input an explicit Kanji ideogram that you cannot obtain through the conversion process by entering any character from the Japanese Industrial Standard (JIS) X 0208-1990 code. G2 supports the entire code set, which consists of four-digit Hexadecimal values. If you enter a JIS code when the current language is not japanese, G2 interprets the value as a Unicode character code.

To enter a JIS code in the Text Editor:

1. Type Alt + i to invoke a secondary text entry box.
2. Enter the four-digit code, as shown next.

The character appears as soon as you enter a valid four-digit code.

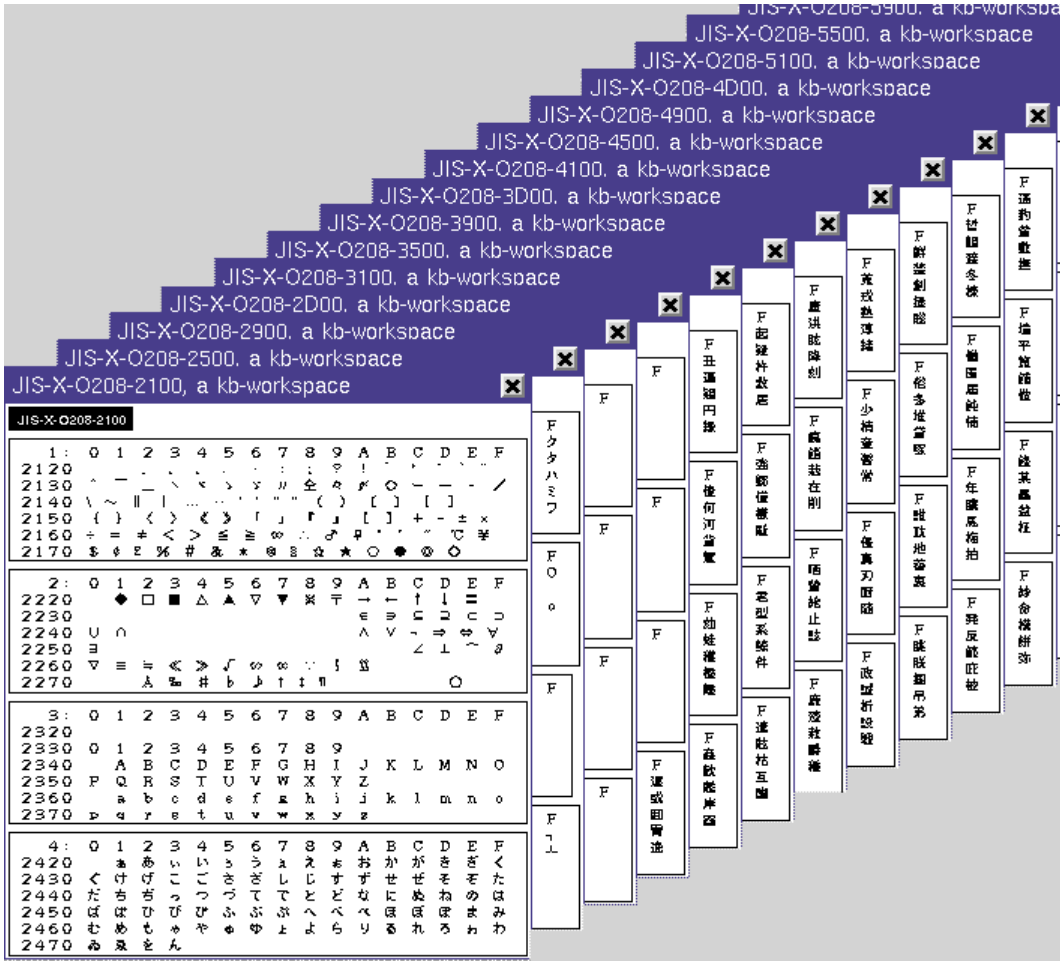


Alternatively, you can merge in the `jiscodes.kl` KL.

To merge `jiscodes.kl` into your KB:

➔ Select Main Menu > Merge KB and enter `jiscodes.kl`.

The `jiscodes.kl` file is located in the `\g2\kbs\utils` (Windows) or `/g2/kbs/utils` (UNIX) directory of the G2 product directory, and displays a series of text inserter workspaces, each with multiple sets of characters:



Entering Text in the Kanji Language Mode

As noted earlier, when Kanji is the language mode and you enter text, the KFEP interprets the characters as Hiragana, displaying Kanji choices for any appropriate conversion options. As a user, you can either convert the Hiragana input to Kanji, by choosing from one of the options listed, or retain the kana form for individual characters.

To help you convert text, the Text Editor displays more activity buttons as you enter text. In addition to the system-defined Text Editor activity buttons (Cancel, Undo, Redo, End, Paste), and the Japanese language mode buttons (Kanji, Katakana, Hiragana, and English), the following activity buttons are also available in Kanji mode:

This activity button...	Translates to...	Which lets you...
変換	Convert	Convert the Hiragana selection to the highlighted Kanji character.
次候補	Next	Highlight the next Kanji character.
前候補	Previous	Highlight the previous Kanji character.
スキップ	Skip (Control + ')	Move the Hiragana selection to the next possible conversion region.
アクセプト	Accept	Accept the highlighted Kanji character to replace the Hiragana selection. If no Kanji character is selected, G2 accepts the first character.
変換範囲拡張	Expand (Control + >)	Expand the conversion region (if possible).
変換範囲縮小	Shrink (Control + <)	Shrink the conversion region (if possible).

Using the Korean Language Facilities

The Korean language facilities support all system-defined G2 features, plus the ability to develop, run, and view applications in Korean (Hangul). KBs developed in Korean are platform independent and can run without modification in other languages. Korean outline fonts are available if you have appropriate authorization. The fonts are identical to the Gulim font supplied with the Korean version of Windows.

Support for the Korean language within G2 consists of the Hangul Front-End Processor (HFEP), which handles all Korean keyboard input; `korean.kl`, which

includes translations of all G2 menu choices; and `kscodes.kl`, which provides the KS C 5601 character set as an online reference.

The Korean language facilities provide the following support for developing KBs in Hangul:

Feature	Description
Customized menus and other facilities	G2 facilities, such as the editor, have localized option buttons. Loading `korean.kl` provides versions of the G2 system-defined menu options with both Hangul and English localizations. Developers can localize any facility element, as described in Using Language Translations for Localization.
Knowledge engineering	Ability to assign Korean names to all the items you define, including objects, workspaces, rules, procedures, and methods.
Character input	Support for the character-input methods built into Windows platforms configured for Korean.
Additional characters	Full support of all characters from the standard KS C 5601 Korean character set.
Standard PostScript compatibility	G2 print functions for Korean KBs fully compatible with any standard PostScript printer.

Accessing Korean Menus

You can access Korean menus and facilities by merging in `korean.kl`.

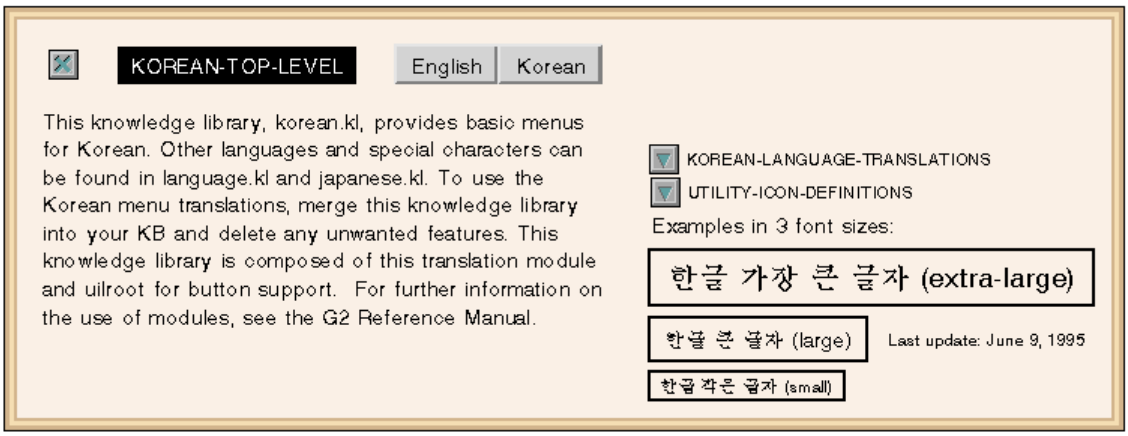
To merge `korean.kl` into your own KB:

1 Select Main Menu > Merge KB.

2 Enter the name of the KL you wish to merge into your KB, `korean.kl`.

The `korean.kl` file is located in the `\g2\kbs\utils` (Windows) or `/g2/kbs/utils` (UNIX) directory of your G2 product directory.

Once you have merged the KL into your own KB, this workspace appears:



The current user mode is developer. Starting the KB activates all go-to-subworkspace and other buttons.

To start the KB:

➔ Select Main Menu > Start.

The G2 menu translations and the translated editor buttons do not appear unless the current language is Korean.

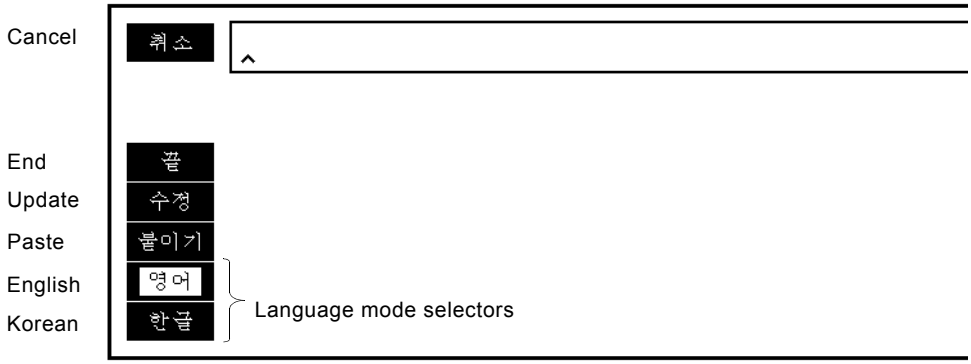
To change the current language to Korean:

1 Select Main Menu > System Tables > Language Parameters.

2 Edit the current-language attribute and select korean.

Entering Korean Text

When Korean is the current language in your KB, the editor includes two language modes to help you enter Korean text, as shown in the following figure:



Each time you invoke the editor, English is the default entry mode and English characters appear as you type.

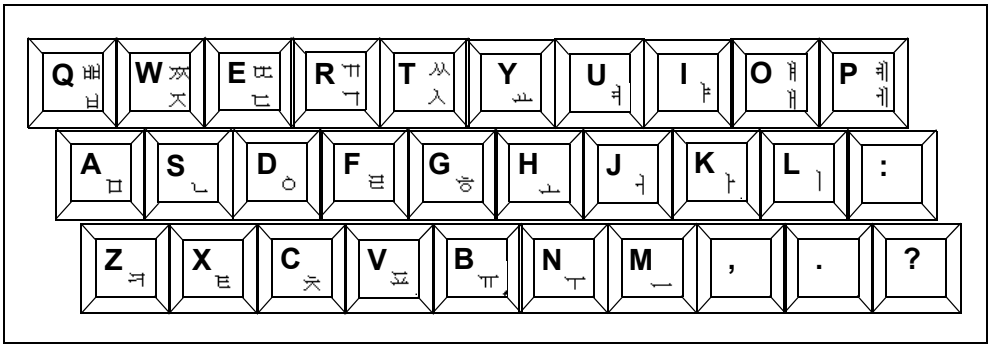
To toggle between language modes:

➔ Click the Korean activity button.

or

➔ Press Control + k.

When Hangul is the current entry mode, you can enter any Korean characters directly from the keyboard, using the standard Korean keyboard layout:

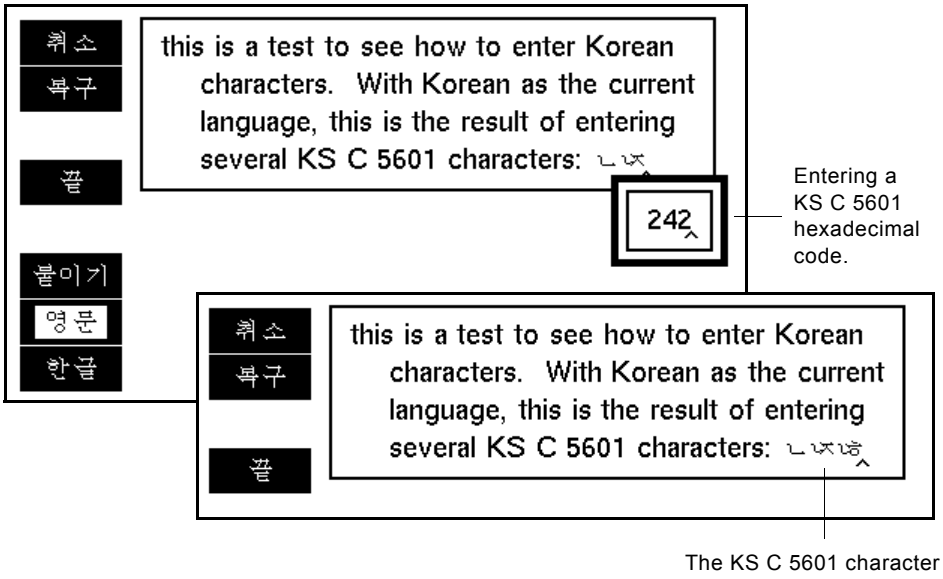


While you can enter Korean characters in the editor directly from the keyboard, you may need to enter other characters as well. When the current language is korean, you can enter any character from the KS C 5601 character set as a four-digit hexadecimal value. If you enter the same value when the current language is not korean, G2 interprets the value as Unicode character. If you do not know the hexadecimal code for the character you wish to enter, you can locate it in the `kscodes.kl`, which provides a complete set of KS C 5601 text inserters.

To enter a KS C 5601 code in the Text Editor:

1. Type Alt + i to invoke a secondary text entry box.
2. Enter the four-digit code, as shown next.

The character appears as soon as you enter a valid four-digit code:



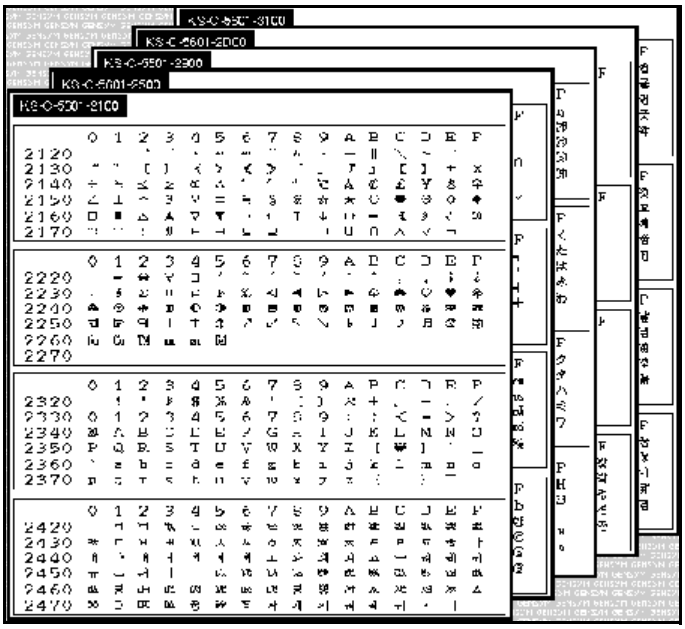
Alternatively, you can merge in the `kscodes.kl` KL and copy a character from the text inserters.

To merge `kscodes.kl` into your KB:

1 Select Main Menu > Merge KB.

2 Enter the name of the KL you wish to merge into your KB, `kscodes.kl`.

The `kscodes.kl` file is located in the `\g2\kbs\utils` (Windows) or `/g2/kbs/utils` (UNIX) directory of your G2 product directory, and displays a series of text inserter workspaces, each with multiple sets of characters:



The hexadecimal codes are listed in series in the upper left-hand corner of each workspace containing the KS C 5601 character set. Chinese characters begin at 4A21 hexadecimal on the workspace called KS C 5601-4900.

You can use text inserters to access Hangul or Chinese characters from these workspaces to use in your KB.

Instead of typing characters in the Text Editor, you can click on any character from one of the KS-5601 text insertion menus included in the `kscodes.kl`. G2 copies the character to the editor.

Using the Chinese Language Facilities

The Chinese language facilities support all system-defined G2 features, plus the ability to input characters from G2 on Windows platforms, and to run and view applications in Chinese. KBs developed in Chinese are platform independent and can run without modification in other languages. Chinese fonts are available for Chinese characters specified in Unicode Version 2.0.

A knowledge library, `gbcodes.kl`, is supplied and contains codes for the GB 2312 characters. These characters are the simplified characters used mainly by the Peoples Republic of China. This knowledge library is useful for reference purposes, but not as a primary method for character input. The Windows Chinese language facility is recommended for inputting Chinese text.

Currently, G2 has no facilities for automatically customizing G2 Menus for Chinese. Changing the current-language attribute of the Language Parameters system table or the g2-window-specific-language attribute of the login dialog to Chinese has no effect on G2 menus. However, you can create your own language-definitions that localize system-defined and user menu choices and other facilities for Chinese. See the *G2 Reference Manual* for information on language translation definitions.

Using the Thai Language Facilities

G2 uses the correct character images to display symbol and text values that include Thai characters.

Because Thai text typically does not use spaces, G2 supports the zero-width-space character. This character functions as a space character for purposes of word wrapping, but it takes up no horizontal space between characters on either side. You can insert the character by using its Unicode hexadecimal character code, which is 200B. To insert this Unicode hexadecimal code in the editor, you enter Alt+I 2 0 0 B.

Using the Russian Language Facilities

The Russian language facilities can help you to develop, run, and view applications in Russian using the Cyrillic alphabet (ISO 8859-5). KBs developed in Russian are platform independent, and can run without modification in any G2 language environment.

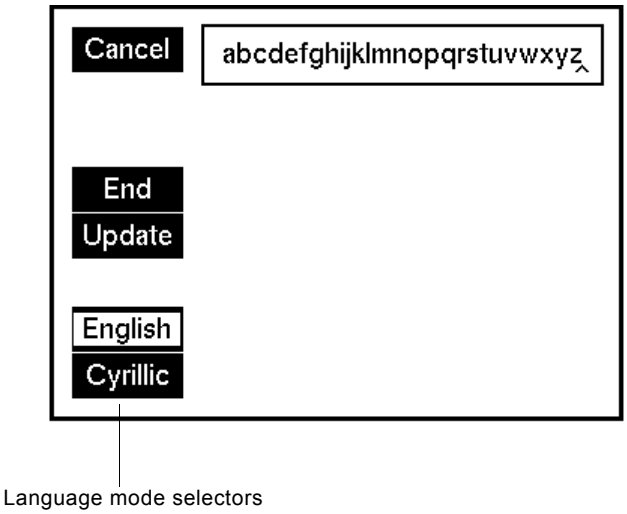
To enter Cyrillic text, you need only change the current language to Russian.

To change the current language:

1 Select Main Menu > System Tables > Language Parameters.

2 Edit the current-language attribute and select russian.

When Russian is the current language in your KB, the editor includes two language modes to help you enter Cyrillic, as shown in the following figure:



Each time you invoke the editor, English is the default entry mode and English characters appear as you type.

To toggle between language modes:

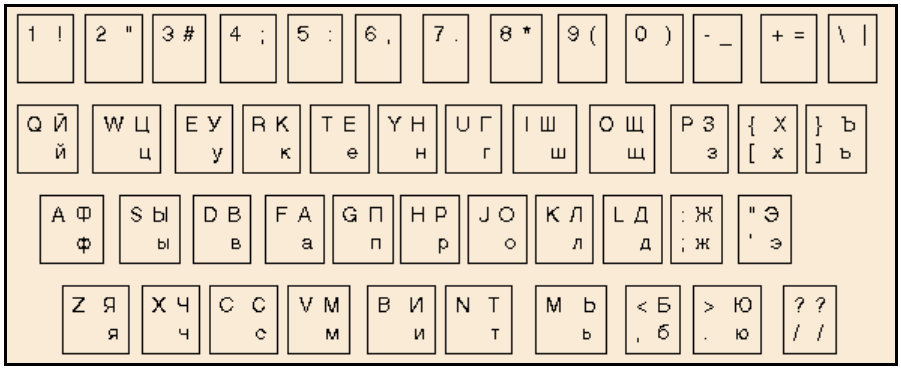
➔ Click the Cyrillic activity button.

or

➔ Press Control + k.

Entering Cyrillic Characters

When Cyrillic is the current entry mode, you can enter any letters from the Cyrillic alphabet directly by using the standard keyboard layout:



Entering Additional Cyrillic Characters

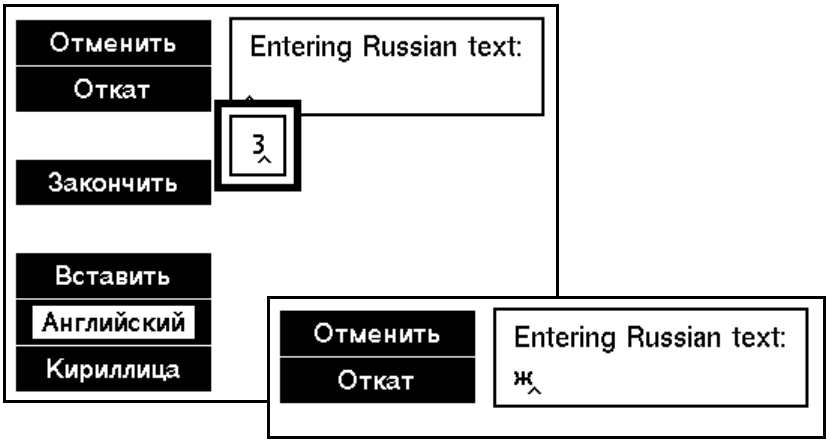
While you can enter Cyrillic characters in the editor directly from the keyboard, you may need to enter other characters as well. When the current language is

russian, you can enter any character from the ISO-8859-5 character set as a two-digit hexadecimal value.

To enter an ISO-8859-5 code in the editor:

1. Type Alt + i to invoke a secondary text entry box.
2. Enter the two-digit code, as the next example shows.

The character appears as soon as you enter a valid two-digit code. This example shows entering 36 for the Cyrillic small letter zhe:



49

G2 Character Support

Presents a description of the G2 character support through Unicode.

Introduction

Characters constitute the information in all text that G2 displays, imports from files and other sources of data, and exports to files and other data destinations. Characters also constitute the information contained in each value declared with the G2 types symbol or text.

G2 character representation is provided by the Unicode Worldwide Character Standard, which supports the storage, exchange, processing, and display of text for most of the world's modern and classical written languages. Supported characters cover the principal languages of the Americas, Europe, Middle East, Africa, India, Asia, and Pacifica.

KBs supporting multiple languages, or those importing data into a KB from an external character set, or exporting data from a KB to an external source, need information about G2 character support.

Unicode Character Support

Unicode represents each character code as a 16-bit unsigned integer in the range 0 - 1000000. Within that encoding space, Unicode separates character representation into four consecutive zones:

- Alphabetic (A zone), containing all general alphabetic, punctuation, and symbolic characters.
- Ideographic (I zone), containing the Han ideographic characters.
- Open (O zone), reserved for future use.
- Restricted (R zone), for private and compatibility characters.

Representing each character within two-bytes guarantees a uniform presentation. Regardless of the character or language, whether an English A, or a Japanese ideograph, text handling facilities are certain of a single character within every two bytes of data. Such uniformity prevents the need for numerous methods and techniques employed with previous character sets to determine how many bytes character encoding entailed.

Detailed information about the Unicode character set is not presented in this document. Information about the Unicode standard and the numerical representation of its supported characters is available online at the time of this publication at:

```
http://www.unicode.org
```

Non-Unicode Character Support

As the Unicode character set continues to be adopted on a global basis, developers may still need to support numerous other character sets for importing and exporting KB data. G2 provides several file I/O system procedures, described in the *G2 System Procedures Reference Manual*. To facilitate conversion to and from Unicode to other character sets, G2 provides the functionality to convert:

- Characters from various imported character sets into Unicode characters.
- Unicode characters into other character sets for exporting data.

Functions for character conversion are presented in Character Set Conversion Functions.

One of the character sets that G2 provides conversion functionality for is the **Gensym character set**. The Gensym character set was the default character set in previous G2 releases.

This chapter identifies the characters in the Gensym character set and shows how to encode each character in files and in data streams that are composed and manipulated outside of G2.

You can use the Gensym character set to:

- Compose attribute files, especially those that load attributes that must contain symbol and text values.

  Attribute files are a superseded capability. For more information, see Appendix F, Superseded Practices.

- Compose GFI input files, especially those that load symbol and text values into symbolic variables, text variables, symbolic parameters, and text parameters.

  GFI is a superseded capability. For more information, see Appendix F, Superseded Practices.

- Write GSI bridge applications, especially those that send and receive symbol and text values to and from G2.
- Write applications outside of G2 that communicate with G2 through a remote procedure call (RPC) interface, especially those that pass and return symbol and text values to and from G2.
- Write applications outside of G2 that work with files written by G2's Inspect facility.

Tip With few exceptions, you can use G2's Text Editor to input any character in the Unicode character set. These features are described in The Text Editor and in Natural Language Facilities.

Defining the Gensym Character Set

The Gensym character set is comprised of these sets of characters:

- A subset of the ASCII standard character set.
- The newline character.
- The tab character.
- A set of alphabetic and symbol characters common to English and other natural languages, referred to as **special characters**.
- The characters in the Japanese Industrial Standard (or JIS) X 0208-1990 standard character set.

- The characters in the KS C 5601 standard Korean character set.
- The characters in the International Standards Organization (or ISO) 8859-5 standard Russian character set.

Subset of ASCII Character Set and Special Characters

The Gensym character set includes 95 characters of the standard ASCII character set. The Gensym character set also includes 69 special characters. The figure Two Groups of Characters in the Gensym Character Set shows how these characters appear when shown in free-text items upon a workspace in a G2 window.

To encode these characters in a file or data stream outside of G2, see Encoding ASCII Characters and Special Characters.

Note The characters @ (at sign), \ (backslash), and ~ (tilde) are escape characters in the Gensym character set. To express these as literal characters, you must use the escape sequences defined for them, as shown in the table Encoding for ASCII and Special Characters.

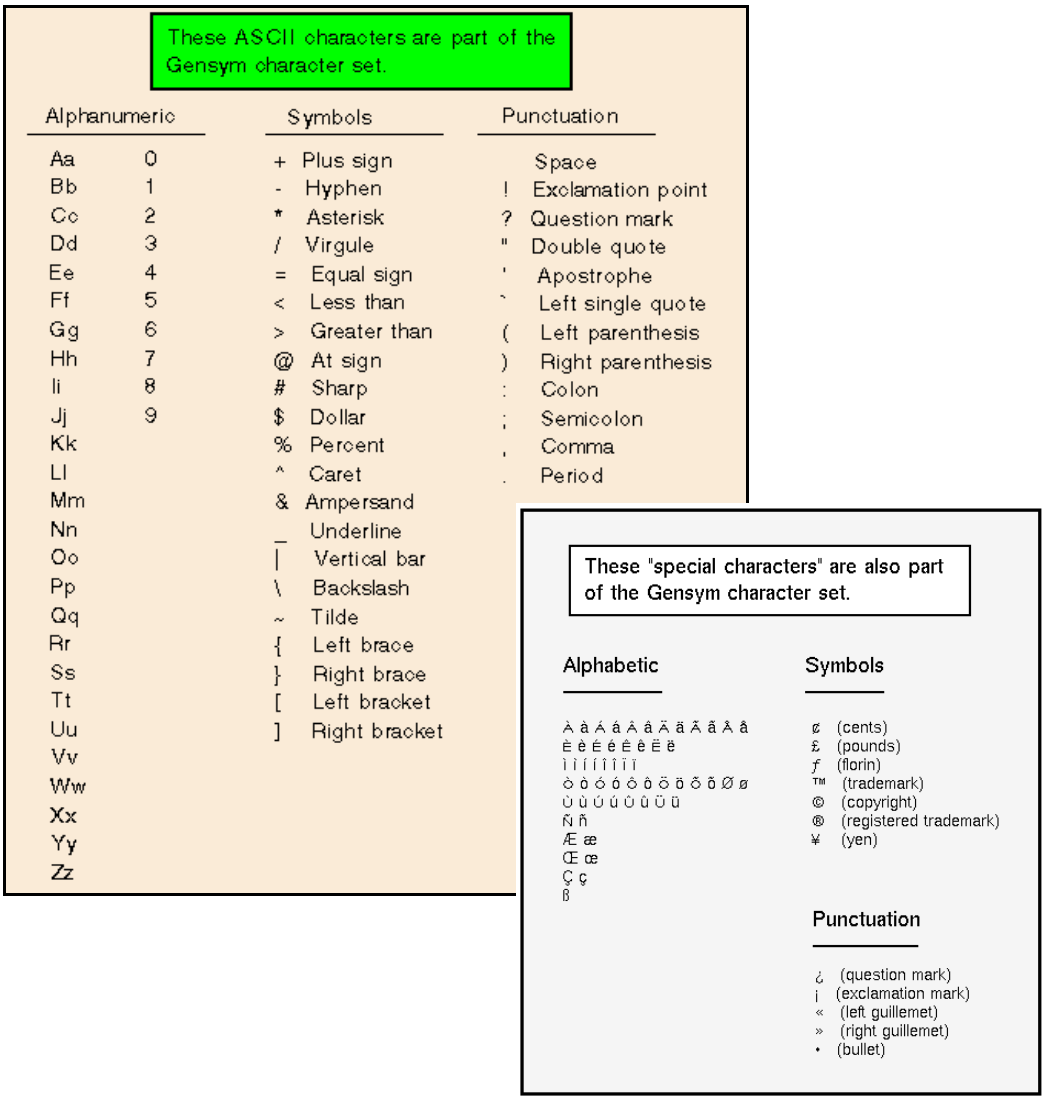
Other Standard Character Sets

The Gensym character set includes the hiragana, katakana, and kanji characters specified in the JIS X 0208-1990 standard character set. To encode these characters in a file or data stream outside of G2, see Encoding Japanese Characters.

The Gensym character set includes the Hangul characters specified in the KS C 5601 standard character set. To encode these characters in a file or data stream outside of G2, see Encoding Korean Characters.

The Gensym character set includes the Cyrillic characters specified in the ISO 8859-5 standard character set. To encode these characters in a file or data stream outside of G2, see Encoding Russian Characters.

Two Groups of Characters in the Gensym Character Set



Using Escape Characters

For file import operations, the Gensym character set defines three characters, called **escape characters**, for the following special uses:

- Use the ~ (tilde) character as the escape character for encoding a special character in the Gensym character set. You also use ~ as the escape character to encode the characters @ ~ and \ themselves as literal characters. The table Encoding for ASCII and Special Characters shows each character in the Gensym character set that must encoded using the ~ character.

Note Using @ as an escape character for the Gensym character set is different from using @ as the quoting character in a symbol value or text value in G2. Using @ as the quoting character is described in Working with Characters in a Symbol Value and in Working with Characters in a Text Value.

- Use the \ (backslash) character as the escape character for encoding a character in the Gensym character set that is a Japanese, Korean, or Russian character.

Using the ~ Escape Character

Use the ~ (tilde) character as an escape character to encode a special character in the Gensym character set for importing text into G2.

For example, you might wish to include the trademark (™) character from the Gensym character set in a symbol or text value that G2 can use. To do so, specify the pair of characters ~: in the value. Thus, to produce this series of characters:

Acme™

as a text value, you must encode this series of characters:

```
"Acme~:"
```

Using the @ Escape Character

Use the @ (at sign) escape character only in the two-character sequence @L, to encode the newline character in the Gensym character set. This corresponds to the explicit line-feed that you produce using the Text Editor by pressing Control + j.

Note The newline character in the Gensym character set does *not* signify the ASCII line-feed character.

You might also wish to use the @ as a literal character in a symbol or text value. To do so, the value in G2 must specify the two characters @@. Thus, to produce this series of characters:

service@gensym.com

you must utilize the @ character as both a quoting character and as a literal character. To produce these characters in a symbol value in G2, specify this series of characters as its value:

service@@gensym.com

To encode the @ character for use in this way, you must express the intended text value as this series of characters:

```
" service~@~@gensym.com"
```

Note Text imported into a KB using the @ escape character may not display correctly in an item attribute display or a readout table.

Using the \ Escape Character

You might wish to include a character from the Japanese, Korean, or Russian alphabets in a symbol or text value that G2 can use. To do so, you use the \ character to signify a series of ASCII characters whose combined value represents the encoded character. For an explanation of this encoding scheme, see Encoding Japanese Characters.

Thus, to encode a Japanese, Korean, or Russian character, you specify a series of characters that signify two or three 8-bit values, represented as hexadecimal values. For example, this series of characters:

```
\+;
```

represents the encoded value 11067 (or 0x2b3b hexadecimal) in the Gensym character set.

Encoding ASCII Characters and Special Characters

When creating an external file to import characters into G2, you need to encode the subset of characters in the Gensym character set comprised of ASCII characters and other special characters. To do so, use the encoding summarized in the table Encoding for ASCII and Special Characters.

Encoding for ASCII and Special Characters

Character	Encoding	Character	Encoding	Character	Encoding
(space)	(space)	A	A	N	N
!	!	B	B	O	O
"	"	C	C	P	P
#	#	D	D	Q	Q
$	$	E	E	R	R
%	%	F	F	S	S
&	&	G	G	T	T
'	'	H	H	U	U
(	(	I	I	V	V
)	)	J	J	W	W
*	*	K	K	X	X
+	+	L	L	Y	Y
,	,	M	M	Z	Z
-	-	a	a	n	n
.	.	b	b	o	o
/	/	c	c	p	p
:	:	d	d	q	q
;	;	e	e	r	r
<	<	f	f	s	s
=	=	g	g	t	t
>	>	h	h	u	u
?	?	i	i	v	v
[	[	j	j	w	w
]	]	k	k	x	x
^	^	l	l	y	y
_	_	m	m	z	z

Encoding for ASCII and Special Characters

Character	Encoding	Character	Encoding	Character	Encoding
‘	‘	0	0	@	~@
{	{	1	1	~	~~
|	|	2	2	\	~\
}	}	3	3	Tab	\ ()
		4	4		
		5	5		
		6	6		
		7	7		
		8	8		
		9	9		
ä	~a	Ñ	~N	Ù	~Z
Ä	~A	ö	~o	ë	~0
á	~b	Ö	~O	Ë	~1
Á	~B	ó	~p	ï	~2
ç	~c	Ó	~P	Ï	~3
Ç	~C	ô	~q	ã	~4
â	~d	Ô	~Q	Ã	~5
Â	~D	å	~r	õ	~6
é	~e	Å	~R	Õ	~7
É	~E	ß	~s	û	~8
ê	~f	ø	~t	Û	~9
Ê	~F	Ø	~T	©	~|
è	~g	ü	~u	™	~:
È	~G	Ü	~U	®	~;
ì	~h	ú	~v	¿	~?
Ì	~H	Ú	~V	¡	~!
í	~i	œ (oe)	~w	«	~<
Í	~I	Œ (OE)	~W	»	~>
î	~j	æ (ae)	~x	£	~#
Î	~J	Æ (AE)	~X	¥	~$
à	~m	ò	~y	*ƒ*	~&
À	~M	Ò	~Y	•	~*
ñ	~n	ù	~z		

Encoding a Tab Character

To encode a tab character in a file for the Gensym Character Set:

➔ Enter these characters sequentially:

backslash (\ ) space ( ) left-parenthesis (( ) right-parenthesis ( ))

Such a character sequence would appear in a file as:

```
\ ()
```

Encoding Japanese Characters

The Gensym character set includes the characters defined in the Japanese Industrial Standard (or JIS) X 0208-1990 character set. G2 uses the correct ideograms to display symbol and text values that include these characters, regardless of whether the Japanese language facility is in use.

For information about the Japanese language facility, see Using the Japanese Language Facilities.

Each JIS character has a distinct representation in the Gensym character set. A JIS character is represented as a *kanji code*, which is a positive integer that can be represented in two bytes. The first byte contains the most significant eight bits of the kanji code's value.

To express a JIS character that is part of the Gensym character set:

➔ Specify the \ (backslash) escape character, followed optionally by a prefix ASCII character, followed by two ASCII characters.

The bit pattern of the two or three ASCII characters represents the value of the JIS character in the Gensym character set. Either the prefix ASCII character or the first ASCII character represents the most significant eight bits of the character in the Gensym character set.

To determine a kanji code's representation in the Gensym character set, you perform the following algorithm (expressed in pseudocode):

```
/* Operators:
   != : is not equal to
   >> : shift bits right
   &  : AND operator
   %  : MODULO operator
*/
/* Include a prefix ASCII character? */
if (kanji_code >> 13) != 1 then
   /* Produce the prefix ASCII character */
   prefix_character = (kanji_code >> 13) + 32
```

```
/* Produce first ASCII character */
first_character = ((kanji_code & 0x1fff) / 95) + 40

/* Produce second ASCII character */
second_character = ((kanji_code & 0x1fff) % 95) + 32
```

For example, to represent the kanji code 8504 (or 0x2138 hexadecimal), follow this sequence of steps:

1. (0x2138 >> 13) is 1.
2. The condition (1 is not equal to 1) is false, so derive no prefix ASCII character.
3. (0x2138 & 0x1fff) is 312.
4. (312 / 95) is 3.28, rounded down to 3.
5. (3 + 40) is 43, or 0x2b hexadecimal, so the first ASCII character is the + (plus sign) character.
6. (312 % 95) is 27.
7. (27 + 32) is 59, or 0x3b hexadecimal, so the second ASCII character is the ; (semicolon) character.

Thus, encode the kanji code 8504 in the Gensym character set as this series of characters:

```
\+;
```

For example, to represent the kanji code 17228 (or 0x434c hexadecimal), follow this sequence of steps:

1. (0x434c >> 13) is 2.
2. The condition (2 is not equal to 1) is true, so derive a prefix ASCII character.
3. 2 + 32 is 34, so the prefix ASCII character is the " (double quotes) character.
4. (0x434c & 0x1fff) is 844.
5. (844 / 95) is 8.88, rounded down to 8.
6. (8 + 40) is 48, or 0x30 hexadecimal, so the first ASCII character is the 0 (zero digit) character.
7. (844 % 95) is 84.
8. (84 + 32) is 116, or 0x74 hexadecimal, so the second ASCII character is the t (lowercase T) character.

Thus, encode the kanji code 17228 in the Gensym character set as this series of characters:

```
\"0t
```

Encoding Korean Characters

The Gensym character set includes the characters defined in the KS C 5601 standard character set for the Korean language. G2 uses the correct ideograms to display symbol and text values that include these characters, regardless of whether the Korean language facility is in use.

For information about the Korean language facility, see Using the Korean Language Facilities.

Each KS C 5601 character has a distinct representation in the Gensym character set. A KS C 5601 character can be represented in two bytes. The first byte contains the most significant eight bits of the KS C 5601 character's value.

To express a KS C 5601 character that is part of the Gensym character set:

➔ Increment the second byte's value by 94 (or 0x5e hexadecimal), then encode the character's two bytes as described in Encoding Japanese Characters.

For example, given the KS C 5601 character code 8483 (or 0x2123 hexadecimal), the character's second byte contains the hexadecimal value 0x23. Add the hexadecimal value 0x5e to this, to obtain the sum 0x81. Next, encode the character's incremented value 8577 (or 0x2181 hexadecimal) as for a JIS character.

Encoding Russian Characters

The Gensym character set includes the characters defined in the International Standards Organization (or ISO) 8859-5 standard character set. G2 uses the correct Cyrillic characters to display symbol and text values that include these characters, regardless of whether the Russian language facility is in use.

For information about the Russian language facility, see Using the Russian Language Facilities.

A ISO 8859-5 character consists of one byte, representing a 7-bit value. Each ISO 8859-5 character has a distinct representation in the Gensym character set.

To express a ISO 8859-5 character that is part of the Gensym character set:

1. Assign the value zero into a two-byte value.
2. Assign the character's 7-bit code value into the two-byte value. The character's 7-bit value must be the least significant seven bits of the resulting two-byte value.
3. Increment the entire two-byte value by 8192 (or 0x2000 hexadecimal), then encode the resulting two bytes as described in Encoding Japanese Characters.

For example, given the ISO 8859-5 character code 81 (or 0x51 hexadecimal), assign its value into a two-byte value containing the value 0 (zero), producing the

sum 81 (or 0x0051 hexadecimal). Increment this value by 8192, and encode the value of the resulting sum 8273 (or 0x2051 hexadecimal) as for a JIS character.

Translating from the Gensym Character Set

Use the pseudocode procedure shown below as the basis for your own routine to translate a series of characters in the Gensym character set to characters in another format, such as the KS C 5601, JIS, ASCII, or ISO Latin-1 character sets. Alternatively, you can use the character set conversion functions described in Character Set Conversion Functions.

The following procedure should be used in a program that maintains an index into a string, beginning at character zero (0). The program calls this procedure to get the next character and to update the index, based on how many characters were used to represent the Gensym character set character.

This pseudocode procedure gets the character code for the Gensym character set character at position *i* in string *s*; that is, *s(i)*.

1 Let *c* be the character *s(i)*. Let i = i + 1.

2 If:

- *c* is the character @, then let *c* be the character *s(i)* and *i* = *i* + 1.
- *c* is the character L, then return the character code or codes for newline.
- *c* is an alphabetic character, then consider this to be an undefined situation (that is, an error) and exit.

Otherwise, return the ASCII character code for *c*. The result is an ASCII-encoded character.

3 If:

- *c* is the character ~, then let *c* be the character *s(i)* and *i* = *i* + 1.
- *c* is not in the table Encoding for ASCII and Special Characters, then consider this to be an undefined situation (that is, an error) and exit.

Otherwise, return the character code or codes for the character that *c* corresponds to in the table Encoding for ASCII and Special Characters. This result might be in ASCII or in some other (undefined) character encoding. Note that the characters ~ @ and \ are included in this set.

4 If:

- *c* is the character \, then let *c* be the character *s(i)* and *i* = *i* + 1.
- The character code for *c* is less than 40, then let *d1* = the character code for *c*, let *d2* = the character code for *s(i)*, let *i* = *i* + 1, let *d3* = *s(i)*, and return the result of the following formula:

```
/* arithmetically shift d1 left 13 bits */
(d1 << 13)
+ (d2 * 95)
+ d3
+ -265976
```

Otherwise, let *d1* = the character code for *c*, let *d2* = the character code for *s(i)*, let *i* = *i* + 1, and return the result of the following formula:

```
(d1 * 95)
+ d2
+ 4360
```

The result is a JIS-encoded character.

5 Return the code for *c*. The result is an ASCII-encoded character.

When this procedure completes, the index *i* points either to the next possible character or to the end of the string *s*.

Part VII

Debugging and Optimization

50

Error Handling

Describes the G2 error-handling capabilities.

Introduction

G2 error handling is object-oriented, and is based on the following capabilities:

- The system-defined class error, an extensible instantiable class for controlling error handling.
- The on error statement, which takes a local variable whose class is error or a subclass of error.
- The signal statement, which specifies an instance of error or of a subclass of error.
- The ability to use object-oriented techniques to handle different types of errors in different ways.
- The ability to shadow the system-defined default error handler with a user-supplied procedure or a method definition.

Note This chapter covers the essentials of G2 error handling. More sophisticated techniques for managing errors are available through GFR and GERR. See the *G2 Foundation Resources User's Guide* and *G2 Error Handling Foundation User's Guide* for details.

Superseded Error Handling Techniques

This chapter covers only object-oriented error handling, in which errors are described by instances of class error. Such error handling is standard in G2 5.0 and higher. In G2 4.0, an error was described using a symbol and a text string. Object-oriented error handling supersedes this practice.

If your KB was new beginning in G2 5.0, you can ignore symbol/text error handling. If your KB already uses G2 4.0 symbol/text error handling, you can continue to use it if you prefer. The syntax is described under Superseded On Error Statement Syntax and Superseded Signal Statement Syntax.

The two types of error handling can be freely intermixed: G2 translates automatically between them, as described under Mixing Error Handling Techniques. New code should use only the object-oriented techniques described in this chapter.

G2 Error Handling Concepts

G2 provides two levels of error handling:

- A **block error handler** can be defined on any block in a procedure. This handler traps errors that occur within the associated block.
- A **default error handler** traps any error that no block error handler traps.

Block error handlers can be nested in two ways:

- A block with an error handler can contain a subordinate block that defines its own error handler.
- A block with an error handler can execute a call that passes control to a block in another procedure that defines its own error handler.

When an error occurs, G2 first looks for an error handler defined on the block that experienced the error. If none exists, G2 searches back through any containing and calling blocks looking for a block error handler. If no block error handler exists, G2 invokes the default error handler, which always exists.

G2 provides two statements for managing errors:

- on error: Associates an error handler with a block.
- signal: Invokes the applicable error handler.

Block error handlers (on error statements) can be defined only in procedures. When an error occurs in any other context, such as a rule, G2 invokes the default error handler.

G2's default error handling capabilities are synchronous: they do not enter a wait state during handling of an error. This protects the context within which the error occurred from asynchronous changes. User-defined error handling capabilities can allow other processing if appropriate. For information on wait states, see Allowing Other Processing.

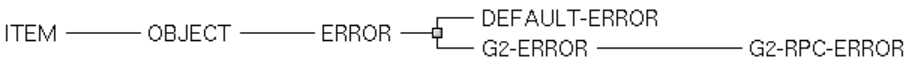
G2 Error Classes

G2 error handling is object-oriented. The basis is a system-defined extensible instantiable class, error. The error class defines one class-specific attribute: a text attribute called error-description.

Any class that is or inherits error is called an **error class**. Every error class defines a separate type of error. G2 provides the following system-defined error classes:

- g2-error: An error that occurred during the ordinary execution of G2.
- g2-rpc-error: An error that occurred in the context of a remote procedure call.
- default-error: Provides compatibility with G2 4.0 error handling.

The class inheritance of the system-defined error classes is:



The class error is inheritable and instantiable in user code. However, the subclasses g2-error, g2-rpc-error, and default-error are reserved for use by G2, and are neither inheritable nor instantiable in user code.

You can define as many additional types of errors as you like by defining additional subclasses of error. You can add class-specific properties to such subclasses as desired, and use multiple inheritance to combine them in arbitrarily complex ways.

Both G2 and a user-supplied procedure can instantiate any error class. Such an instance is called an **error object**. You can use such an object to:

- Specify a particular type of error by providing an error object of the appropriate class.
- Associate text with a particular error by setting the error-description attribute of an error object.

Error objects have a read-only hidden attribute called error-source-trace, which provides a history of error descriptions and source information, and is updated each time the error is signaled. The error-source-trace attribute has this syntax:

```
sequence
(structure
    (error-trace-description: text,
    error-trace-source-item: symbol,
    error-trace-source-line: integer,
    error-trace-source-column: integer)
. . .)
```

Each structure uses the current values from the error at the time it was signaled, where the first entry corresponds with the most recent error that was signalled.

For more information, see Obtaining Source Information From the Error Object,

Defining an Error Handler

The syntax of an on error statement is:

```
on error (local-name)
    statement [; statement]...
end
```

Argument	Description
local-name	A local name whose type is class error or any subclass of error.

An on error statement appears immediately after the end statement of the block to which it applies. For example:

```
demonstrate-block-error-handler()
errobj: class error;
begin
     post "Call sigproc now.";
     call sigproc(0);
     post "Return from sigproc."
end
on error (errobj)
     post "An error of the class [the class of errobj] occurred:
          [the text of the error-description of errobj]";
delete errobj
end
```

An on error statement executes if and only if G2 signals an error within the scope of the statement, or a signal statement executes within the scope of the statement. Otherwise, control skips over the on error block and continues sequentially.

Handling Errors in a Procedure

When G2 signals an error during procedure execution, or a user-supplied procedure signals an error, the signaler specifies the type of the error by supplying an error object of appropriate type.

- When G2 signals an error during KB execution, the error object is a transient g2-error or g2-rpc-error, and G2 places a description of the error in the object's error-description attribute.
- When a user-supplied signal statement signals an error, the error object is a transient or permanent instance of any instantiable error class, and the user code optionally places a description of the error in the object's error-description attribute.

In either case, G2 searches the call stack back from the point where the error was signalled, looking for a block error handler whose type matches that of the error object. Since error is the parent all error classes, a block error handler that specifies the class error matches any error. A handler that specifies a more specific subclass matches only errors of that class and its subclasses.

If G2 encounters a block error handler whose class matches the class of the error object, G2 executes the statement(s) in the body of the error handler. These statements can reference the error object in any standard way.

If G2 never finds a matching block error handler, it executes the default error handler. The system-defined default error handler is an implicit on error statement that:

- Takes an object of class error, and thus matches any error.
- Posts the class of the error, and the contents of its error-description attribute, to the Operator Logbook.

After a user-defined block error handler has executed, control passes to the first statement after the end of the block that experienced the error. If no such statement exists, the procedure containing the block returns.

Obtaining Source Information From the Error Object

If the error-generating procedure has been compiled with source-code position information and you have defined your own error handler, G2 can tell you what statement in your procedure caused the error. G2 does through three attributes on your error object: error-source-line, error-source-column, and error-source-item.

In the next example procedure, the on error statement directs G2 to create a g2-error object. The procedure simply posts the name of the error-causing procedure and the line and column positions of the error to the Message Board:

```
test-error-location()
procedure-error: class error;
begin
    call undefined-procedure()
end
on error (procedure-error)
    post "An error occurred in [the error-source-item of procedure-error]
        on line [the error-source-line of procedure-error]
        and in column [the error-source-column of procedure-error].";
    delete procedure-error
end
```

The error source line is indexed beginning at 1; and the error column position is indexed starting at 0. The line-column location is within the statement that generated the error, but G2 is not always able to pinpoint the exact position within the statement. The line and column values are -1 if G2 is unable to determine what procedure statement generated the stack error.

For more information on G2's source-code error location facility, see Obtaining Procedure Source-Code Error Location Information.

Synchronous and Asynchronous Error Handling

Error handler invocation is synchronous: nothing else will execute in the G2 that executes the handler until that handler returns. If the error did not occur during execution of an RPC call, and the error handler does not execute an RPC call, the

G2 context is guaranteed to be the same when the handler returns as it was when the handler was invoked, except insofar as the handler itself has changed it.

When error handling either executes or occurs during execution of an RPC call, the remote process runs asynchronously from the error handler, and can change the context of the code that invoked the handler. When the handler returns, code that validates context may have to be rerun before code that assumes that context can safely continue execution.

Default Handler Example

The following procedure attempts to call an undefined procedure named sigproc:

```
demonstrate-block-error-handler()
begin
    post "Call sigproc now.";
    call sigproc(0);
    post "Return from sigproc."
end
```

If you invoke demonstrate-default-error-handler, the procedure:

1. Posts "Call Sigproc Now:" on the Message Board.
2. Attempts to call sigproc.

G2 cannot find the nonexistent procedure, so it:

1. Creates a transient error object of class g2-error and sets its error-description attribute to describe the error.
2. Signals an error with that error object.

To process the signaled error, G2:

1. Seeks a block error handler that takes a g2-error.
2. Does not locate such a handler,
3. Invokes the default error handler, passing it the error object.

The system-defined default error handler:

1 Posts the following message on the Operator Logbook:

```
#31   2:40:16 p.m.   Error:

No item named SIGPROC exists.

Operation: fetch SIGPROC
Activity: call statement
Within: DEMONSTRATE-DEFAULT-ERROR-
  HANDLER()
Aborting procedure stack from DEMONSTRATE-
  DEFAULT-ERROR-HANDLER().
```

2 Deletes the error object. See Error Object Memory Management for information on error object deletion.

An error handled by the default error handler aborts the procedure, so the statement posting "Return from Sigproc:" on the Message Board never executes.

Block Error Handler Example

The following procedure attempts to call an undefined procedure named sigproc. It is the same as the preceding example, except that it defines a block error handler on the calling procedure's begin-end block:

```
demonstrate-block-error-handler()
errobj: class error;
begin
     post "Call sigproc now.";
     call sigproc(0);
     post "Return from sigproc."
end
on error (errobj)
     post "An error of the class [the class of errobj] occurred:
          [the text of the error-description of errobj]";
delete errobj
end
```

If you invoke demonstrate-block-error-handler, the procedure:

1 Posts "Call Sigproc Now:" on the Message Board.

2 Attempts to call sigproc.

G2 cannot find the nonexistent procedure, so it:

1 Creates a transient error object of class g2-error and sets its error-description attribute to describe the error.

2 Signals an error with that error object.

To process the signaled error, G2:

1 Seeks a block error handler that takes a g2-error.

2 Locates the error handler associated with the procedure's begin-end block.

3 Invokes the handler, passing it the error object.

The block error handler:

1 Posts the following message on the Message Board (not the logbook):

```
#53   3:08:14 p.m.   An error of class G2-ERROR
  occurred: Error:

No item named SIGPROC exists.

Operation: fetch SIGPROC
Activity: call statement
Within: DEMONSTRATE-BLOCK-ERROR-
  HANDLER()
Local Names:
  ERROBJ: class g2-error = no value
```

2 Explicitly deletes the error object. See Error Object Memory Management for information on error object deletion.

The error aborts the block, and demonstrate-block-error-handler contains no further statements, so the procedure returns.

Note that the value of error-description in the Message Board is the same as in the Operator Logbook message in the previous example. It is the same because the error object generated by G2 is the same in both cases. The only difference is the handler that posts the information: the system-defined default error handler in the previous example, and a block error handler in this one.

Error Object Memory Management

During KB execution, error objects can be instantiated in indefinitely large numbers. Such objects must not be allowed to accumulate without limit, or the resulting memory leak will eventually consume all memory and abort G2.

When the system-defined default error handler receives a transient error object, it automatically deletes the object before returning. In all other cases, code that signals and handles errors must explicitly delete error objects as needed to prevent them from accumulating.

Since G2 cancels execution of a block that signals an error, the code that deletes an error object cannot follow the signal statement in that block. The most convenient place to delete an error object is in the error handler that receives it. Any other technique will do as well, provided that it can never fail.

Reusing Error Objects

When the system-defined default error handler receives a permanent error object, it does *not* delete the object before returning, and a block error handler need not delete the error object that it receives. An undeleted error object persists, and can be used by signal statements as desired.

This provision allows you to avoid the overhead of creating and deleting an object with every signaled error. You can create libraries of permanent error objects, and select from them as needed when errors occur.

Handling Non-Procedural Errors

Block error handlers can be defined only within procedures. When G2 signals an error in any context other than a procedure, it invokes the default error handler, passing it a transient g2-error object whose error-description attribute describes the error.

The system-defined default error handler posts to the Operator Logbook a message similar to that in the preceding example, then:

1. Deletes the error object.
2. Aborts the construct within which the error occurred.
3. Proceeds with the next scheduled task.

If the error occurred within the simulator, G2 halts the simulator.

The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices.

Signaling Errors in a Procedure

You can use the signal statement to signal an error during procedure execution. Such an error is essentially the same as an error signaled by G2 itself. The only difference is that the signal statement, rather than G2, supplies the error object.

The syntax of the signal statement is:

signal *error-object*;

Argument	Description
error-object	An instance of the class error or of any subclass of error.

When a signal statement executes, G2 looks for a block error handler whose class matches that of the *error-object* specified in the statement. If G2 finds such a

handler, it invokes the handler, passing it *error-object*. If G2 does not find a block error handler, it invokes the default error handler on *error-object*.

Thus the signal statement, rather than G2, defines the type of a signalled error. This feature, in conjunction with the ability to define error classes using multiple inheritance, allows very complex handling of signaled errors.

The signal statement affects the error message that appears on the Operator Logbook. Regardless of whether both the error-object creation and signalling are done within the same procedure, or a procedure signals an error object created in another procedure, the error message on the Operator Logbook contains information for both the error object and the signal. Selecting the go to source code menu option on the combined message locates the signal statement, not the error-generating statement.

The examples in this section assume that you have read Handling Errors in a Procedure, and do not reiterate the detailed descriptions of error handling that appear in that section.

Signaling the Default Error Handler

The following procedures are demonstrate-default-error-handler, the same procedure that appeared in Default Handler Example, and sigproc, the procedure that was undefined in that example. The procedure sigproc contains an example of a signal statement:

```
demonstrate-block-error-handler()
begin
    post "Call sigproc now.";
    call sigproc(0);
    post "Return from sigproc."
end

sigproc(index: integer)
zdev: class zerodivide;
begin
    create a zerodivide zdev;
    change the text of the error-description of zdev to "Cannot divide by zero.";
    if index = 0 then signal zdev;
    post "Ratio: [45387 / index]"
end
```

The signal statement in the example specifies an error object of class zerodivide, a subclass of error. Calling demonstrate-default-error-handler calls sigproc with an argument of 0, invoking sigproc's signal statement on an error object of class zerodivide.

Since no block error handler is in effect, G2 passes zdev to the default error handler. The system-defined default error handler posts the following to the Operator Logbook:

```
#74   4:19:16 p.m.   Error:  (ZERODIVIDE)

Cannot Divide by Zero

Operation: signal <unavailable argument>
Activity: signal error statement
Within: SIGPROC(0)
  <- DEMONSTRATE-DEFAULT-ERROR-
    HANDLER()
Local Names:
  INDEX: integer = 0;
  ZDEV: class zerodivide = ZERODIVIDE-XXX-3
Aborting procedure stack from DEMONSTRATE-
  DEFAULT-ERROR-HANDLER().
```

The system-defined default error handler has added additional information to the error-description of zdev. G2 provides such information as a convenience whenever a signaled error reaches the default error handler.

Signalling a Block Error Handler

The following procedures are demonstrate-block-error-handler, the same procedure that appeared in Block Error Handler Example, and sigproc, the procedure that was undefined in that example.

```
demonstrate-block-error-handler()
errobj: class error;
begin
     post "Call sigproc now.";
     call sigproc(0);
     post "Return from sigproc."
end
on error (errobj)
     post "An error of the class [the class of errobj] occurred:
          [the text of the error-description of errobj]";
delete errobj
end

sigproc(index: integer)
zdev: class zerodivide;
begin
     create a zerodivide zdev;
     change the text of the error-description of zdev to "Cannot divide by zero.";
     if index = 0 then signal zdev;
     post "Ratio: [45387 / index]"
end
```

Calling demonstrate-default-error-handler calls sigproc with an argument of 0, invoking sigproc's signal statement on zdev. G2 passes zdev to the block error handler of the calling block, which posts the following to the Message Board (not the logbook):

```
#96   4:54:28 p.m.   An error of class
  ZERODIVIDE occurred: Cannot Divide by Zero
```

When a signal statement communicates directly with a block error handler, G2 does not add any additional information: it uses the error object exactly as supplied by the signal statement.

Shadowing the Default Error Handler

You can shadow the system-defined default error handler with a **user-defined default error handler**. Such a handler is a procedure that takes one argument: an error object.

Shadowing the system-defined handler with a user-defined handler allows you to specify the handling of any or all errors, including those that originate outside the context of a procedure, such as rule, formula, and simulator errors.

The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices.

When a user-defined default error handler exists, G2 invokes it for any error whose type matches the class of the handler's argument. G2 then invokes the system-defined default error handler only for:

- Errors whose type does not match that specified by the user-defined handler.
- Errors that originate in the user-defined handler itself and are not trapped by a block error handler defined within it.

A user-defined default error handler can be a method declaration. When G2 invokes such a handler, it looks for a method defined for the class of the error object, and invokes it if it exists. If no such method exists, G2 invokes the system-defined default error handler.

Note More sophisticated techniques for managing error messages are available through GFR. See the *G2 Foundation Resources User's Guide* for details.

Creating a User-Defined Default Error Handler

Any procedure (including a method) that takes one argument of class error, or of any subclass of error, can be a user-defined default error handler. The procedure need have no other special properties.

If the procedure's argument is...	The procedure traps these errors...
error	All errors signalled by G2 or by a signal statement supplied by the user.
g2-error	All errors signaled by G2.
g2-rpc-error	Errors signaled by G2 as a result of a remote procedure call, such as a broken connection. This error handler does not trap g2-errors signalled in the remote procedure call itself.
A user-defined error class.	Any error of the user-defined class or any subclass of it.

To put a user-defined default error handler into effect, you must register the handler. G2 provides system procedures to register and deregister default error handlers, and to obtain the name of the handler currently in effect.

Resetting G2 does not affect handler shadowing: any registered handler remains in effect when G2 restarts.

To register a default error handler:

➔ g2-register-default-error-handler
(*procedure*: class procedure)

Registers a procedure or a method declaration to handle all errors.

To deregister a default error handler:

➔ g2-deregister-default-error-handler ( )

Deregisters the currently registered default error handler. The system-defined handler is then unshadowed, and behaves as if no user-defined handler had ever been registered.

To get the default error handler:

➔ g2-get-default-error-handler ( )
-> {*handler*: class procedure | false }

Returns the procedure or method declaration currently registered as the default error handler, or false if none is registered.

Each of these procedures is described in more detail in the *G2 System Procedures Reference Manual*.

Caution When you shadow the default error handler, be sure that the user-defined handler manages transient error objects correctly, or a memory leak will result. For details see Error Object Memory Management.

Mixing Error Handling Techniques

Error handling in G2 4.0 used a symbol and a text string, rather than an object, to describe an error. The essential nature of G2 4.0 error handling is the same as in G2 5.0 and higher, but the syntax is different: in the G2 4.0 implementation, the on error and signal statements each take two arguments—the symbol and the text string that describe the error—rather than one.

The syntax of the G2 4.0 error handling statements is described under Superseded On Error Statement Syntax and Superseded Signal Statement Syntax.

G2 distinguishes between the G2 4.0 and 5.0 and higher on error and signal statements by the number of arguments in the statement. To allow code to mix the two techniques without requiring special action, G2 automatically interconverts between them. Such interconversion is possible because:

- The *error-name* argument to a G2 4.0 on error or signal statement is analogous to the class name of the *error-class* argument to the corresponding G2 5.0 and higher statement.
- The *error-text* argument to a G2 4.0 on error or signal statement is analogous to the error-description attribute of the *error-class* argument to the corresponding G2 5.0 and higher statement.

The following table shows how G2 interconverts between one-argument and two-argument error handling:

- The rows of the table show the three ways in which a search for an error handler can originate.
- The columns show the two types of error handler on which the search can terminate.

	A one-argument on error statement	A two-argument on error statement
An execution error	No conversion needed.	G2 supplies the on error statement with an *error-symbol* and an *error-text*.
A one-argument signal statement	No conversion needed.	G2 converts the error object's class name to an *error-symbol* and the error object's error-description to an *error-text*, and supplies the two arguments to the on error statement.
A two-argument signal statement	G2 supplies the on error statement with an error object whose error-description is *error-text*, and whose class name is: • g2-error if *error-name* is error. • g2-rpc-error if *error-name* is rpc-error. • default-error otherwise.	No conversion needed.

Note The object-oriented default error handler is effectively a one-argument on error statement.

51

Debugging and Tracing

Describes G2 facilities that can assist in debugging your KB.

Introduction

G2 provides facilities for tracing and debugging items in your knowledge base (KB) by:

- Displaying **error and warning messages** about stack errors and KB discrepancies.
- Making **source-code error location** information available to you when a procedure generates a stack error.
- Displaying **trace messages** that give information on the execution state of procedures, methods, rules, formulas, and display expressions.
- Setting **breakpoints** that pause G2 between steps in your executing code so you can view trace messages and examine other KB data.
- Using **dynamic breakpoints** and **stepping through procedure code**, using the Windows debugger.
- Displaying the **disassembled code** for procedures, methods, and rules.
- Writing G2-state information after an internal error to an abort workspace and to the launch window.
- Writing logbook messages to a file.

Many of the debugging facilities are controlled by the values you specify for the Debugging Parameters system table. For a summary description of these attributes, see Debugging Parameters.

Displaying Error and Warning Messages

By default, G2 displays Operator Logbook messages for some KB discrepancies and all stack-errors that do not have user-defined error handlers. The value of the warning-message-level attribute in the Debugging Parameters system table determines which messages G2 displays.

Note See Error Handling for information on error handling and its effect on error display.

To specify the level of warning and error messages for display:

➔ Edit the warning-message-level attribute in the Debugging Parameters system table, and specify one of the following values:

- 0 (no warning messages)
- 1 (kb errors only)

- 2 (kb errors and deficiencies)
- 3 (kb errors, deficiencies, and other conditions)

The default level is 2, which specifies that G2 display all Level 1 and Level 2 warning messages, as well as all logbook-bound stack-error messages. Stack-error messages are displayed for all levels except Level 0. Specifying Level 0 directs G2 not to display any warning or stack-error messages.

Internally, each warning message has a warning-message level, which is sometimes included in the message display. You might want to specify a higher level when debugging your KB, and a lower level when you are finished debugging.

Note See Controlling Error and Warning Message Displays for more information on what kinds of KB deficiencies are displayed for each warning-message level.

This example shows a variable that has a non-existent data server in its data-server attribute, and shows the warning message that is displayed when the warning-message level is set to Level 1 or higher:

#63 3:52:16 p.m. Inconsistencies were detected in TEST-VARIABLE; see its notes for details.

TEST-VARIABLE, a quantitative-variable

Options	do not forward chain, breadth first backward chain
Notes	OK, and note that a validity interval of "supplied" (the default value) is only valid when the data server is "inference engine"
Item configuration	none
Names	TEST-VARIABLE
Tracing and breakpoints	default
Data type	quantity
Initial value	none
Last recorded value	no value
History keeping spec	do not keep history
Validity interval	supplied
Formula	none
Simulation details	no simulation formula yet
Initial value for simulation	default
Data server	non-existent-server
Default update interval	none

Obtaining Procedure Source-Code Error Location Information

When a procedure in your KB generates a stack error, G2 can tell you what statement in your code is responsible for the error. G2 does this by retrieving the source-code annotation location information it creates when it compiles your procedure code. The position information is within the statement in your procedure which is responsible for the error. Only procedures are compiled with source-code location information, and the information is saved with the KB.

When a stack-error is generated by a procedure compiled with annotation information, G2 makes the error source-code location available from two sources:

- From the error message G2 posts to the Logbook.
- From your error-handler object.

Controlling the Creation of Error-Location Information

G2 generates source-code position information only when the generate-source-annotation-info attribute of the Debugging Parameters system table is set to its default value of yes. The creation of annotation information slightly increases compile time and results in a slightly larger KB, but has little impact on run-time performance. At any time, you can direct G2 to refrain from creating annotation information.

To control the creation of source-code annotation information:

1 Select Main Menu > System Tables > Debugging Parameters.

2 Specify yes/no for the generate-source-annotation-information attribute.

G2 does not generate source-code position information when the generate-source-annotation-info attribute is no. If you wish to remove existing source-code location information from procedures, you must recompile them.

To remove existing source-code annotation information from your procedures:

1 Set the generate-source-annotation-information attribute on the Debugging Parameters system table to no.

2 Select Main Menu > Inspect and enter recompile every procedure in the edit box, or recompile procedures individually.

Note G2 always removes source-code information when you text strip a procedure regardless of the value of the generate-source-annotation-information attribute.

Obtaining Error-Location Information from the Logbook

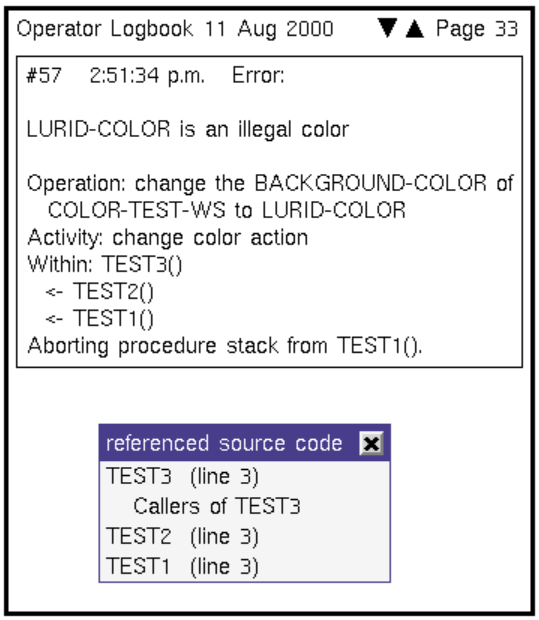
When G2 signals an error, it posts an error message to the Operator Logbook.

To find the source-code location of that error:

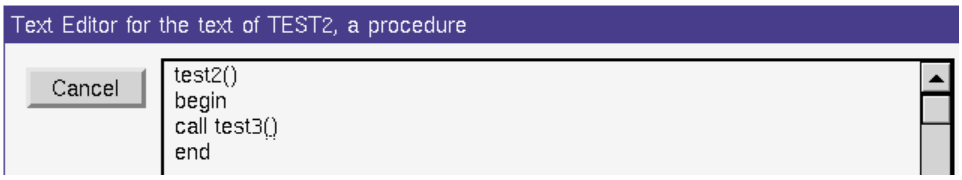
1 Click on the error message to bring up its menu.

2 Select the go to source menu choice.

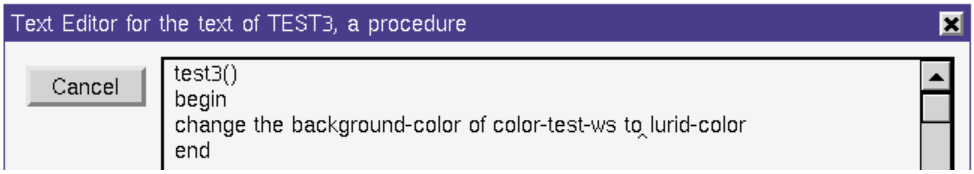
If there are other procedures on the call stack, G2 displays a menu that contains the error-generating procedure and its calling procedures. For example:



When you select one of the callers, G2 opens a text editor on the calling procedure, and places the cursor within the statement that calls another procedure on the stack:



When you select the error-generating procedure, G2 opens a text editor and places the cursor within the statement that caused the error:



If the call stack contains only the error-generating procedure, selecting go to source code immediately brings up the text editor on that procedure.

Obtaining Error-Location Information from the Error Object

If you shadow G2's error handler by defining your own error handler, the source-code location information is accessible from the error-source-line, error-source-column, and error-source-item attributes of the error object.

In this example procedure there is a call to an undefined procedure. The on error statement directs G2 to create an error object. The procedure simply posts the name of the error-causing procedure and the line and column positions of the error to the Message Board:

```
test-error-location()
procedure-error: class error;
begin
    call undefined-procedure()
end
on error (procedure-error)
    post "An error occurred
        in [the error-source-item of procedure-error]
        on line [the error-source-line of procedure-error]
        and in column [the error-source-column of procedure-error].";
    delete procedure-error
end
```

The error source line is indexed beginning at 1; and the error column position is indexed starting at 0. The line-column location is within the statement that generated the error, but G2 is not always able to pinpoint the exact syntax within the statement. The line and column values are -1 if G2 is unable to determine what procedure statement generated the stack error.

If you call a signal statement from within the on error statement, G2 replaces the information in the error-source-item, error-source-line, and error-source-column attributes with values that correspond to the signal statement. If an error object already contains values for these attributes, G2 overwrites them. If you want to retain this information, you must save it before signaling the error.

Caution Procedure code that handles errors must explicitly delete any error objects that are created to prevent them from accumulating and consuming memory.

Procedure Statements That Divert Error Location

Your procedures can contain code that causes G2 to point to statements that are not directly responsible for a stack error. This occurs in these cases:

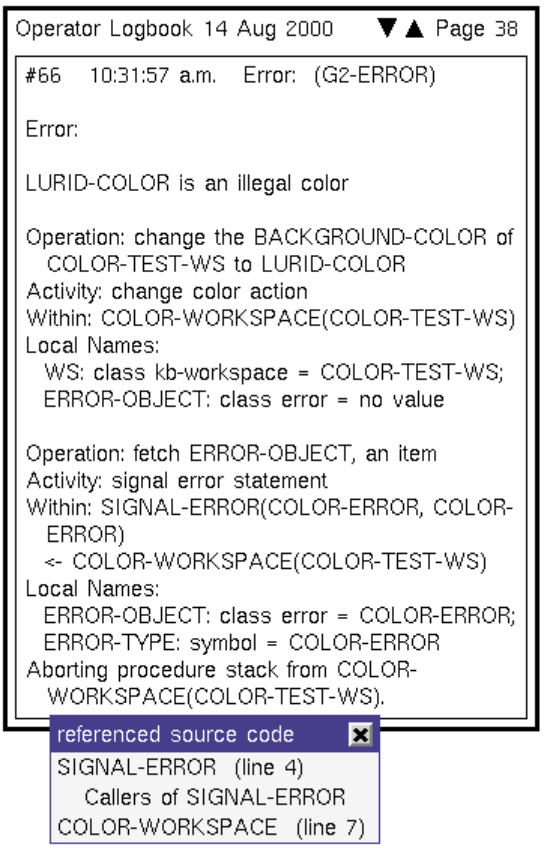
- When a procedure statement contains a function invocation and the function causes an error, the error position is within the procedure statement that invokes the function, not within the function itself.
- When a procedure statement contains a call to an inlined procedure and the inlined procedure causes an error, the error position is within the procedure statement that calls the inlined procedure, not within the inlined procedure itself.
- When a procedure has a signal statement, the error position is within the signal statement, not within the statement that caused the error. The error message that appears on the Operator Logbook contains information for both the error object and the signal.

The example below shows location diversion due to a signal statement. There are two procedures on the call stack. The procedure color-workspace has a change statement that references an illegal color symbol, and an on-error statement that creates an error object but does not signal the error. Instead, color-workspace calls the procedure signal-error which signals the error with the error object created by color-workspace.

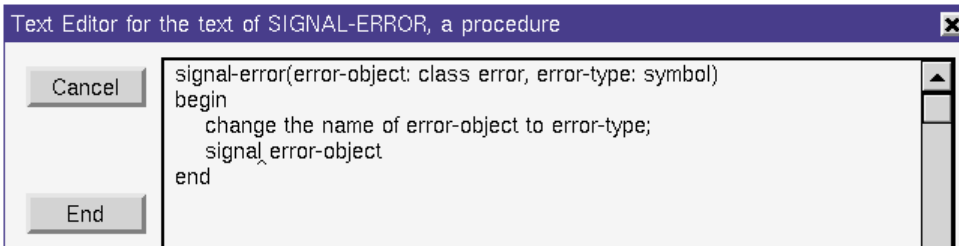
Here is the code for the two procedures:

```
color-workspace (ws: class kb-workspace)
error-object: class error;
begin
    change the background-color of ws to lurid-color
end
on error(error-object)
    call signal-error(error-object, the symbol color-error)
end

signal-error(error-object: class error, error-type: symbol)
begin
    change the name of error-object to error-type;
    signal error-object
end
```

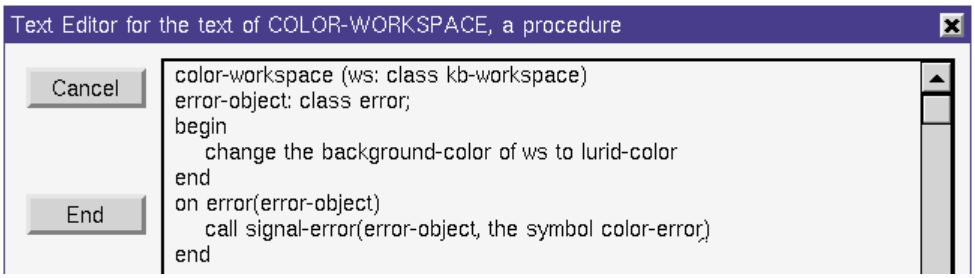
The double error message that appears from the signal statement is shown below. The error-message menu-choice go to source code has been selected:



Selecting signal-error brings that procedure into the text editor with the cursor within the signal statement:



Selecting color-workspace brings that procedure into the text editor with the cursor within the statement that calls signal-error.



Go-to-Source-Code Errors

Selecting the go to source code menu choice fails in these error situations:

- The location information in the Logbook error is not the same as the annotation location information in the procedure. This happens when you recompile a procedure after the procedure Logbook error is posted. G2 notifies you by posting this Logbook error message:

  This message is no longer valid because the procedure has been recompiled.

- The procedure is currently not editable because it is proprietary. G2 posts this message to the Logbook:

  Can't edit the procedure.

- This message is posted to the Logbook when G2 is unable to determine the source-code error location:

  Couldn't find source code mapping information.

Displaying Trace Messages

During debugging, you might want to display messages in the Operator Logbook that trace the execution of procedures, methods, rules, formulas, and display expressions. This can be helpful for determining what is being evaluated and in what order.

By default, G2 does not display trace messages.

To enable trace messages:

➔ Set the tracing-and-breakpoints-enabled? attribute in the Debugging Parameters system table to yes.

This attribute provides a convenient way of turning tracing and breakpoints on and off without re-editing the attributes that specify what items should be traced and at what level they should be traced.

To display trace messages:

➔ Edit the tracing-message-level attribute of the Debugging Parameters system table to specify tracing for the entire KB.

or

➔ Individually edit the tracing-and-breakpoints attribute of selected executable items to specify the tracing message level.

You can specify item-specific tracing by editing the tracing-and-breakpoints attribute of these classes of items:

- Procedures
- Methods
- Variables
- Readout-tables
- Meters
- Dials

For variables, G2 traces the formulas specified on the variable attribute tables; and for readout-tables, meters, and dials, G2 traces the display expressions.

You can specify the tracing message level to be one of these values:

- 0 (no trace messages)
- 1 (trace messages on entry and exit)
- 2 (trace messages at major steps)
- 3 (trace messages at every step)

Note The value of the tracing-message-level attribute of the Debugging Parameters system table overrides the values of the tracing-and-breakpoints attributes of individual items.

Specifying tracing for the entire KB causes G2 to display messages for all executing items. Unless your KB is small or has been selectively disabled, this might generate more information than you would like and make it difficult to perform other KB activities.

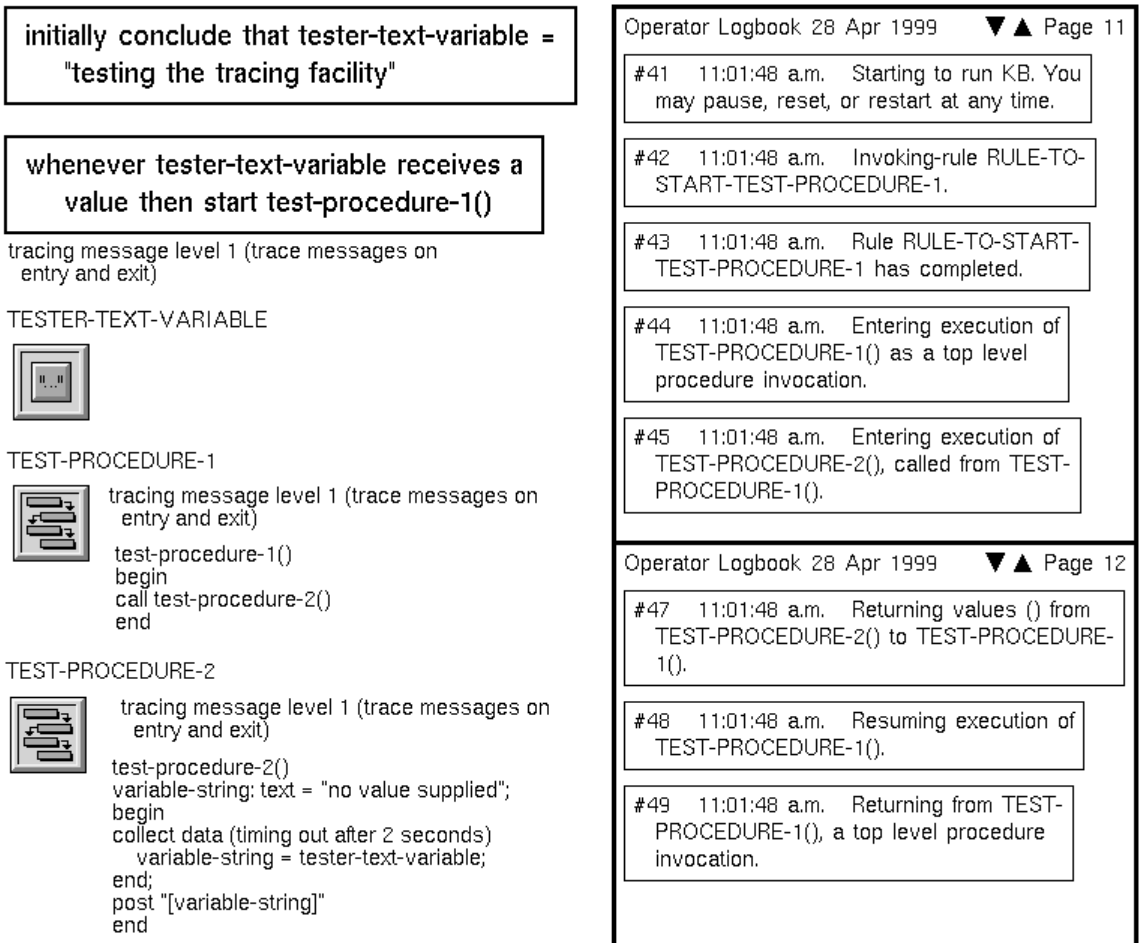
Pressing Control + z pauses the KB and the tracing messages, but when you resume, G2 resumes tracing message display until you change your tracing specification. Specifying no for the tracing-and-breakpoints-enabled? attribute of

the Debugging Parameters system table disables tracing and breakpoints regardless of related attribute specifications.

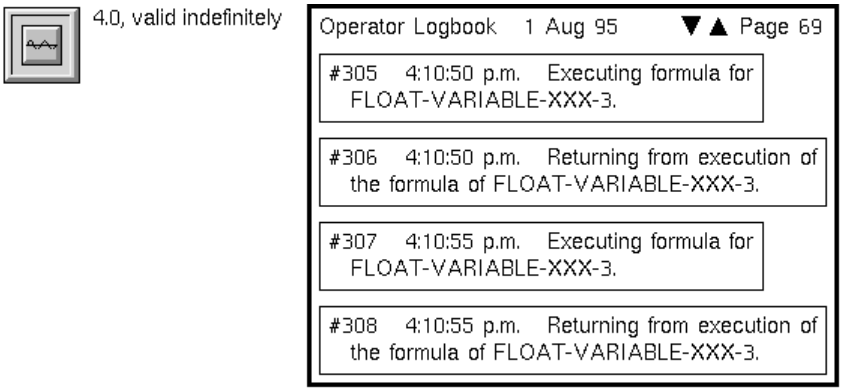
The following example shows the tracing messages that appear on the Operator Logbook for a KB in which Level 1 tracing has been specified for three individual items:

- rule-to-start-test-procedure-1, a whenever rule
- test-procedure-1, a procedure that calls test-procedure-2
- test-procedure-2

When tracing is specified for the entire KB, tracing messages would also be displayed for the initially rule and for any other executable items in the KB:



The next example shows the tracing messages G2 displays when a variable receives a value twice from the formula defined for its formula attribute. These tracing messages are displayed when the variable tracing level is Level 1 or 2.



Saving Tracing Data to a File

You can save the tracing messages that are displayed in the Operator Logbook to a file.

To enable writing tracing data to a file:

➔ In the Miscellaneous Parameters system table, set the enable-explanation-controls attribute to yes.

G2 does not open a tracing file, but it now has the option to do so.

To start writing tracing data to a file:

➔ In the Debugging Parameters system table, enter a file name as a text value in the tracing-file attribute.

If G2 is running and tracing is enabled, G2 immediately opens the specified file and begins writing tracing information to it. Any existing file having the specified name is overwritten.

Writing continues as long as G2 runs. Once writing is underway, you can perform the following actions to effect output to the tracing file:

This action...	Has this effect...
Change the value of the tracing-file attribute to none	The trace file closes.
Change the value of the tracing-file attribute to a different filename	The trace file closes, a new file with the new name opens, and writing begins to the new file.

This action...	Has this effect...
Pause G2	Writing stops, but the trace file remains open.
Resume G2	Writing to the trace file continues from the point where G2 paused.
Reset G2	Writing to the trace file stops and the file closes. To preserve the data already in the trace file, change the name in the tracing-file attribute before starting G2 again.
Restart G2	The trace file closes and is immediately overwritten by a new empty file with the same name. Writing begins to the new file.

To view the trace file:

1 Perform one of the following operations to close the trace file:

- Turn tracing off by specifying no for the tracing-and-breakpoints-enabled? attribute.
- Specify none for the tracing-file attribute.
- Specify a different filename for the tracing-file attribute.
- Reset G2.

2 Open the trace file for viewing.

Note You can also write the text of all the messages that are sent to the Logbook to a file. See Writing Logbook Messages to a Log File of this chapter for information on this related facility.

Specifying Breakpoints and Tracing

By setting **breakpoints,** you can both display trace messages and pause the KB during code execution. This feature allows you to step through the execution of procedures, methods, formulas, and display expressions, pausing between steps so you can view trace messages and examine other changes in the state of your running KB.

By default, G2 does not set breakpoints.

To enable breakpoints and tracing:

➔ Set the tracing-and-breakpoints-enabled? attribute in the Debugging Parameters system table to yes.

This enables breakpoint and display functionality, but code execution does not pause until you specify breakpoints for the KB or for individual items, or until you use the halt action.

To specify breakpoints and tracing:

➔ Edit the breakpoint-level attribute in the Debugging Parameters system table to specify breakpoints for the entire KB.

or

➔ Edit the tracing-and-breakpoints attribute of selected items to specify the breakpoint level.

or

➔ Use the halt action to halt the execution at a particular point in the code for an item.

Specifying breakpoints for the entire KB causes G2 to pause for all executing procedures, rules, formulas, and display expressions. You can specify item-specific breakpoints by editing the tracing-and-breakpoints attribute of these classes of items:

- Procedures
- Methods
- Variables
- Readout-tables
- Meters
- Dials

For variables, the breakpoints apply to the formulas specified on the variable attribute tables. For readout-tables, meters, and dials, the breakpoints apply to the display expressions.

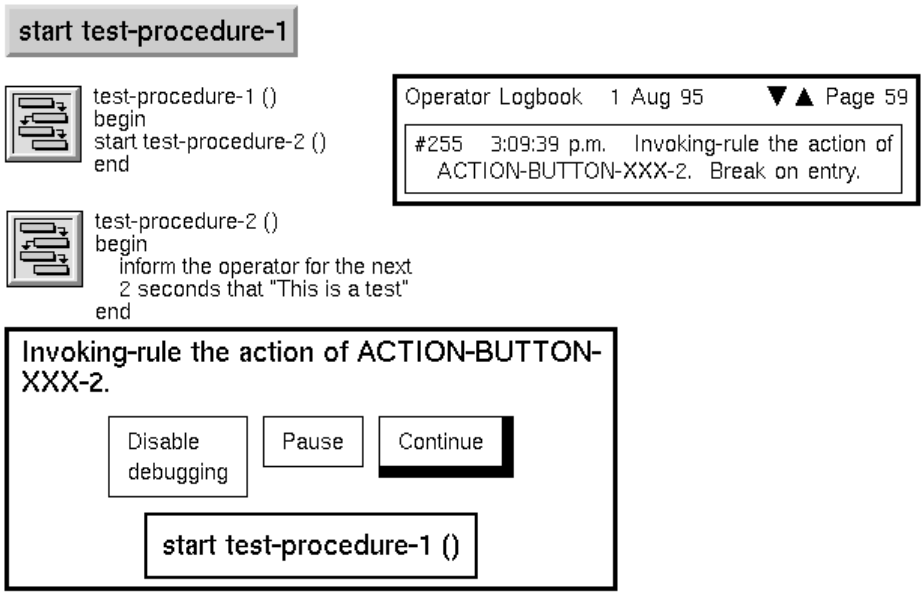
You can specify the breakpoint level to be one of these values:

- 0 (no breakpoints)
- 1 (breakpoints on entry and exit)
- 2 (breakpoints at major steps)
- 3 (breakpoints at every step)

For example, suppose you have an action button that executes a procedure, which in turn calls another procedure. Setting the breakpoint-level of the Debugging

Parameters system table to level 1 (breakpoints on entry and exit) causes G2 to display a trace message and pause each time the action button and the two procedures begin and end, as well as for all other executing items.

This example shows the initial display that appears when G2 begins executing the action button:



At this point, you can click:

- The Pause button to pause the KB and remove the breakpoint.

  If the show-procedure invocation-hierarchy-at-pause-from-breakpoint attribute of the Debugging Parameters system table is yes, clicking the Pause button also automatically executes the Inspect command show on a workspace the procedure invocation hierarchy. The resulting workspace is slightly different than it is when invoked directly from Inspect. The currently running procedures appear at the left of the workspace, so the procedure that contains the most recent breakpoint appears at the bottom left.

- The Continue button to create a trace message and breakpoint for the next step.

- The Disable debugging button to turn tracing and breakpoints off entirely.

To continue running after pausing, choose Continue from breakpoint from the Main Menu. G2 displays a trace message and pauses at the next step in the execution.

Using Dynamic Breakpoints

You can set breakpoints dynamically in procedure code in the client through the Windows text editor or the standard debugger. You can also set and remove dynamic breakpoints in the server through a hidden attribute on the procedure called dynamic-breakpoints.

When you execute the procedure with dynamic breakpoints set, G2 performs the same operation as it does when the halt action is executed.

Dynamic breakpoints are saved in and loaded from a KB.

Note If procedure A inlines procedure B, procedure A does not inherit procedure B's dynamic breakpoints.

For information on using the Windows text editor, see Editing Text in Using Telewindows in the *Telewindows User's Guide*.

Setting Dynamic Breakpoints in the Client

To set a dynamic breakpoint in the client, edit the procedure to display the standard text editor, then click in the column to the right of the line number at which you want to set a breakpoint for a line of code. You can also set a dynamic breakpoint directly from within the standard debugger. The dynamic breakpoint appears as a filled circle next to the line of code.

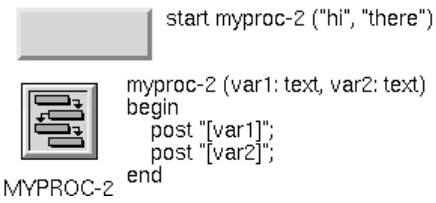
To disable a dynamic breakpoint, click the filled circle. The dynamic breakpoint appears as an open circle. To remove the breakpoint, click the open circle again. Disabling a breakpoint behaves as if the breakpoint were not there.

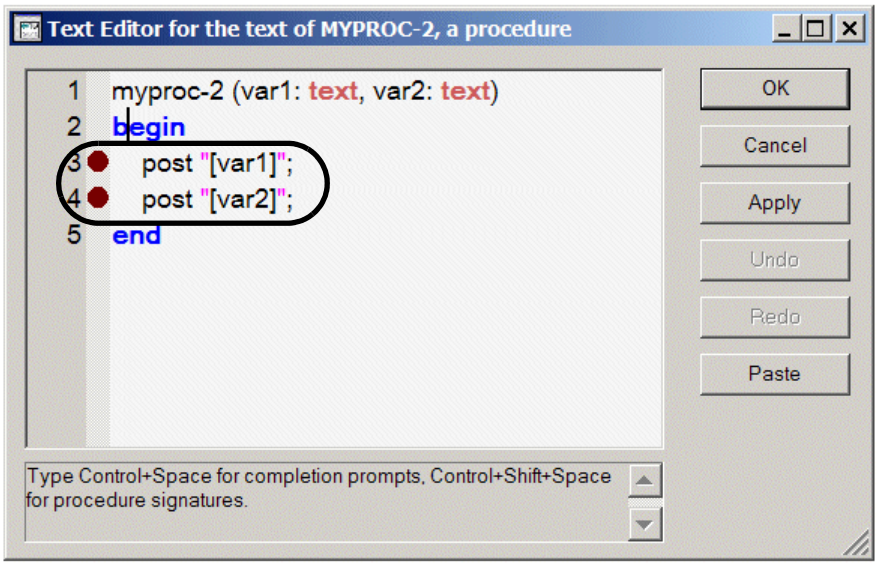
Before you can edit dynamic breakpoints in the standard text editor, you must apply all pending edits to compile the procedure.

You can only set dynamic breakpoints for valid lines of code, such as procedure statements. You cannot set dynamic breakpoints for invalid lines of code, such as the procedure name, local variables, begin/end statements, comments, and blank lines. The text editor beeps if you attempt to place a dynamic breakpoint next to an invalid line of code.

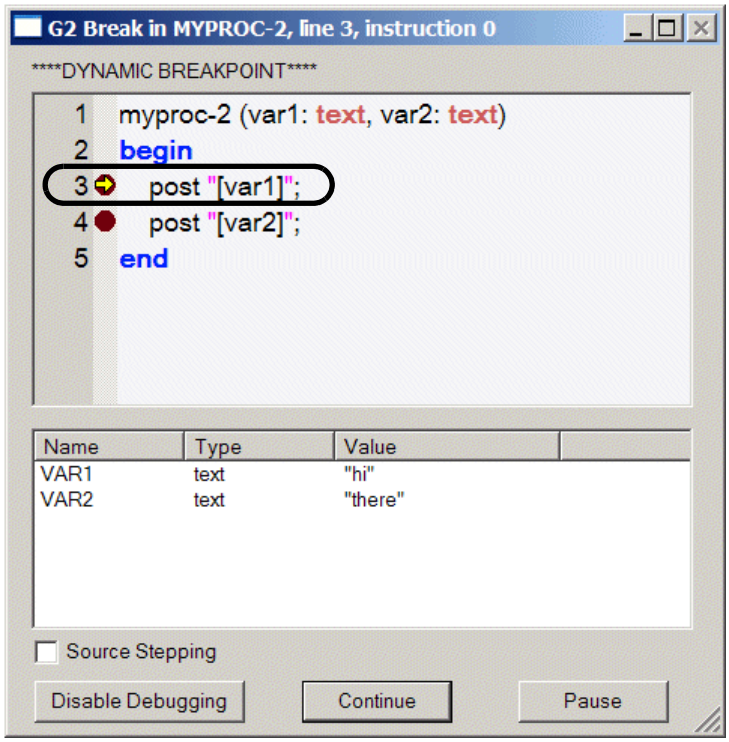
You cannot set dynamic breakpoints on the first line of a for statement.

For example, this procedure sets dynamic breakpoints on each line in the body of the procedure:





When you execute the procedure, G2 displays the debugger and places an arrow next to the line that is about to execute at which a breakpoint has been set:



At this point, you can set and remove dynamic breakpoints, as needed. Click in a column to set a breakpoint or click an existing breakpoint once to disable it or twice to remove it, then click Continue to continue executing the procedure. You can also enable Source Stepping.

Setting Dynamic Breakpoints in the Server

Procedures define a hidden attribute called dynamic-breakpoints, which has the following signature:

```
sequence
    (structure (line-number: integer,
                is-enabled: truth-value,
                is-valid: truth-value)
    . . . )
```

where:

- line-number is the line number of the source code on which to set a dynamic breakpoint.
- is-enabled determines whether dynamic breakpoints are enabled or disabled for the specified line of source code.

- is-valid indicates whether the specified line of source code is a valid line on which to set a breakpoint. A line of source code is invalid if the specified line does not exist or if it is a line of source code on which you cannot set a breakpoint, such as the procedure name and begin/end statements. Invalid dynamic breakpoints are ignored at runtime. This attribute is read-only and is ignored when setting via the hidden attribute.

To set dynamic breakpoints in the server, you can conclude a value for the dynamic-breakpoints hidden attribute. All attributes of the structure except is-valid are required when specifying the sequence of structures. An empty sequence means the procedure has no dynamic breakpoints.

For example, the value of the dynamic-breakpoints hidden attribute for the previous procedure looks like this, where dynamic breakpoints are set for lines 3 and 4, where is-enabled and is-valid are both true:

dynamic breakpoints	sequence (structure (line-number: 4, is-enabled: true, is-valid: true), structure (line-number: 3, is-enabled: true, is-valid: true))

When an invalid dynamic-breakpoint is added, G2 automatically sets is-enabled to false and is-valid to false for the corresponding line of source code. G2 ignores invalid breakpoints at runtime. A procedure with invalid dynamic breakpoints has notes indicating invalid dynamic breakpoints.

Setting and removing dynamic breakpoints via the hidden attribute does not recompile the procedure.

Note You can create invalid breakpoints through the text editor as a side-effect of editing a procedure with existing breakpoints, which are subsequently made invalid by editing the procedure.

Stepping Through Procedure Source Code

G2 allows you to single-step through procedure source code. To use this feature, tracing-and-breakpoints-enabled? must be set to yes in the Debugging Parameters system table.

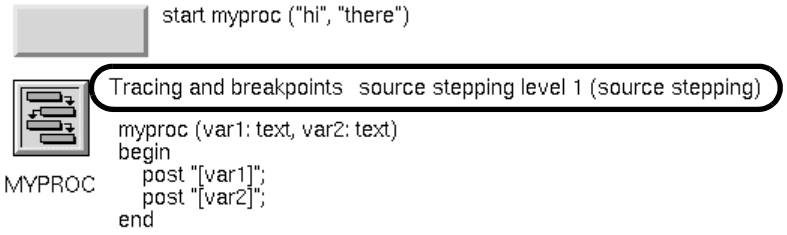
You enable this feature in one of two ways:

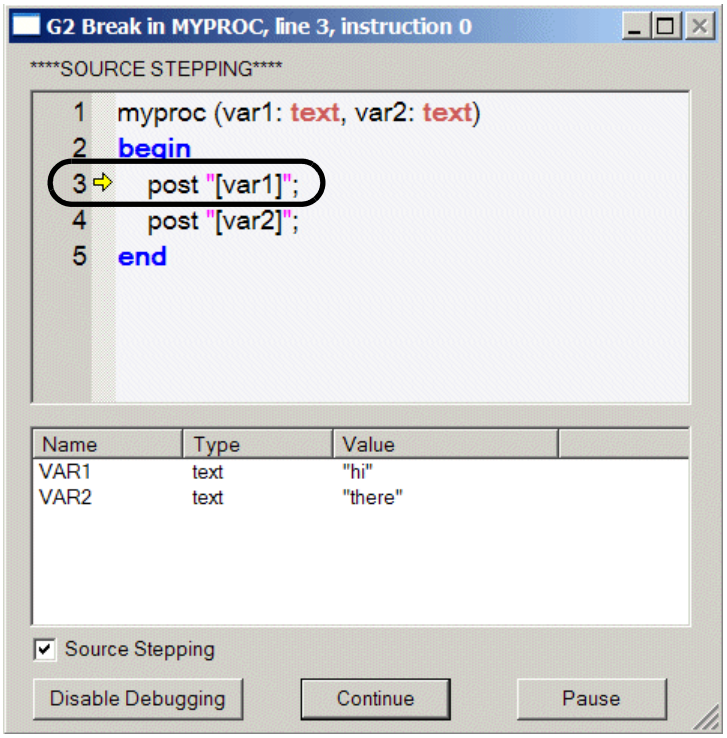
- To enable it globally, set the source-stepping-level attribute in the Debugging Parameters system table to 1 (source stepping). By default, the value is 0 (no source stepping), which means do not allow single-stepping.
- To enable it for individual procedures, set the tracing-and-breakpoint attribute for an individual procedure to source stepping level 1 (source stepping). The other option is source stepping level 0 (no source stepping).

When G2 is single-stepping through source code, before the next line of source code is executed, it performs a similar action as when a halt action is executed. In the client, the Windows debugger appears, and in the server, a dialog appears that shows the source code around the line of source code G2 is about to execute, the line numbers, the contents of the stack, and the local variable bindings.

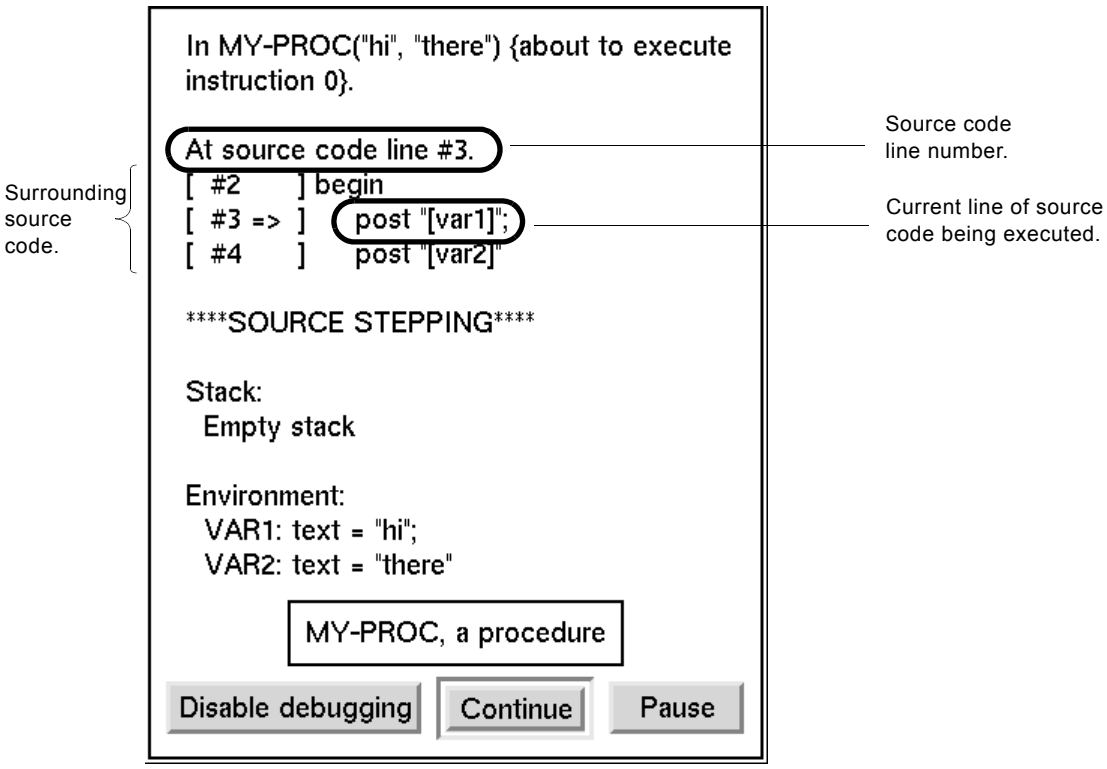
Note When single-stepping through source code, if procedure A inlines procedure B, procedure A does not step into procedure B, because the source code of procedure B is not in procedure A. Instead, procedure A steps over the call to procedure B.

Here is the result of setting tracing-and-breakpoints to source stepping level 1 (source stepping), then executing the procedure in the client to display the debugger. Notice that the Source Stepping option is enabled at the bottom of the dialog.





In the server, you see the following dialog, which shows the line of source code being executed and one line of source code above and below the line currently being executing.



Stepping Through Procedure Source Code

G2 allows you to single-step through procedure source code. When G2 is single-stepping through source code, before the next line of source code is executed, it performs a similar action as when a halt action is executed. In the client, the standard Windows debugger appears, and in the server, a dialog appears that shows the source code around the line of source code G2 is about to execute, the line numbers, the contents of the stack, and the local variable bindings.

For information on using the standard debugger in the client, see the *Telewindows User's Guide*.

To enable single-stepping through source code:

➔ Set the tracing-and-breakpoints-enabled? attribute in the Debugging Parameters system table to yes.

This enables breakpoint and display functionality, but code execution does not pause until you single step through code.

To specify single stepping:

➔ Edit the source-stepping-level attribute in the Debugging Parameters system table to specify breakpoints for the entire KB.

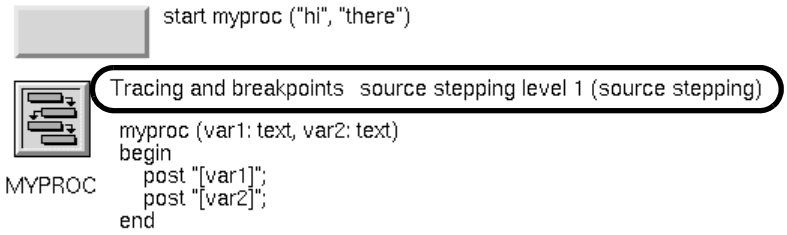
or

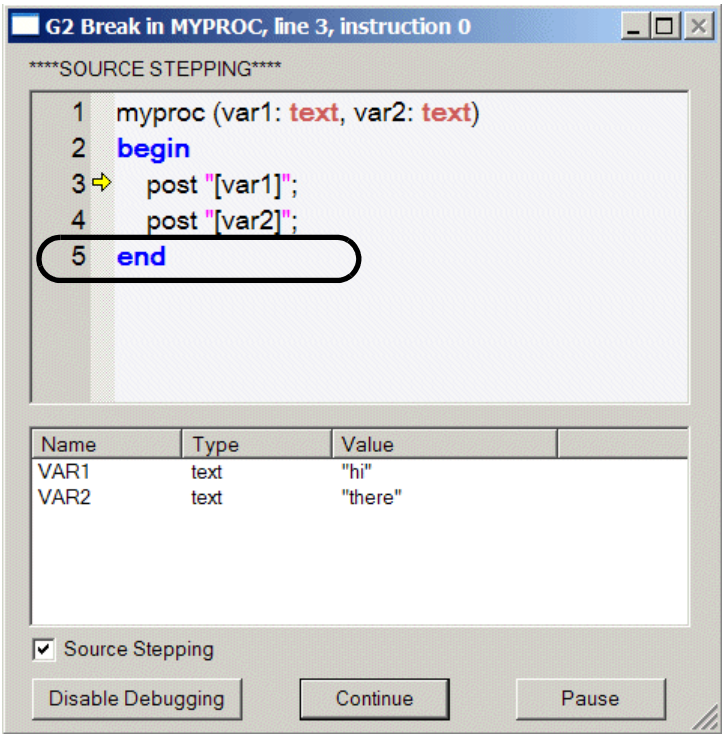
➔ Edit the tracing-and-breakpoints attribute of selected items to specify the source stepping level.

You can specify the source stepping level to be one of these values:

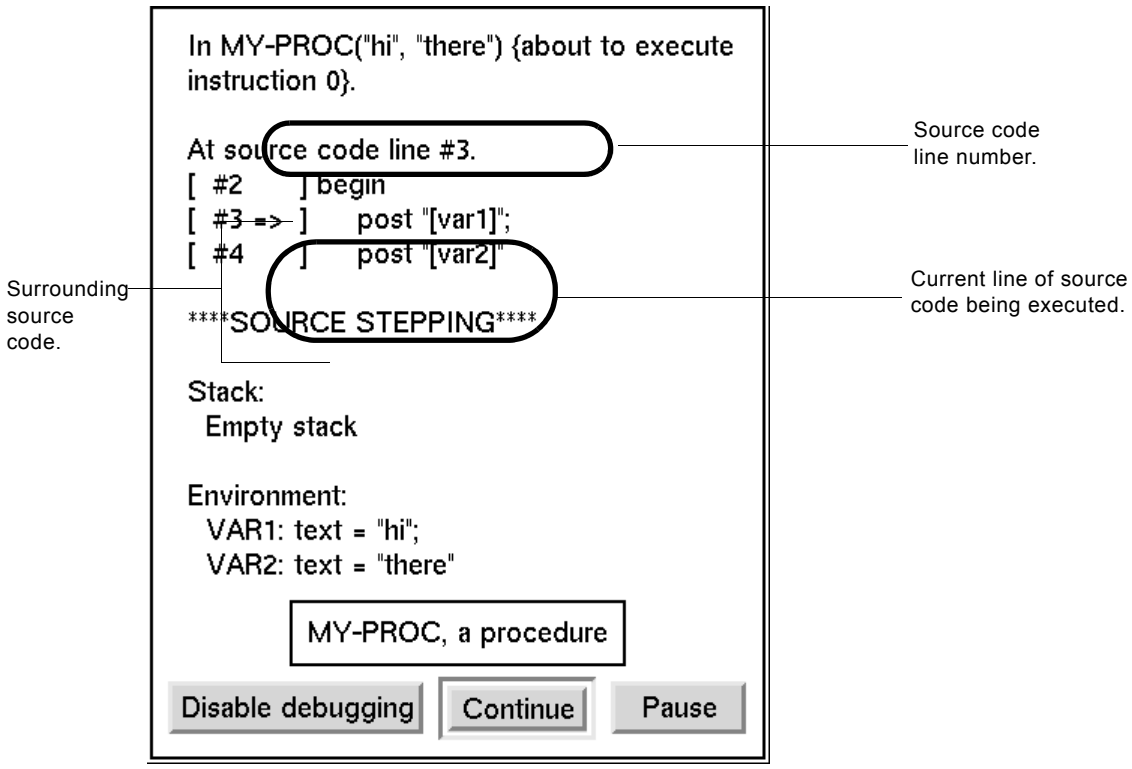
- 0 (no source stepping)
- 1 (source stepping)

Here is the result of setting tracing-and-breakpoints to source stepping level 1 (source stepping), then executing the procedure in the client to display the standard debugger in the Telewindows client:





In the server, you see the following dialog, which shows the line of source code being executed and one line of source code above and below the line currently being executing.



Removing Tracing and Breakpoints

You have three options for removing tracing and breakpoints once they are enabled.

To remove tracing and breakpoints:

➔ Choose Main Menu > Run Options > Remove Tracing and Breakpoints.

This sets the tracing-and-breakpoints attribute of every item to default, and sets the tracing-and-breakpoints-enabled?, tracing-message-level, and breakpoint-level attributes in the Debugging Parameters system table to their default values.

or

➔ Select the disable debugging button from a breakpoint.

This sets the tracing-and-breakpoints-enabled? attribute in the Debugging Parameters system table to no. It does not effect the value of the tracing-and-breakpoints attribute of individual items.

or

➔ Change the value of the tracing-and-breakpoints-enabled? attribute to no in the Debugging Parameters system table.

This attribute has no effect on the value of the tracing-and-breakpoints attributes of individual items.

Showing Disassembled Code

G2 translates procedures, methods, rules, and a few other classes of items into an internal format called **byte code**. Although the format of byte code is undocumented and may change in the future, you can sometimes use the byte-code representation of a procedure to locate an error or a breakpoint in the source code.

The disassembler-enabled? attribute of the Debugging Parameters system table controls whether disassembled code is ever displayed.

When the disassemble-enabled? attribute is yes, three changes occur to the G2 environment that facilitate debugging:

- The describe menu choice for a procedure, method, or rule shows the corresponding byte-code representation.
- G2 error messages indicate the byte-code instruction that was running when the error was generated.
- The Inspect command show on a workspace the procedure invocation hierarchy indicates the byte-code instruction that is running for every procedure invocation.

Obtaining Information from Abort Workspaces

If G2 aborts during execution, G2 displays an abort workspace, that contains information about the state of the program at the time that G2 stopped executing.

Before contacting Customer Support, make a note of all information in the abort workspace so you can provide Gensym's Customer Support staff with the most complete information.

This information also appears in the window such as the UNIX console window from which the G2 process was launched. This makes it easier to place the information into a file, which you can send to Gensym via electronic mail (e-mail) or facsimile (FAX).

Note The abort workspace recommends that you save the current KB to a KB file. Please save the current KB to a *new KB file*, not to the same KB file from which you loaded the current KB.

Writing Logbook Messages to a Log File

You can write warning and trace messages to a log file by using the Log File Parameters system table.

To enable the log file:

1. Select Main Menu > System Tables > Log File Parameters.
2. Set the log-file-enabled? attribute to yes.

G2 automatically writes all Logbook messages to the log file.

By default, G2 writes the log file in the default directory. You can specify a different directory by editing the directory-for-log-files attribute in the Log File Parameters system table.

G2 adds a prefix to all log filenames; the default is g2-log-.

For more information on setting log file parameters, see Log File Parameters.

52

Explanation Facilities

Describes the facilities that collect and display data about rules and formulas and the objects they reference.

Introduction

You can use G2's explanation facilities to:

- Statically display one level of forward and backward chaining for a variable.
- Dynamically display:
  - All invocations of backward-chaining rules for a variable.
  - All invocations of rules for an object that contain a generic reference to that object.
  - All invocations of a particular rule.
- Cache explanation data for variables, parameters, and rules and create explanation items that display the data on explanation trees.

Note The explanation facilities are solely for use during KB development and debugging, and are therefore restricted from appearing within a deployed KB. Gensym reserves the right to limit access to any or all explanation facilities to a development license in future releases.

For information on programmatic access to the explanation facilities, see the description of g2-get-explanation-hierarchy in Get Hierarchy Operations in the *G2 System Procedures Reference Manual*.

Example KB in the Demos Directory

There are examples of all explanation facilities in explnfac.kb in the demos directory that is supplied with G2. This KB contains detailed instructions and procedures for demonstrating and experimenting with the explanation facilities.

Enabling the Explanation Facilities

The explanation facilities are not available until you explicitly enable them, either programmatically or interactively.

To interactively enable the explanation facilities:

1 Display the Miscellaneous Parameters system table:

Main Menu > System Tables > Miscellaneous Parameters

2 Specify yes for the Enable-explanation-controls attribute.

To programmatically enable the explanation facilities:

➔ Use the conclude action:

```
conclude that the enable-explanation-controls of miscellaneous-parameters
is true
```

When you set the Enable-explanation-controls attribute to yes, the following changes occur in your KB:

- These choices appear on the Miscellany menu:
  - Turn On All Explanation Caching
  - Turn Off All Explanation Caching
- This option appears in the Options attribute of every variable, parameter, and rule table:

  [do not] cache data for explanation

- The choices below appear on variable, parameter, object, and rule popup menus. They do not appear on the subtables of objects that are the values of attributes. However, you can programmatically initiate displays for subobjects, and for items on workspaces, by calling the system procedure, g2-system-command which is described in the *G2 System Procedures Reference Manual*.

Menu Choice	Variable	Parameter	Rule	Object
describe chaining	✓			
dynamic backward chaining	✓			
dynamic generic rule display	✓	✓		✓
dynamic rule invocation display			✓	

- There are improvements to the facility that saves tracing data to a file. See Saving Tracing Data to a File for more information.
- Separate tracing messages for rule invocations are collected into a single message.

The rest of this chapter describes the explanation facilities. This chapter assumes that you have specified yes for the Enable-explanation-controls attribute of the Miscellaneous Parameters system table.

This chapter uses interactive menu-choice examples for initiating explanation displays. You can also initiate the displays programmatically by calling the system procedure, g2-system-command, which is fully described in the *G2 System Procedures Reference Manual*. An example call to this procedure is:

```
call g2-system-command(the symbol dynamic-rule-invocation-display,
    my-window, probe-rule, the symbol none)
```

Displaying Current Chaining and Rule Invocation

You can use the explanation facilities to obtain static displays and displays that dynamically update while G2 is executing.

For static displays, G2 selects the rules and formulas that would update a variable when G2 is executing. It bases its selection on such information as the priorities that rules and formulas have for updating a variable and on the chaining options specified for rules. See Obtaining Requested Values for update priorities, and Invoking Rules for information on using chaining options.

Note The chaining display appears on a temporary workspace that you should delete when you no longer wish to view the display.

Statically Displaying One-Level of Chaining for a Variable

You can display a static view of one level of forward and backward chaining for a variable while G2 is running, paused, or reset.

To display a static view of one level of chaining:

1. Click any variable that is on a workspace to display its item menu.
2. Choose describe chaining.

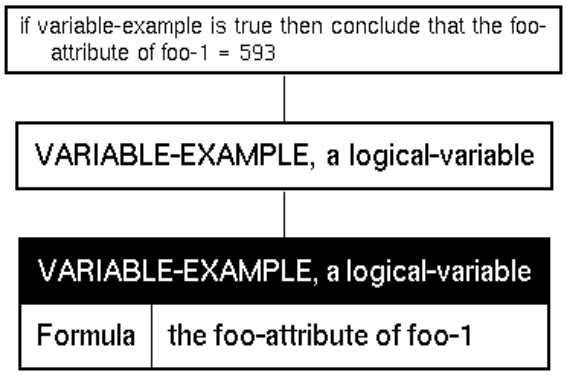
Static displays include:

- A representation of the variable.
- Representations that appear above the variable are the specific and generic rules that reference the variable and may execute through forward chaining when the variable receives a value.
- Representations that appear below the variable are:
  - The specific formula of the variable that will execute when the variable needs a value. A variable formula takes precedence over any rules for supplying the value for a variable.

    or

  - The specific and generic rules that backward chaining will invoke when the variable needs a value. When the variable has a specific formula, no backward chaining rules will be displayed because the variable will receive its value from the specific formula.

This example shows a specific forward chaining rule and a specific formula which is the value of the formula attribute of the variable:



Dynamically Displaying Backward Chaining for a Variable

You can display a dynamic view of all backward chaining rules that are being invoked for a variable.

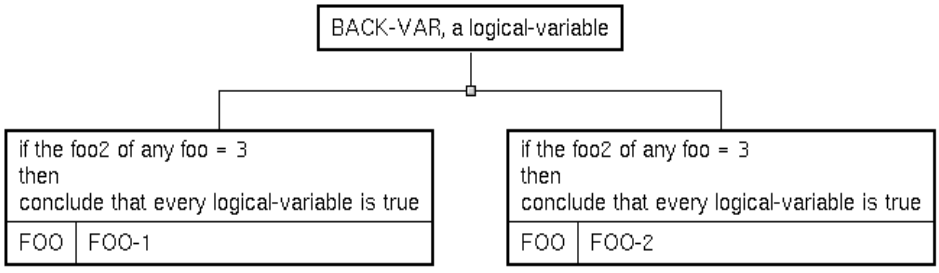
To display a dynamic view of backward chaining rules for a variable:

1 Click any variable to display its item menu.

2 Choose dynamic backward chaining.

The display appears on a temporary workspace and updates dynamically as the KB executes. To cancel the display, delete the temporary workspace.

This example shows two invocations of the same rule, one for each instance of the class foo that satisfied the rule antecedent condition:



Dynamically Displaying Generic Rule Invocations for an Object

You can display a dynamic view of all invoked rules that reference an object generically.

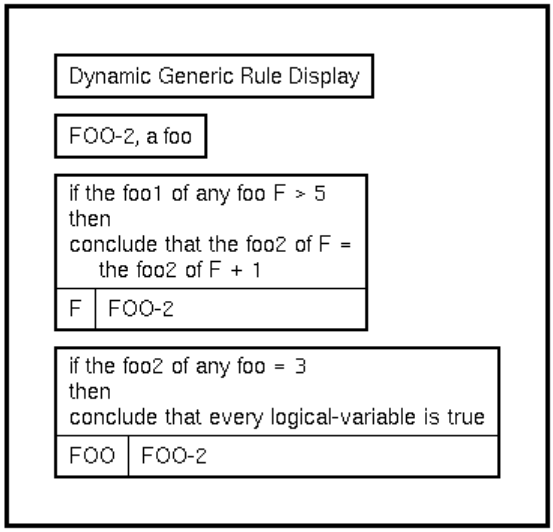
To display generic rule invocations:

1 Click any object to display its item menu.

2 Choose dynamic generic rule display.

The display appears on a temporary workspace and updates dynamically during KB execution. To cancel the display, delete the temporary workspace.

This example shows the generic rule display for an object that has been referenced generically in two rule invocations:



Dynamically Displaying the Invocations of a Rule

You can display a dynamic view of all active invocations of a rule.

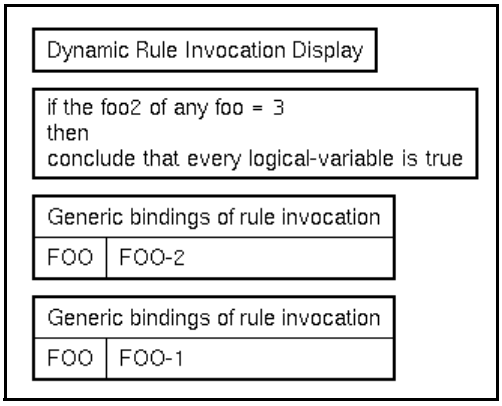
To display current rule invocations:

1 Click any rule to display its item menu.

2 Choose dynamic rule invocation display.

The display appears on a temporary workspace that updates dynamically during KB execution. To cancel the display, delete the temporary workspace.

For example, this display shows that there were two invocations of the rule, one for foo-1and one for foo-2:



Delaying Dynamic Display Updates

All G2 processing halts during the interval between dynamic display updates. To avoid slowing G2 unnecessarily, the default interval is 200 milliseconds, which is typically not enough time to examine the data.

You can change the default to be any integer from 0 to 60,000 milliseconds. Setting this attribute delays the update of all explanation facility dynamic displays for the specified number of milliseconds before allowing processing to continue. No G2 processing of any kind occurs during the delay interval.

To set a dynamic display update delay:

1 Choose:

Main Menu > System Tables > Debugging Parameters

2 In the Dynamic-display-delay-in-milliseconds attribute, enter an integer between 0 and 60,000.

Note Pausing or resetting the KB discontinues dynamic-display updating, but it does not cancel it. Restarting the KB causes dynamic displays to begin updating again. To cancel dynamic displays, delete the temporary workspace containing the display.

Displaying Explanation Trees of Cached Chaining and Rule Invocation Knowledge

The explanation facilities let you cache knowledge about how a variable or parameter received its latest value, and what caused the invocation of a generic rule. You can then create an instance of the explanation class to display the cached data for that variable or parameter.

The display is in the form of an explanation tree that appears on the subworkspace of an explanation item. The display is a graphical depiction of the source of the current value of the variable or parameter, including any relevant rule invocations. For a generic rule, an explanation shows which variables were referenced during its execution.

Caching Explanation Data

There are two ways to start and stop caching explanation data in items:

- Individually in specific items
- Globally for the entire KB

Data Caching for Specific Items

You set a variable, parameter, or rule to cache or not cache explanation data through specifying one of two options in the Options attribute:

Cache Data for Explanation

Do Not Cache Data for Explanation

These options are available on variables, parameters, and rules when the Enable-explanation-controls attribute of the Miscellaneous Parameters system table is set to yes. By default, items do not cache explanation data.

Choosing the Cache Data for Explanation option directs the variable, parameter, or rule to begin caching knowledge of its value or its execution, and to display the current state of that knowledge at the time you create an EXPLANATION item for a specific variable or parameter. By creating multiple explanations for a single variable or parameter, you can capture cached data about the variable as its values change over time.

To set data caching for a variable, parameter, or rule:

➔ In the options attribute of a variable, parameter, or rule, specify:

Cache Data for Explanation

For a . . .	Setting this option describes . . .
variable	Whatever formula, rule, or data server provided the latest value for the variable.
parameter	Whatever rule or data server provided the latest value for the parameter.
rule	Which variables were used in generic rule references.

To stop data caching for a variable, parameter, or rule:

➔ In the Options attribute of the item specify:

Do Not Cache Data for Explanation

Data Caching for the Entire KB

You can globally set the cache data for explanation/do not cache data for explanation options on the Options attribute for all variables, parameters, and rules in the entire KB.

To start data caching for all variables, parameters, and rules in your KB:

➔ Choose:

Main Menu > Miscellany > Turn On All Explanation Caching

To stop data caching for all variables, parameters, and rules in your KB:

➔ Choose:

Main Menu > Miscellany > Turn Off All Explanation Caching

Creating Explanation Items

When explanation data is being cached, you can create EXPLANATION items to view the cached data for a variable or parameter.

To create an explanation item:

create an explanation [*local-name]* for {*variable* | *parameter*} [; transfer *local-name* to *kb-workspace*]

This action creates a transient EXPLANATION item that holds the cached explanation data of a variable or parameter. The information contained in an explanation is static. By creating multiple explanations for a single variable or parameter, you can capture the cached data as its values change over time.

Displaying Explanations

The graphical depiction that explains the value of a variable or parameter at the time the create an explanation action was performed appears on the subworkspace of the explanation item.

To display an explanation tree, do one of the following:

➔ Enter a command such as:

show the subworkspace of *explanation*

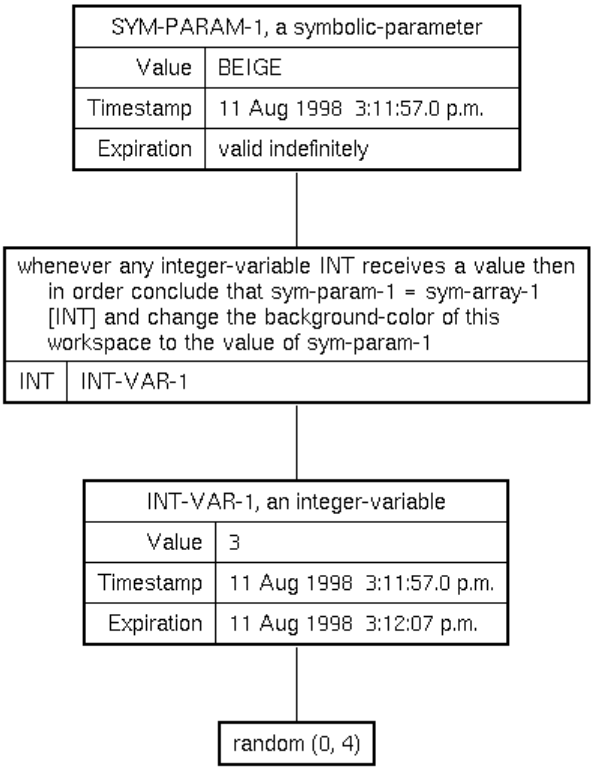
or

➔ Choose go to subworkspace on the explanation item table.

Understanding Explanation Trees

Each variable and parameter in the explanation tree is shown with the value, value expiration period, and value collection time it had when its explanation was last cached before the creation of the explanation item.

For example, this diagram shows an explanation for the sym-param-1 parameter:



The explanation is an inverted tree which displays a representation of the create-action parameter or variable argument at its root (top). In this example, the root is the parameter sym-param-1; and immediately below sym-param-1, and linked to it, is the rule that concluded its value.

If sym-param-1 were the only item caching explanation data, the tree would contain only two items: sym-param-1 as its root and the rule as a leaf. There would be no items linked to the rule from below.

In this example, the rule does cache explanation data; and because it is a generic rule, the variables that were evaluated during its execution appear below, and linked with, the rule. Only one variable, int-var-1, was evaluated in this example; and because int-var-1 also caches data, the formula that supplied its value is linked to int-var-1 from below.

The caching of information for this example could have been started by either setting caching for the entire KB, or by individually setting caching for the parameter, the rule, and the variable.

These are the items that can be the leaves of an explanation tree:

- Variables that do not cache explanation data.
- Rules that do not cache explanation data, or which are specific, rather than generic.
- Specific formulas that determine values for a variable or parameter

Deleting Explanations

Since explanations are items that can accumulate and consume memory, you should delete them when they are no longer useful.

Deleting an explanation also deletes its subworkspace and the explanation tree on the subworkspace.

53

Profiling and KB Performance

Describes techniques for evaluating and improving KB performance.

Introduction

The study of techniques for improving your knowledge base's performance is an advanced topic. The information presented in this chapter is directed toward experienced G2 users.

You can use G2's profiling system procedures to gather information about your KB's execution. After identifying which portions of your KB can benefit from further optimization, you can apply compilation configurations to the appropriate definition items and executable items, so that G2 compiles the statements and actions in those items more efficiently.

Profiling the Execution of Your KB

G2 can collect profile data about the executable items in your KB. **Profile data** show which parts of your KB executed during a particular time interval and reveal how long each part executed.

G2 collects profile data about these executable items:

- Procedures
- Methods
- Rules
- Functions
- Formulas
- Display items: meters, readout-tables, trend-charts, freeform tables, and so on

The profile data that G2 collects include three kinds of information:

- The real time that passed during the profiling period.
- Execution timings and number of invocations for each item that was invoked during the profiling period.
- Number of times performed for each invocable action or statement in each invoked item.

Techniques for Profiling

G2 provides various essential capabilities that support profiling, as described in this chapter. These do not in themselves constitute a profiling capability: they are the nuts and bolts from which such a capability can be constructed. If you prefer, you can write your own profiler and give it any properties and user interface you need, using the techniques described in this chapter in the *G2 GUIDE User's Guide*.

The G2 software includes three KBs that supply higher-level profiling capabilities:

- **`profile-demo.kb`** in the G2 `samples` directory.
- **`profiler.kb`** in the G2 `utils` directory.
- **`profroot.kb`** in the G2 `utils` directory.

The first is an example of a simple profiler; it can be useful for studying the essential techniques for designing profilers and as a basis to extend and customize. The second is a complete profiling utility; it provides sophisticated profiling capabilities, but is more difficult to extend and customize due to its greater specialization. The third provides a front-end to `profiler.kb`.

Both the `profile-demo.kb` sample and the `profroot.kb` utility are self-documenting. To use either of them, merge it into the KB that you want to profile, then follow the directions in the KB to start and stop collecting profile data, and display reports of the KB's activity.

In most cases, the `profiler.kb` utility provides all the profiling capabilities that are needed. However, even if you do not intend to create your own profiler or

extend one that is provided with G2, you should skim this chapter to gain a general understanding what a profiler does, and how and why it does it.

Understanding the Profiling Process

G2 collects profile data in memory. G2 retains profile data in memory until you direct it to clear all profile data.

You can direct G2 to start collecting profile data, to stop collecting data, and to start and stop again as many times as necessary. G2 always adds new profile data to any profile data already collected. By starting and stopping profiling more than once as your KB runs, you can measure different parts of the KB at various times without losing data.

You direct G2 to create a system-profile-information item that contains a copy of the profile data collected so far.

Identifying Resource Requirements for Profiling

While collecting profile data does not significantly impact KB performance, profiling a large KB can consume large amounts of memory. Always have sufficient memory available for G2's use before you begin collecting profile data.

Using System Procedures for Profiling

To collect profile data about the current KB, the KB's items must call G2's profiling system procedures. Merge `sys-mod.kb` into your KB before you attempt to collect profile data. The KB defines these system procedures for profiling:

g2-enable-profiling	Directs G2 to start collecting profile data.
g2-disable-profiling	Directs G2 to stop collecting profile data.
g2-get-profiled-information	Creates and returns a system-profile-information item, which contains a copy of the profile data that currently resides in G2's own memory.
g2-clear-profile	Directs G2 to discard any collected profiled data that currently resides in G2's own memory.

Add actions or statements in your KB's items so that they call the procedures in a manner that supports your profiling strategy. Identifying Your Profiling Strategy offers basic recommendations.

Note Profiling does not track activities performed by the same item from which profiling is enabled.

Collecting Profile Data

G2 starts collecting profile data when your KB invokes the g2-enable-profiling system procedure. G2 stops collecting profile data when your KB invokes the g2-disable-profiling system procedure.

The first time that your KB calls g2-enable-profiling, G2 collects a new set of profile data. If your KB calls g2-disable-profiling to stop collecting profile data, and, later in the KB's processing, calls g2-enable-profiling again, G2 adds the newest profile data to the data already collected.

Note G2's profile data exists only as long as the G2 process itself exists. When you exit or close a G2 process, G2 discards any profile data already collected.

Changing the run-state of the current KB does not affect whether profiling is enabled. If profiling is enabled when the current KB is reset or paused, profiling remains enabled, but G2 collects no additional profiling data until the KB starts again. When the KB resumes, G2 adds any new profile data to the existing profile data.

See the *G2 System Procedures Reference Manual* for a complete description of the g2-enable-profiling and g2-disable-profiling system procedures.

Creating a Copy of the Collected Profile Data in G2

While G2 is collecting profile data, or after G2 has stopped collecting profile data, you can direct G2 to create an item that contains a copy of the profile data collected so far. The g2-get-profiled-information system procedure creates a transient system-profile-information. You can optionally transfer this system-profile-information to the workspace of your choice.

Note You cannot add data to, or remove data from, a system-profile-information item.

See the *G2 System Procedures Reference Manual* for a complete description of the g2-get-profiled-information system procedure.

Identifying the Contents of a System-Profile-Information

Each call to the g2-get-profile-information system procedure creates a new system-profile-information. The definition for the system-profile-information class is stored in `sys-mod.kb`.

Each system-profile-information contains three kinds of information:

- An attribute whose value represents the length of the profiling period in real time.
- Attributes whose values represent the time that G2 spent performing tasks managed by the G2 task scheduler.
- An item-list attribute whose elements are items of the item-profile-information-class class.

Use a system-profile-information's data to analyze how well your KB uses the CPU timeslice that your computer gave to your G2 process during profiling. In other words, a system-profile-information's data shows how efficiently G2 spends its time performing your KB's work relative to other tasks maintained by the G2 task scheduler.

A system-profile-information's data does *not* show the sum of the CPU timeslices within which your G2 process executed during the profiling period. Therefore, you cannot use the data from a system-profile-information to evaluate how efficiently G2 uses your computer's resources in relation to the computer's other activities. Such an analysis requires platform-specific techniques to monitor operating-system processes.

The following table summarizes the attributes specific to the system-profile-information class:

Attribute	Description
total-profiled-time	The interval of real time that passed during profiling.
idle-time	G2 clock time spent not performing activities that are managed by the G2 task scheduler.
clock-tick-time	G2 clock time spent servicing the G2 clock.
icp-time	G2 clock time spent on ICP data transfer and other network-related tasks, such as performing or checking for G2-to-G2 communication, G2 foreign functions, or remote procedure calls.

Attribute	Description
workstation-time	G2 clock time spent receiving user input and setting up G2 output to the workstation for activities such as keyboard and mouse events.
cisplay-time	G2 clock time spent rendering information on the workstation's display.
scheduling-time	G2 clock time spent by the G2 scheduler itself.
data-service-time	G2 clock time spent performing data service tasks for the current KB.
kb-io-time	G2 clock time spent on interactively saving the current KB.
overhead-time	G2 clock time spent on printing and licensing tasks managed by the scheduler, but not attributable by G2 to a particular task.
processing-time	(Derived) G2 clock time spent performing the KB's actions and statement; calculated as: total-profiled-time - (idle-time + icp-time + workstation-time + display-time + scheduling-time + data-service-time + kb-io-time + overhead-time)
profiled-items	An item-list whose elements are items of the item-profiling-information class.

Understanding Relationships among System-Profile-Information Attributes

As you attempt to make your KB perform more efficiently, you should observe the ratio of its processing-time to its total-profiled-time and the ratio of its idle-time to its total-profiled-time.

For example, in the table below, the attributes of profile-data indicate that, during this particular profiling period, G2 spent more of its time *waiting* for user-defined processing to execute than *performing* that processing.

PROFILE-DATA, a system-profil...	
Notes	OK
Item configuration	none
Names	PROFILE-DATA
Idle time	5.528
Clock tick time	0.02
ICP time	0.008
Workstation time	0.005
Display time	0.155
Scheduling time	0.001
Data service time	0.008
KB io time	0.0
Overhead time	0.0
Processing time	0.257
Total profiled time	5.982
Profiled items	an item-list

The ratio of processing-time to total-profiled-time (t0.257 / 5.982 = 0.04, or 4%) is much smaller than the ratio of idle-time to total-profiled-time (5.528 / 5.982 = 0.92, or 92%).

Representing Empty Profile Data

If your KB has not called g2-enable-profiling since your G2 process started, or since your KB most recently called g2-clear-profile, and your KB calls g2-get-profiled-information, G2 creates a system-profile-information whose timing attributes are values of zero and whose profiled-items item-list contains zero elements.

Understanding the Processing-Time Attribute

G2 derives system-profile-information's processing-time value from the total-profiled-time minus the sum of the time spent performing schedulable tasks, which consists of the values in the idle-time, icp-time, workstation-time, display-time, scheduling-time, data-service-time, kb-io-time, and overhead-time attributes.

Note G2 calculates the value of the processing-time attribute from a real time measure, not from G2's CPU timeslice measure. Therefore, to allow g2-get-profiled-information to return a processing-time measure that most accurately reflects your KB's actual performance within G2, you should profile your KB when G2 is the only process running, or one of very few processes running, on your computer.

Note If you are profiling a G2 process whose process window (or connected Telewindows) displays under the control of an X Windows server process, the actual time spent during scheduled screen-drawing cannot be included in the processing-time attribute's value.

Understanding the Profiled-Items Attribute

The profiled-items attribute of a system-profile-information is an item-list. If G2 has collected profile data before g2-get-profiled-information was most recently called, then G2 inserts item-profile-information items into this item-list. Each item-profile-information contains profile data about each executable item that was invoked during profiling.

The definition for the item-profile-information class is stored in `sys-mod.kb`. The following table summarizes the attributes specific to the item-profile-information class:

Attribute	Description
procedure-id	Item's names attribute, or a G2-generated name.
calls	Number of times the item was invoked during profiling.
total-time	Total G2 clock time spent executing this item during profiling.
time-per-call	(Derived) Average G2 clock time spent per invocation of the item calculated as: total-time / calls
profiled-activities	An item-list of activity-profile-information items.

For each item-profile-information inserted in the profiled-items attribute's item-list, G2 automatically concludes an instance of a profiled-by relation between the item-profile-information and the item it references. Find the profiled-by relation definition in the Profiling Procedures workspace in the `sys-mod.kb` file.

Tip Use the profiled-by relations in conjunction with the g2-name-for-item system procedure to assign G2-generated names to unnamed items that were invoked during profiling.

Understanding the Profiled-Activities Attribute

The profiled-activities attribute of an item-profile-information is an item-list. For each action or statement that G2 performed within an invoked item during profiling, G2 creates an activity-profile-information item and inserts it into this item-list.

The definition for the activity-profile-information class is stored in the Profiling Procedures workspace in `sys-mod.kb`. The following table summarizes the attributes specific to the activity-profile-information class.

Attribute	Description
activity-name	Name of a G2 executable activity (See the table Actions and Statements that G2 Profiles below.)
activity-count	Number of times this activity executed for this item during profiling

The following table lists the activities in an invocable item for which G2 collects profile data. Each activity represents either an action or a statement.

Actions and Statements that G2 Profiles

Activity Name (as reported)	Description
abort action	Abort a procedure or procedure invocation.
activate action	Activate an activatable subworkspace.
allow other processing statement	allow other processing statement.
assign local variable statement	Assignment to a local name.
begin rule actions	Invoke a rule.
call next method statement	Call another procedure.
call statement	Call another procedure.
case statement	case ... of statement.

Actions and Statements that G2 Profiles

Activity Name (as reported)	Description
change action	All change actions other than change color attribute/color pattern, change the text of, and change element in list or array.
change array or list element action	change element in list or array.
change color action	Change color attribute or color pattern of an item.
change text action	Change value of a text attribute.
collect data statement	collect data statement.
conclude action	Assign value to a table attribute.
conclude has no current value action	Force a variable's value to be expired.
conclude has no value action	Assign value to a table attribute.
conclude not related action	Remove a relation instance between two items.
conclude relation action	Create a relation instance between two items.
create action	Create a transient item (not a connection).
create by cloning action	Create a transient item (not a connection) by cloning.
create connection action	Create a transient connection.
create an explanation action	Create an explanation item.
deactivate action	Deactivate an activatable subworkspace.
delete action	Delete a transient item.
do in parallel statement	do in parallel statement.
do in parallel until one completes statement	do in parallel until one completes statement.
exit if statement	exit if statement.
focus action	Invoke rules of specific focal class.
for in parallel statement	Parallel iteration.

Actions and Statements that G2 Profiles

Activity Name (as reported)	Description
for in parallel until one completes statement	Parallel iteration with race condition.
for statement	for statement.
halt action	G2 stops running the current KB.
hide action	Hide a workspace.
if-then statement	if ... then statement.
if-then-else statement	if ... then ... else statement.
inform action	Place a message on the specified workspace.
insert action	Add an element to a g2-list.
invoke action	Invoke a rule.
make permanent action	Make an item permanent
make subworkspace action	Create a subworkspace of an item.
make transient action	Make a item transient.
move action	Change position of an item's icon within a workspace.
on error statement	on error statement.
pause kb action	Change the run-state of the current KB to paused.
print action	Print a workspace.
remove action	Remove an element from a g2-list.
repeat statement	repeat statement.
reset kb action	Change the run-state of the current KB to initial/reset.
return statement	return statement.
rotate action	Rotate an item's icon on a workspace.

Actions and Statements that G2 Profiles

Activity Name (as reported)	Description
set action	Assign a value to a GSI variable or a simulation variable. The G2 Simulator, which can use simulation variables, is a superseded capability. For more information, see Appendix F, Superseded Practices.
show action	Display a workspace.
shut down G2 action	Shut down G2.
signal error statement	signal statement.
start action	Invoke a G2 procedure.
start rpc action	Invoke a remote procedure.
system call statement	Call an internal G2 operation or procedure.
transfer action	Transfer an item to a workspace.
update action	Update a display item or variable.
wait for interval statement	wait for interval statement.
wait until event statement	wait until statement with event predicate.
wait until statement	wait until statement.

Profiling Executable Items and Activities

G2 profiles executable items starting with the first operation executed in the item and ending with the first operation executed in the next executable item that the KB's processing invokes, or ending with the first operation in the next distinct G2 task that is invoked.

Because individual G2 activities are fine-grained, G2 does not collect timing data on each activity invocation. Rather, when profiling, G2 collects only the number of times each activity is invoked within a given executable item.

Resetting Profile Data in G2

In some situations, you might prefer to clear G2's existing profile data before collecting more profile data. To clear all profile data from G2, invoke the g2-clear-profile system procedure.

After your KB has invoked g2-enable-profiling once, G2 retains a set of profile data in memory until g2-clear-profile is invoked.

See the *G2 System Procedures Reference Manual* for a complete description of the g2-clear-profile system procedure.

Identifying Your Profiling Strategy

Typically, you collect profile data for some length of time that is significant for testing purposes. You can begin by collecting profile data for all executable items in your KB. For example, to do so, you could include an initially rule in your KB that invokes g2-enable-profiling.

You must code your own actions or statements in your KB that call these system procedures, as follows:

- Include a call to g2-enable-profiling in the item that represents the start of the portion of KB processing that you want to profile.

Note Profiling does not track activities performed by the same item from which profiling is enabled.

- Include a call to g2-disable-profiling in the item that represents the end of the portion of KB processing that you want to profile.
- Include a call to g2-get-profiled-information; you might prefer to transfer the new system-profile-information to a particular workspace.
- Include a call to g2-clear-profile in an item that is invoked when you wish to collect an entirely new set of profile data.

For example, code the actions or statements in your KB's items so that they call the profiling system procedures in this order:

1. Call g2-enable-profiling to begin collecting profile data.
2. Call g2-disable-profiling to stop collecting profile data.
3. Whether automatically or manually, invoke the item that calls g2-get-profiled-information.
4. After calling g2-clear-profile or not, repeat Steps 1 through 3 to capture and report on your KB's processing again under different circumstances.
5. Whether automatically or manually, invoke the item that calls your own reporting procedure for the profile data contained in a particular system-profile-information item.

Reporting the Contents of a System-Profile-Information

You can code your own procedure that reports the contents of a system-profile-information. Use the procedure code below as a foundation for your report procedure.

Hint Some items that are profiled might not have a name. As shown in the sample procedure that follows, you can use the g2-name-for-item system procedure to assign a generated name to any unnamed profiled item. See the *G2 System Procedures Reference Manual* for a full description of g2-name-for-item.

The following procedures demonstrates profiling system procedures:

```
report-profile-data ( spi : class system-profile-information )
{
This procedure reports the contents of a system-profile-information.
This procedure uses the g2-name-for-item system procedure and the
profiled-by relation (also found in sys-proc.kb).

This procedure calls three procedures that you must also create:

* Report the value of an attribute of a system-profile-information:
    report-spi-attribute
        ( spi-value : item-or-value , report-workspace : class kb-workspace )

* Report the value of an attribute of an item-profile-information:
    report-spi-profiled-item-attribute
        ( profiled-item-value : item-or-value,
        report-workspace : class kb-workspace)


* Report the value of an attribute of an activity-profile-information:
    report-spi-profiled-activity-attribute
        ( profiled-activity-value : item-or-value , report-workspace : class
         kb-workspace )

Code your own versions of the three procedures above to report the
attributes of a system-profile-information, item-profile-information, and
activity-profile-information in the manner you prefer.
}
report            : class kb-workspace ;
item-profile      : class item-profile-information ;
activity-profile  : class activity-profile-information ;

use-this-name     : item-or-value ;
```

```
begin
    { Create a kb-workspace to display the report. }

    create a kb-workspace report ;
    change the name of report to the symbol profile-report ;

    { Call report-spi-attribute once for each attribute of the
        system-profile-information }

    call report-spi-attribute (the clock-tick-time of spi, report);
    { call report-spi-attribute ( the ... of spi , report ) ; }

    { Report on each profiled item for the system-profile-information. }

    for item-profile =
        each item-profile-information in the profiled-items of spi do
        begin
            use-this-name =
                call g2-name-for-item (the item that is profiled-by item-profile);
                call report-spi-profiled-item-attribute (use-this-name, report);

            { Call report-spi-profiled-item-attribute once for each attribute
             of this item-profile-information ... }

            call report-spi-profiled-item-attribute (the calls of item-profile,
                                                     report ) ;
            { call report-spi-profiled-item-attribute ( the ... of item-profile ,
                                                        report ) ; }

            { Report on each profiled activity for this profiled item. }
            for activity-profile = each activity-profile-information in the
             profiled-activities of item-profile do
                begin

                    { Call report-spi-profiled-activity-attribute once for each
                     attribute of this activity-profile-information . . . }

                    call report-spi-profiled-activity-attribute ( the
                     activity-name of activity-profile , report ) ;
                    { call report-spi-profiled-activity-attribute ( the ... of
                          activity-profile , report ) ; }

                end { begin }
            end { do }
        end { begin }
    end ; { do }
end
```

Analyzing Profiling Data

Collecting and analyzing profile data are the first steps in evaluating your KB's performance. With this information, you can use several techniques to improve your KB's efficiency of execution:

- From your best assessment of the data that your KB must process, implement the most appropriate algorithm in each procedure and function whose operation is time-critical.
- For rules, procedures, and functions that execute most often, apply compilation configurations (see Using Compilation Configurations) for best performance.
- Use G2's data-service features to collect the least amount of data that your application requires.
- Design and organize your KB's rules so that they fire under the minimum set of circumstances that your application must support.

Using Compilation Configurations

Compilation configurations make possible an incremental improvement in the performance of your KB. Some compilation configurations are designed to work in conjunction with each other, while others can be used alone.

G2 offers several compilation configurations:

- inlineable: Declares that a procedure or method can be inlined into another.
- stable-hierarchy: For modules and related items, this configuration declares that changes will not be made to the module in which the configuration appears, or to the class hierarchy with which a module is associated.
- stable-for-dependent-compilations: Declares that an item is not subject to further change. This allows G2 to compile more efficiently other items that refer to the configured item.
- independent-for-all-compilations: For an item whose attributes refer to another item that is declared stable-for-dependent-compilations, directs G2 *not* to compile the item to take advantage of the other item's stability.

In this section we present the rationale behind using compilation configurations and their role in improving the performance of your KBs.

Stability Configurations

Use the stable- prefix configurations to declare that a portion of your KB is not subject to change, even when the KB is running. G2 consults these declarations when compiling items that refer to the stable portion of the KB.

G2 compiles references to a stable item differently than it compiles references to an item that is not stable. For stable items, G2 can assume that the name, class, and item location in the KB's workspace will not change as the KB runs. Therefore, G2 compiles the referencing item in a way that reduces the number of validation checks G2 must perform when executing or evaluating the actions, statements, or expressions that refer to the stable item.

Declaring the Configurations

An item can fall within the scope of configurations declared in other items that are higher in the KB's workspace and class hierarchies. Use the describe configuration menu choice to see all configurations that apply to an item.

You declare an item to be stable or not by including one or more of these configuration statements in the item's item-configuration or instance-configuration attributes:

declare properties ... as follows : stable-hierarchy

declare properties ... as follows : stable-for-dependent-compilations

declare properties ... as follows : independent-for-all-compilations

See Configuring Properties of Items for an introduction to G2's configurations features and for the complete syntax of the declare properties ... as follows configuration statement.

Understanding Compiled Attributes

In general, G2 compiles each item attribute that can contain an expression, an action, or a statement. These are called **compiled attributes**.

When a compiled attribute contains an action or statement, G2 can *invoke* that attribute's contents, such as the text of a procedure or rule. When a compiled attribute contains an expression, G2 can *evaluate* or *reference* the attribute's contents, such as the formula of a variable or the expression-to-display of a readout-table.

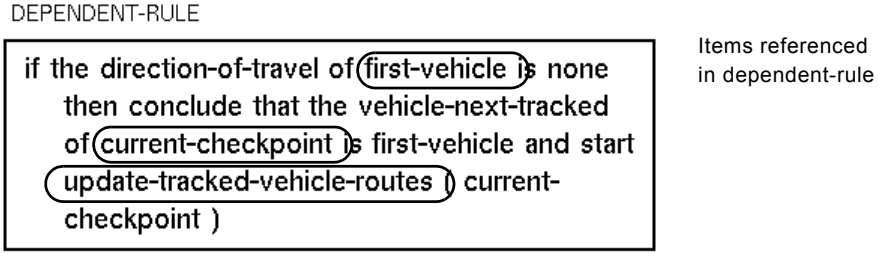
Note If you change a compiled attribute's value using the Text Editor, G2 recompiles that attribute after you direct G2 to save the attribute's edited text. If you make a valid change to a compiled attribute's value using a change the text of or conclude action, G2 automatically recompiles the attribute.

Validating References at Run-Time

The contents of a running KB are always subject to change. Therefore, when G2 compiles an attribute that refers to another item, G2 does not, by default, record any assumptions about the referenced item, such as whether that item actually exists or whether it is of a certain class.

Instead, by default, G2 compiles such an attribute to include **run-time validation** instructions. When that compiled attribute is invoked, evaluated, or referenced, the attribute's run-time validation instructions cause G2 to verify whether the attribute's own assumptions about the referenced item (its name, class, etc.) are still true.

To illustrate, consider the rule dependent-rule, shown in the following figure. The text of dependent-rule refers to a user-defined object named first-vehicle, whose class is vehicle, a user-defined procedure named update-tracked-vehicle-routes, and a user-defined object named current-checkpoint, whose class is checkpoint:



By default, when G2 compiles this rule's text, G2 does not assume that first-vehicle, update-tracked-vehicle-routes, and current-checkpoint exist. (Even if, for example, first-vehicle does exist at the time that G2 compiles this rule, the item might cease to exist, or its class's class inheritance path might change, before your KB next invokes the rule.) Therefore, in the rule's compiled instructions G2 includes run-time validation instructions.

In this case, before G2 allows the rule to be invoked, the rule's run-time validation instructions verify the following:

- first-vehicle exists, and its class is vehicle.
- update-tracked-vehicle-routes exists, and its one argument is of type class checkpoint.
- current-checkpoint exists, and its class is checkpoint.

On the other hand, when G2 compiles this rule's text, if G2 can assume the items referenced by name do exist, then G2 can avoid including the run-time validation instructions that the rule's references require.

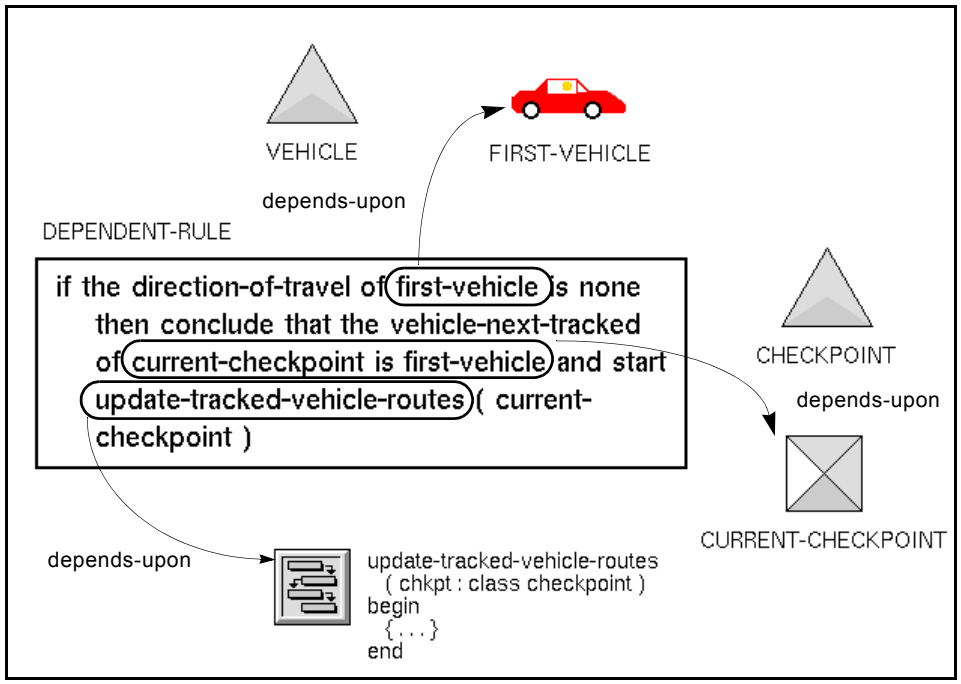
To apply this idea to your KB as a whole: the more items whose compiled attributes can avoid including run-time validation instructions, the greater the potential for an incremental improvement in your KB's performance.

Understanding Compilation Dependencies

An item with a compiled attribute whose text refers to another item has a **compilation dependency** relationship to that other item. When G2 compiles an attribute, it automatically identifies and maintains any compilation dependency relationships between the compiled item and the item (or items) that its compiled attributes reference.

To illustrate, dependent-rule's antecedent refers to the item first-vehicle. Due to this reference, for your KB to invoke this rule without error requires that an item named first-vehicle exists in the KB, that first-vehicle's class is vehicle, and that the vehicle class maintain the same direct superior classes and direct subclasses that it has at the time dependent-rule is compiled.

These requirements represent a compilation dependency between the rule and first-vehicle. The diagram in the following figure represents this relationship graphically:



Likewise, dependent-rule has compilation dependencies on the existence of current-checkpoint and update-tracked-vehicle-routes.

When compiling an attribute, G2 can identify the compilation dependency relationships shown in the following table:

When the reference that causes compilation dependency...	Then the compiled item depends on...
Refers to an item by name	The existence of the named item.
Refers to an item's type	The named item being of a particular type specification.
Refers to an item's class	The named item being of a particular class.
Invokes a procedure (or remote-procedure) by name using a start action or call statement	The existence of the named procedure (or remote-procedure) and on the types and number of its arguments and return values.
Refers to a class as being a subclass of another class	One class being a subclass of another.
Refers to a class as not being the subclass of another class	One class not being a subclass of another.

You can make changes to stable items that do not affect those characteristics, such as concluding values into attributes, establishing relation instances, or storing items in lists.

Declaring Procedures and Methods as Inlineable

The inlineable configuration is applicable to:

- Procedures
- Methods

When procedures and methods are declared as inlineable, they exist as separate items, but are compiled as part of the method or procedure code from which they are called. Inlining can improve performance by:

- Avoiding the overhead of procedure invocations, which consume runtime memory.
- Reducing the total number of instructions executed between the calling procedure or method and the inlined procedure or method.

When declaring a procedure as inlineable, you must also declare it to be stable-for-dependent-compilations.

When declaring a method as inlineable, you must also declare it to be stable-hierarchy and stable-for-dependent-compilations.

Recompilation Considerations

After configuring a procedure or method as inlineable, and making it stable with its other required configuration statements, you must manually recompile both the inlineable item and the procedure or method that calls it. G2 does not present a recompilation dialog, though a message about recompiling is added to the notes of the procedure or method.

By recompiling the calling procedure or method, G2 compiles the inlined code as part of the calling code. Since inlined procedures and methods are compiled in their calling procedure, they do not require a procedure invocation at run time, and performance improves.

Editing an existing inlined procedure or method causes G2 to display a dialog indicating that the item is stable and editing could make recompilation necessary.

Declaring Items as Stable-Hierarchy

The stable-hierarchy configuration is applicable to:

- A method.
- A method declaration.
- The class for which a method exists.

Declaring an item as stable-hierarchy indicates that neither the class hierarchy of the method, nor the class associated with a method will be changed in any way.

Declaring a method as stable-hierarchy also implies that the return value types and the number and type of method arguments will not change.

When you declare an item as stable-hierarchy, G2 can complete optimizations, including inlining, only if these conditions are met:

- The method or method-declaration is configured as stable-hierarchy.
- In the calling procedure or method, the class for which the method is called (the first argument of the method) has no subclass that defines a method of the same name with the same number of arguments.

  For example, if there is a fill method whose first argument is vessel, for optimization to occur, the class vessel could not have a subclass that defined a fill method with the same number of arguments. At run time, such a method hierarchy would make it impossible for G2 to know which method was actually being called, since the vessel class passed to the fill method could be either vessel or one of its subclasses.

Declaring Items Stable-for-Dependent-Compilations

You apply a stable-for-dependent-compilations configuration to an item upon which other items have a compilation dependency. If the item is an instance of a user-defined class, then for G2 to consider that item as stable, you must also apply a stable-for-dependent-compilations *item configuration* to the definition item for its class.

Note After applying a stable-for-dependent-compilations configuration to a set of items, use the Inspect facility to recompile your entire KB. Until you do so, your KB's performance cannot take advantage of the newly declared item stability.

In our example, if you apply a stable-for-dependent-compilations configuration on first-vehicle then recompile your KB, the compiled attributes in dependent-rule that have a compilation dependency upon first-vehicle no longer include instructions that check, for instance, whether first-vehicle exists. This helps dependent-rule execute more quickly than it could before first-vehicle was configured stable-for-dependent-compilations.

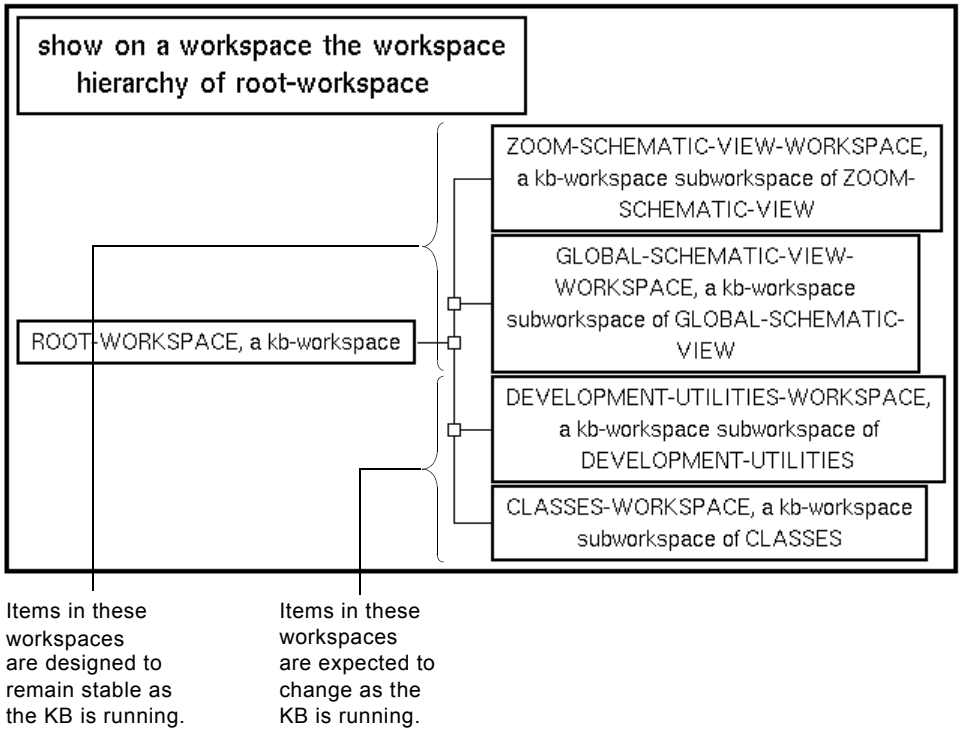
Identifying Potential Performance Improvements

To gain the greatest potential performance improvements in your KB, we recommend that you take this approach:

1. Configure the *entire* KB as stable-for-dependent-compilations. To do so, include a declare properties as follows : stable-for-dependent-compilations statement in the item-configuration attribute of each top-level workspace in your KB. (This approach is especially preferred for modular KBs.)
2. Configure each workspace subhierarchy containing items whose knowledge can change as the KB runs as *not* being stable. You do this by including a declare properties as follows : not stable-for-dependent-compilations statement in the item-configuration attribute of the workspace representing each subhierarchy of items that is subject to change.

To take this approach, you must organize the workspaces in your KB's workspace hierarchy so that some workspace hierarchies are designed to remain stable during your KB development project, while other regions are designed to contain

transient items and other items whose knowledge is subject to change as the KB runs. The following figure illustrates this approach:



This Inspect workspace shows the workspace hierarchy for a simple KB that contains five workspaces. The items located in the development-utilities-workspace and classes-workspace workspaces contain items that represent the KB's stable knowledge (items whose definitional characteristics don't change as the KB runs).

On the other hand, the global-schematic-view-workspace and zoom-schematic-view-workspace workspaces contain items that represent the KB's dynamic knowledge (the items whose knowledge is expected to change as the KB runs).

For this sample KB, after configuring the entire KB as stable-for-dependent-compilations, you would configure only the global-schematic-view-workspace and zoom-schematic-view-workspace workspaces as *not* stable-for-dependent-compilations.

If you choose not take this approach, the following principles determine the potential performance improvement for your KB:

- As more compiled attributes are recompiled to take advantage of item stability by including run-time validation instructions, a greater potential exists to improve incrementally your KB's performance.
- When your KB is running, the KB's processing invokes, evaluates, and references some compiled attributes more often than others. When G2 compiles a more frequently used compiled attribute without run-time validation instructions, your KB has a greater potential for an incremental performance improvement than when a less frequently used compiled attribute is so compiled.

Identifying Knowledge That is Not Eligible for Performance Improvements

Some aspects of your KB's processing are not subject to performance improvements based on declaring item stability:

- Graphics drawing operations.
- Network transmission operations.

Understanding Guidelines for Configuring Groups of Items

Keep the following guidelines in mind before adding a stable-for-dependent-compilations configuration that has a wide scope:

- Avoid configuring too many items as stable-for-dependent-compilations too early in your KB development project. Doing so will require you to recompile dependent items more frequently, as you make changes to your KB.
- Avoid configuring a class's definition as stable-for-dependent-compilations unless its direct superior classes are also configured stable-for-dependent-compilations. While G2 does not prevent you from constructing your KB's class hierarchy to consist of unstable classes with stable direct subclasses, such a practice is likely to lead to problems.
- Apply the stable-for-dependent-compilations configuration and perform KB-wide compilation *before* text-stripping any of the KB's items.

Understanding Guidelines for Configuring Items in a Modular KB

In a modular KB, items in a required module should not have compilation dependences upon items (such as definitions) in a requiring module. In general, do not declare compilation configurations based upon compilation dependencies that run contrary to the KB's module dependencies.

Declaring Items Independent-for-All-Compilations

When G2 compiles any attribute in an item declared as independent-for-all-compilations, G2 does *not* compile the item to take advantage of compilation dependencies upon any other items declared as stable-for-dependent-compilations. Declaring an item independent-for-all-compilations affects that item's compilation only if the item's compiled attributes reference other items. In general, only use this option if you don't trust stable configured items to remain stable, probably a rare occurrence.

For example, if a compiled attribute in item A depends upon other items, one of which (item B) is configured as stable-for-dependent-compilations, then you can declare item A as independent-for-all-compilations. This causes G2 to compile item A *without* taking advantage of any optimizations due to its dependency on item B.

Note You do *not* gain any potential improvement in your KB's performance when you configure items as independent-for-all-compilations. Rather, use this configuration to isolate an item from requiring a recompilation whenever another item, upon which it depends, changes its name, its class, and so on.

For an item that has compilation dependencies on other items, configuring that item as independent-for-all-compilations allows you to choose when that item is next recompiled. This can be preferable in the following two situations.

Isolating a Group of Items From Automatic Recompilation

Assume that you have created an item, such as a procedure, that has compilation dependencies on a few other items, such as other procedures and functions, and those other items will change often as you develop your KB. Assume also that many other items depend upon your item, such as more general-purpose procedures and functions that must call your procedure. In this case, by configuring your procedure as independent-for-all-compilations, you can prevent your procedure from being subject to automatic recompilation due to changes in an item upon which it depends.

Further, because other items in your KB depend upon your procedure, you can also configure it as stable-for-dependent-compilations.

Isolating a Group of Items from Items Provided by Other Developers

In a modular KB that directly requires a proprietary modular KB provided by other G2 developers, assume that you have created a procedure that has compilation dependencies upon other procedures and functions in the provided modular KB. If the next version of the provided modular KB contains a change in the procedures and functions that your procedure depends upon, you must recompile your procedure, as well any other items in your modular KB that depend upon your procedure.

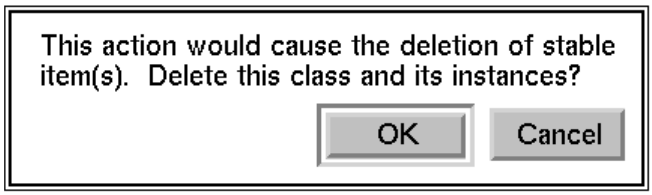
To manage the activity of recompiling items in your own modular KB that depend on items in other modular KBs that you do not control, declare as independent-for-all-compilations all items in your modular KB that depend upon items in the provided KB.

Changing Items That Have Compilation Configurations

As items change during your KB's processing, the compilation dependency relationships declared among the KB's items can also change. This section describes the kinds of changes to compilation dependencies that G2 recognizes and how the dependencies themselves are changed.

After Deleting an Item Declared Stable-for-Dependent-Compilations

The most drastic change to a stable item is to delete it. When you interactively delete an item declared as stable-for-dependent-compilations, G2 first displays a confirmation dialog. For instance, if deleting a definition item whose instances are within the scope of a stable-for-dependent-compilations configuration, G2 displays this dialog:



If your KB's processing deletes stables items programmatically, G2 does not prompt for confirmation.

After G2 deletes an item that is within the scope of a stable-for-dependent-compilations configuration, G2 removes the compilation dependency relationships between the deleted item and its dependent items.

After Changing the Knowledge of Items Declared Stable-for-Dependent-Compilations

For items that depend on an item declared stable-for-dependent-compilations, G2 compiles those items with the assumption that the following knowledge will not change:

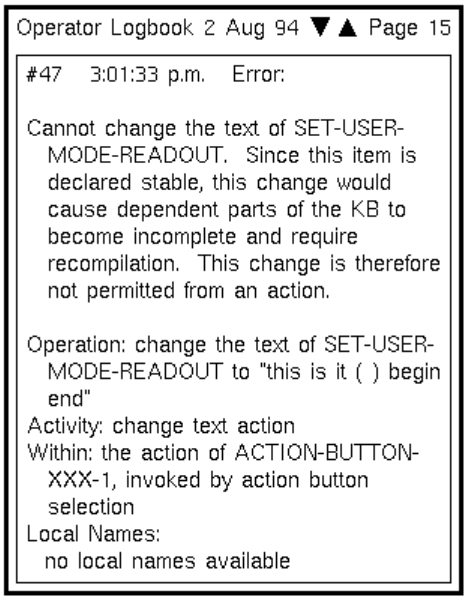
- The stable item's name.
- The stable item's class.
- The direct superior classes of the stable item's class.

- The direct subclasses of the stable item's class.
- If the stable item is a procedure or remote-procedure-declaration, the number and types of arguments and return values.

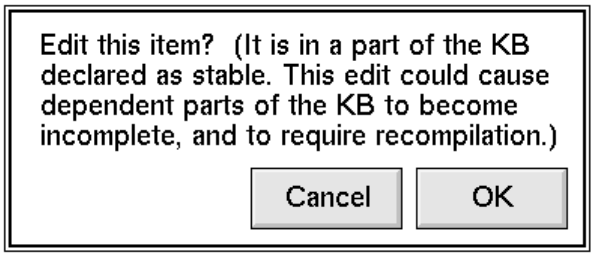
Therefore, G2 prevents a KB's processing from performing the following actions on items declared stable-for-dependent-compilations:

- For any item, a change the name of action applied to the item that changes its names attribute.
- For a class definition, a change the text of or conclude action applied to the class-name or direct-superior-classes attribute that changes the defined class's name or any direct superior class.
- For a procedure or remote procedure declaration, a change the text of action applied to the item's text that changes the item's name, or the number and types of its arguments and returned values.
- For a permanent item, a make transient action applied to the item.

If your KB performs one of these actions programmatically, G2 denies the attempt and signals an error. A sample Operator Logbook error message appears below:



If you interactively edit a compiled attribute of an item declared stable-for-dependent-compilations, G2 displays the following dialog:



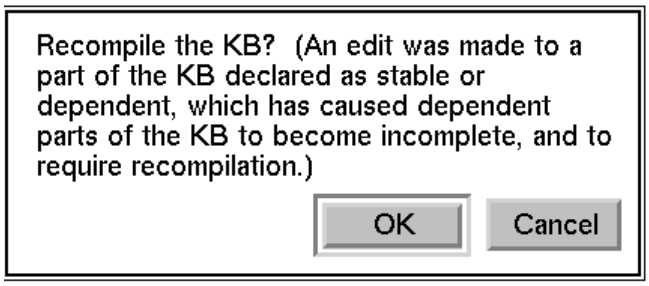
Press OK to enter the Text Editor. Press Cancel to rescind your attempt to edit the attribute.

When you finish editing the compiled attribute in the Text Editor, G2 compiles the attribute, then checks whether any compilation dependency relationships with dependent items are affected by the edit. For each dependent item whose compilation dependency is affected, G2 changes its OK/incomplete/bad status to *incomplete*. Before G2 can again use the affected item in your KB's processing, you must recompile the item so that its compilation status (shown in the item's notes attribute) is again OK.

After Removing a Stable-for-Dependent-Compilations Configuration

Assume that your KB contains an item declared stable-for-dependent-compilations, and contains other items, each with its own compilation dependency relationship with the stable item. Further, assume that the dependent items have been compiled to take advantage of their dependency on the stable item.

Given these assumptions, if you remove the declare properties ... as follows : stable-for-dependent-compilations configuration statement from the stable item, you break the compilation dependencies among the KB's items. Therefore, after you remove a stable-for-dependent-compilations configuration from an item, G2 sets the OK/incomplete/bad status of the dependent items to *incomplete*. G2 also displays the dialog shown next:



You have two choices:

- Press OK to direct G2 to recompile all items in the KB whose OK/incomplete/bad status is *incomplete*. This is equivalent to performing the directive within the Inspect facility:

  recompile every item whose status is incomplete

- Press Cancel to direct G2 *not* to recompile at this time the KB's items whose OK/incomplete/bad status is *incomplete*.

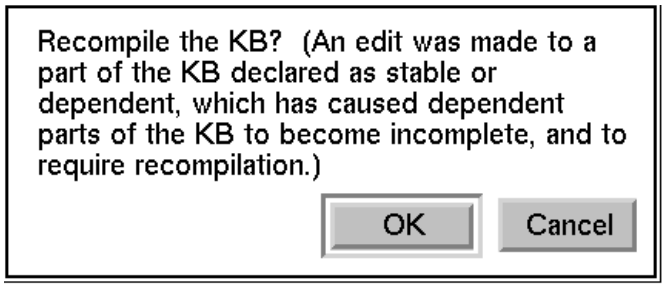
Tip Your KB's processing cannot use items whose OK/incomplete/bad status is *incomplete* until they are successfully recompiled.

After Changing an Unconfigured Dependent Item to an Independent Item

For an item that has a compilation dependency on another item, but is not itself subject to any compilation configuration, G2 recognizes the following ways to change the item's compilation status:

- Add a declare properties ... as follows : independent-for-all-compilations item configuration to the item.
- Transfer the item from a workspace that is dependent on a stable item to a workspace that is subject to a declare properties ... as follows : independent-for-all-compilations item configuration.

Performing any of these operations causes G2 to set the item's OK/incomplete/bad status to incomplete, then displays the following dialog that initiates a recompilation of all items with incomplete status in the KB:



54

G2-Meters

Shows how to create, configure, and use G2-meters.

Introduction

G2-meters are specialized quantitative variables that monitor G2 and compute statistics about its performance, such as how much memory it is using, and how fast it is processing.

Caution Don't confuse G2-meters with display meters. A display meter is a graphical item that shows a numeric value, as described in Readout Tables, Dials, and Meters.

G2-meters measure G2's actual performance, not its capability. For example, the meter percent-run-time measures how much processing time G2 is using per

second. It does not measure the amount of processing time that G2 is capable of using within a second.

Note All G2-meters put an extra load on the system. They should be used only when needed.

This chapter does not describe G2 memory organization and management. For information on these topics, see Memory Management.

Working with G2-Meters

The general technique for working with G2-meters is:

- Set system table attributes that control G2-meters. You can set these attributes at any time.
- Define a subclass of quantitative-variable that inherits the g2-meter-data-service mixin.
- Instantiate the class to create a G2-meter.
- Edit the G2-meter's attributes as needed to specify the meter's name, type, and other properties.
- Use the meter to measure performance statistics.

Enabling and Disabling G2 Meter Service

You can enable and disable all G2-meters at any time by changing the g2-meter-data-service-on? attribute in the Data Server Parameters system table. When you enable G2-meter service, all G2-meters begin to function; when you disable it, they cease to function.

G2-meter service is disabled by default. When G2-meter service is disabled, G2-meters do not put any load on the system. You can create a new G2-meter while G2-meter service is disabled, but the meter will not function until you enable the service.

To enable G2-meter service:

➔ Edit the g2-meter-data-service-on? attribute in the Data Server Parameters system table to specify yes.

To disable G2-meter service:

➔ Edit the g2-meter-data-service-on? attribute in the Data Server Parameters system table to specify no.

For more information, see Data Server Parameters.

Specifying the Meter Lag Time

You can change the degree to which all G2 meters smooth data by changing them meter-lag-time attribute of the Timing Parameters system table. You can change this attribute at any time; the change takes effect immediately.

G2-meters monitor events in time. G2-meters can compute values for the most recent clock-tick, or they can compute values based on a smoothed result of recent clock-ticks. Such values are called **lagged values**, because they do not vary as drastically as the instantaneous events they measure. Rather, they represent a first-order delayed reaction to the event. The time interval over which a G2-meter smooths values is called the **meter lag time**.

To clarify this, it may be helpful to think of meter lag time as it exists on the dashboard of a car. For example, the fuel gauge in a car has a large meter lag time, because you don't want the needle on the fuel gauge to swing wildly as the gasoline sloshes in your fuel tank. Conversely, the speedometer has a small meter lag time, because you want it to respond immediately as the car goes faster and slower.

G2 computes lagged values as follows:

new lagged value = (1 - β) * previous lagged value + (β * current value)

where:

β = min (1.0, clock tick length / meter lag time)	Is an Euler approximation of first-order delay. Note that if the meter lag time is zero or is less than the latest clock tick length, then β = 1.0, and the new lagged value equals the current value, with no lag.

The meter-lag-time attribute of the Timing Parameters system table holds a value of 0 seconds or any longer time interval. If it holds 0 seconds, G2-meters reflect only the activity in the most recently completed clock tick. As its value increases, the values of G2-meters change more smoothly over time.

To set the meter lag time:

➔ Edit the meter-lag-time attribute of the Timing Parameters system table to specify the desired value.

The change takes effect immediately. For more information, see Timing Parameters.

Some G2-meters keep absolute counts of events, so they do not provide lagged values. This is noted in the descriptions of particular G2-meters later in this chapter.

Creating G2-Meters

This section describes the steps for creating any kind of G2-meter. Types of G2-Meters lists all G2-meters and describes what each type does.

Before you can create a G2-meter, you must create a user-defined class called a **G2-meter class**. Every G2-meter is an instance of such a class.

To create a G2-meter class:

1. Select KB Workspace > New Definition > class definition > class definition to create a new class definition on a workspace.
2. Give the class a unique name.
3. Specify the class's direct superiors as quantitative-variable and g2-meter-data-service.

The g2-meter-data-service is a mixin class that sets the data server to g2-meter, and gives the class an additional attribute called g2-meter-name. For information about mixin classes, see Using Mixin Classes.

You can customize and subclass a G2-meter class in any way, as with any user-defined class. You can define as many customized G2-meter classes as you need. Such customization does not affect the essential operation of the class, so this chapter refers only to G2-classes in general.

To create a G2-meter:

1. Create an instance of a G2-meter class.
2. Edit the g2-meter-name attribute of the meter to specify the type of meter.
3. Edit the meter's names attribute to specify any available name.
4. Edit the meter's validity-interval attribute to be 0 seconds.

   The meter's validity-interval cannot be supplied, because data derived by monitoring G2's performance has no intrinsic validity interval. Hence the g2-meter data server cannot supply explicit expiration times.
5. Edit the meter's default-update-interval to be 1 second.

   To correctly reflect the simulator's performance, the meter must be updated every second. The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices.
6. If you want to display the meter's readings on a graph, change history-keeping-spec to keep history.
7. Create a display to show the meter's value. You can use any type of display. For information on displays, see Readout Tables, Dials, and Meters.

If g2-meter-data-service-on? is enabled, the G2-meter begins functioning and displaying data as soon as you have completed its definition.

Disabling and Enabling Individual G2-Meters

After you have created a G2-meter, you can disable it without deleting it, then reenable it when desired. You can also disable and enable all meters at once, as described under Enabling and Disabling G2 Meter Service.

To disable a G2-meter:

➔ Edit the meter's g2-meter-name attribute to be none.

To enable a G2-meter:

➔ Edit the meter's g2-meter-name attribute to specify the type of the meter.

You can enable a meter to have any meter type, not just the type it had previously.

Interpreting G2-Meters That Measure Memory

Many G2-meters measure memory: how much has been allocated by the operating system, how much of this allocation is currently in use, and how much remains available. All memory measurements are in 8-bit bytes.

The values shown by G2 memory meters reflect only the space that G2 uses for storing data and graphics; they do *not* include the space that holds G2 itself. G2's intrinsic memory requirement is largely determined when G2 is compiled for a particular platform, and does not vary significantly with KB size or activity, so including it in memory meter measurements would accomplish little.

G2-Meter and Operating System Measurements

Unlike G2 memory meters, operating system commands that measure memory, such as:

- UNIX: `ps -l`
- Windows: The Process tab of the Task Manager

show both the memory that G2 itself occupies and the memory that it uses for storing data. Depending on the platform, they may or may not also include the memory G2 uses for storing graphics. See your system documentation for information on these commands, and to identify the analogous command(s) on other platforms.

Note On some UNIX systems, measurements printed by `ps -l` omit memory that has been allocated but has never been used.

For information on G2 memory management, see Memory Management.

Approximations in Memory Meter Readings

Small inaccuracies in memory meter readings, on the order of a few kilobytes, may occur due to the round-off necessary for efficient internal memory measurement. For this reason, the sum of the sizes reported for parts of G2 memory may not exactly equal the size reported for the whole.

Types of G2-Meters

The following table lists all G2-meters. A brief description of each type of meter follows the table. Some general considerations about memory meters appear in the previous section. All meters can have lagged values except where otherwise noted.

Memory Meters	Time Meters
instance-creation-count-as-float	clock-tick-length
memory-size	maximum-clock-tick-length
memory-usage	percent-run-time
memory-available	simulator-time-lag
region-1-memory-size	priority-1-scheduler-time-lag
region-1-memory-usage	priority-2-scheduler-time-lag
region-1-memory-available	priority-3-scheduler-time-lag
region-2-memory-size	priority-4-scheduler-time-lag
region-2-memory-usage	priority-5-scheduler-time-lag
region-2-memory-available	priority-6-scheduler-time-lag
region-3-memory-size	priority-7-scheduler-time-lag
region-3-memory-usage	priority-8-scheduler-time-lag

Memory Meters	Time Meters
region-3-memory-available	priority-9-scheduler-time-lag
	priority-10-scheduler-time-lag

The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices.

Instance-Creation-Count-as-Float

Historically, G2 has had an upper limit on the number of instances that it could create in a single session, both explicitly and as a side-effect of other actions. In previous releases, this limit was 2^{29} - 1, which was the largest integer that G2 could represent. In some cases, this limit is known to have been reached, in which case G2 behaves unpredictably.

To address this problem, beginning with G2 Version 6.0, this limit was increased to 2^{58} - 1, an increase of more than eight orders of magnitude over the previous limit, which allows for virtually unlimited object creation.

G2 Version 5.1 Rev. 9 introduced a new g2-meter, which allows you to monitor the creation of instances. The g2-meter is also available in G2 Version 6.0 and higher versions; however, you should no longer encounter this limit and, therefore, should not need to use the meter to monitor the creation of items.

In addition, G2 issues warnings on the console or to the log file on Windows platforms when 95, 96, 97, 98, and 99% levels of utilization are reached.

Memory-Size

Measures the total memory allocated to G2 by the operating system for holding data. The figure includes both used and available memory. The sum of memory-usage and memory-available should equal memory-size.

Memory-Usage

Measures the total amount of memory that G2 currently uses. As G2 creates more items, schedules more tasks, and the like, memory usage increases.

Memory-Available

Measures the total amount of memory currently allocated by the operating system but not used by G2.

Region-N-Memory-Size

Each meter measures the memory in the G2 region specified by *n*. The figure includes both used and available memory. For each region, the sum of memory available and memory usage should equal memory size.

Region-N-Memory-Usage

Each meter measures the amount of memory that G2 currently uses in the G2 region specified by *n*.

Region-N-Memory-Available

Each meter measures the total amount of memory currently available to G2 but not used by it in the G2 region specified by *n*.

Clock-Tick-Length

Computes how many seconds a G2 clock tick lasts. If the scheduler mode is real time, this value varies closely around 1 second. In other modes, clock-tick-length may vary more widely.

Maximum-Clock-Tick-Length

Holds the duration in seconds of the longest clock tick that G2 has experienced since the knowledge base started running. If the scheduler mode is not real time, the maximum clock tick length can be large, for example if you pause the knowledge base for a long time. maximum-clock-tick-length is not a lagged value.

Percent-Run-Time

Computes how much processing time G2 is using, as a percent of the processing time available for it to use. The value of the meter is recalculated at each clock tick by computing:

(elapsed-time - sleep-time) / elapsed-time

The time that G2 sleeps even though it could be processing acts as a reserve of processing power on which it could draw if the demand on it increased. The percent-run-time meter measures the size of that reserve: the lower the value, the more reserve exists.

In general, the percent-run-time should not be more than 80% unless you have used priorities to be sure that high-priority tasks will get done even if demand exceeds available time, causing G2 to lag.

Simulator-Time-Lag

Computes how many seconds behind the current system time the G2 simulator is. If the value is positive, this means that the simulator is behind schedule; if it is negative, the simulator is ahead. This value cannot be positive if the scheduler mode is simulated time. simulator-time-lag is not a lagged value.

The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices.

Priority-*N*-Scheduler-Time-Lag

These meters compute how many seconds behind current system time the scheduler is for a given priority. There are ten such meters, each representing the queue of tasks of a particular priority, within the current task queue. Tasks of priority one are first in the current task queue, then tasks of priority two, and so on.

If the value of one of these meters is positive, the scheduler is behind schedule on that priority's queue. For example, if the value of the priority-1-scheduler-time-lag meter is 0 and the value of the priority-2-scheduler-time-lag meter is 3, then all tasks of priority one are being completed, but task of priority two (and probably the lower priorities, as well) are behind by three clock ticks.

The value of any of these meters will always be 0 when the scheduler mode is either simulated time or as fast as possible. Scheduler-time-lag is not a lagged value.

55

Memory Management

Describes G2's memory regions and shows how to manage them.

Introduction

G2 memory management differs from that of most programs. Because G2 performs real-time process control, it must avoid pausing for memory management to the greatest possible extent. This requirement affects every aspect of G2's use of memory.

G2 uses memory of various kinds in various contexts:

- Memory belonging to system services, such as the window system.
- RAM on the computer that runs G2.
- Virtual memory provided by the operating system.
- Memory that holds G2's executable code.
- Memory that holds data constituting or kept by a KB.

This chapter describes all of these uses of memory, and shows you how to achieve optimal performance with each of them.

Managing KB Data Memory

Most G2 memory management decisions relate to memory that holds the data associated with a KB. The essential principles for managing such memory are:

1. Allocating and deallocating memory causes unpredictably long synchronous waits for the operating system to process the request. Such waits are incompatible with real-time performance. The goal of G2 memory management is to eliminate them.
2. When G2 starts, it receives an initial memory allocation, which is specified on the command line or with environment variables. G2 never asks for more memory unless an executing KB has exhausted this initial allocation. A correctly configured KB never exhausts its initial allocation.
3. Except for some memory used for icons and fonts, G2 never returns memory to the operating system. It retains all allocated memory in internal memory pools. Creating an item withdraws memory from a pool; deleting it returns memory to the pool. The memory is then available for reuse by G2.
4. A correctly configured KB does not require ever-increasing amounts of memory. The usual cause of unbounded memory requirement is code that creates transient items and does not explicitly delete them, causing them to accumulate as the KB executes. Other possible causes also exist.

After describing other types of memory, this chapter expands the four principles listed into a complete description of KB data memory management, the problems that can arise with it, and ways in which such problems can be solved.

G2 and System Services

As G2 runs on a system, it makes use of various services that the system provides, such as the network manager, the file system, and the window system. All of these use memory to accomplish their tasks, and their use of it is intrinsically beyond G2's knowledge and control.

G2 can be very demanding of system services, and may ask more of them than they are configured to provide, causing them to run out of memory. Such problems are a function of the particular system and its configuration, and not of G2 per se, so this chapter cannot provide an exact formula for preventing them.

Determining System Adequacy

The `readme-g2.html` file and the *G2 Bundle Release Notes* describes various requirements that a system must meet before G2 can run on it. However, due to the great variety of systems, problems can occur even when these guidelines are met. The best defense is testing: be sure that your application has made the maximum possible demands on the system it runs on before you conclude that the system is adequate for your needs.

G2, RAM, and Virtual Memory

Virtual memory can provide the effect of very large amounts of real memory but the benefit in space is bought with a sacrifice in time. Virtual memory is much slower than real memory, because it requires paging data between real memory and the disk.

Virtual memory systems page only when they must. If you have enough RAM to hold all executing programs in real memory, none of them will ever be paged. If G2 needs more than the amount of RAM your computer provides, or some other program competes with G2 for that memory, the operating system will page G2 as needed to provide the necessary memory virtually.

Moderate amounts of paging do not typically cause significant performance problems. However, incessant paging (thrashing) caused by lack of RAM can drastically degrade performance. Gensym strongly advises against attempting to use G2 under such circumstances.

Determining RAM Requirements

To determine how much RAM you need, measure the total G2 memory requirement as described under Measuring Memory with Operating System Commands. You should have at least that much RAM in your machine, preferably at least 16 MB more, and in no event less than 128 MB for Alpha AXP

machines and 64 MB for other platforms. See the `readme-g2.html` for further information.

If you intend to run other programs on the machine that runs G2, you should provide additional RAM as needed to prevent them from competing excessively with G2. Many installations provide monitoring tools that can be used to measure how different processes use virtual memory. These can be helpful in deciding how much more RAM to provide when G2 does not run alone.

Introduction to G2 Memory Management

Providing an adequate system and sufficient RAM is necessary to ensure good G2 performance, but such provision does not constitute memory management. G2 memory management consists of two things:

- Preallocate all memory that G2 will need while executing the current KB.
- Ensure that the KB does not require ever-increasing amounts of memory.

The goal in both cases is the same: to minimize and if possible eliminate memory allocation requests to the operating system while a KB executes.

Most programs manage memory by calling on the operating system to allocate space, and returning space to the operating system when the program no longer needs it. G2 does not manage most memory in this way, because calling the operating system to allocate or free memory can cause waits of arbitrary length while the system processes the call. Such waits can impair real-time performance.

When G2 begins execution, it obtains an allocation of memory from the operating system. G2 thereafter manages memory internally. G2 does not obtain additional memory unless it has exhausted its allocation and needs more. G2 never returns memory to the operating system under any circumstances, including clearing the current KB or loading another KB.

G2 uses many strategies internally to minimize the time spent managing memory. For example, when you delete a KB item, G2 does not return its memory to a generic pool, but retains it for future reuse if you create a similar-sized object. Due to this technique, a G2-meter that shows memory usage never goes down, no matter how many items you delete.

Memory Management Problems

If G2 has enough memory to work with, it automatically handles all other details of memory management. You do not need to select or tune the strategies that it uses, and there is no way for you to do so. Correct G2 memory management consists of one thing: ensuring that G2 never runs out of memory while executing a KB.

Two problems can cause G2 to require additional memory during KB execution: insufficient memory allocation and unlimited memory consumption.

Insufficient Memory Allocation

Insufficient memory allocation exists when a KB's memory requirement is bounded, but the initial allocation of memory was insufficient to provide it. For example, a KB that creates many new items, or keeps long histories, needs increasing amounts of memory over time. If this memory was not allocated in advance, G2 will have to request more as the KB executes. After the KB has reached its maximum size, G2 will not request any more memory.

The problem in this case is not with the KB, but with the size of the initial memory allocation. This chapter tells you how to gauge a KB's maximum memory requirement, and how to allocate the needed memory.

Unlimited Memory Consumption

Unlimited memory consumption exists when an executing KB creates and retains an indefinitely increasing number of items, symbols, text strings, procedure invocations, or other things that occupy memory. Their accumulation will eventually exhaust any possible preallocation of memory. G2 will then have to request more, which will again be consumed, and so on until no more can be obtained.

A KB that requires indefinitely increasing amounts of memory has a **memory leak**. The problem in this case is with the KB itself: it is fundamentally incorrect, and must be tested and changed as needed to eliminate the problem. This chapter describes the necessary techniques.

Memory Management During Development

Since G2 can obtain more memory as it runs, and KB development often does not involve real-time testing until late in the development cycle, developers often neglect memory requirements while they develop a KB.

If you are aware of memory requirements throughout KB development, you will develop an understanding of how the KB's requirements vary with different conditions. This understanding can help you to determine how much memory to allocate when the KB is used for real-time processing.

G2 Memory Regions

To understand G2 memory management, you need a general understanding of how G2 uses the memory that it receives from the operating system.

When you invoke G2, the operating system provides the memory necessary for the G2 process to run. This memory comes in two blocks: one for code, and one for data.

Code memory holds G2's executable code, a table of constants, and a stack. You do not need to understand code memory internal organization, and you cannot control code memory allocation: G2 and the operating system automatically handle everything relating to code memory.

Data memory holds everything relating to the particular KB that G2 is executing. G2 memory management consists almost entirely of managing data memory. When this chapter refers to memory without qualification, the reference is to data memory only.

G2 subdivides its data memory into two regions, named Region 1 and Region 2. Each of these regions holds a particular kind of data:

- **Region 1**: Items, non-symbolic values, and a cache for workspace and icon background images.
- **Region 2**: Symbols and related internal data.

When you start G2, you can accept default sizes for these regions, or you can preallocate memory to either or both of them as needed by your KB. G2 cannot swap memory between these regions, so exhausting the memory available to one region will cause G2 to request more memory even if there is unused memory in the other.

The background image cache that exists in Region 1 acts like a separate region in many ways. When it must be referred to separately, it is called **Region 3**. Since Region 3 is a subset of Region 1, you cannot explicitly preallocate Region 3 memory: the Region 1 preallocation supplies Region 3 also. However, you can specify a maximum size for Region 3 when you start G2. This maximum limits the size of the background image cache.

Measuring G2 Memory Usage

There is no simple way to determine in advance how much memory a particular KB will need, because the maximum requirement depends on what the KB actually does as it runs. Therefore, determining the correct amount of memory to allocate to G2 for use with a particular KB requires measurement rather than calculation.

The general technique is to run the KB until it reaches its maximum memory allocation, then find out how much memory it has obtained for each region. Since

G2 never returns allocated memory except for memory for icons and fonts, the figure for each region is the maximum that the region needed at any time during KB execution. With this information, you can:

- Obtain additional RAM as needed to prevent excessive paging.
- Preallocate the needed memory to each region when you start G2.

The KB can then execute without thrashing, and will not need additional memory as it executes.

Generating the Maximum Memory Allocation

The best technique is to run your KB until you believe it has done everything it will ever do when used, specifically, until it has:

- Created the maximum number of items.
- Created every symbol the KB will ever use.
- Created the maximum number of strings of all lengths possible in your KB.
- Displayed every workspace that it will ever display while running.
- Filled every history.
- Invoked each procedure the maximum number of times that the procedure will ever run concurrently.
- Executed all for any rules for a stable number of items. (If the number of items that are iterated over by a for any rule increases, the number of for any rule invocations will increase, which increases memory usage.)
- Imported the maximum number of data points per any given time interval from a G2 Gateway interface or a G2-to-G2 interface.
- Stabilized all priority queues. (Use G2-meters to measure these, as described in G2-Meters.)

Unfortunately, it is not always easy to determine when all these maxima have been reached. To increase confidence that you have obtained as much memory as

you will ever need, you can evaluate some or all of the following factors individually:

To evaluate this maximum...	Use this approach...
The number of transient items at any single time, including the messages on the message board and in the logbook. The number of procedure invocations at any single time. The number of relations.	Measure these using the expression the maximum value of *xxx*, where *xxx* is a history-keeping variable whose formula is the count of each *yyy*, and where *yyy* is the class of all transient objects you will create, the class procedure-invocation, or the class relation. You should run your knowledge base until these numbers are stable. Note that the history keeping specification in these variables may affect memory use until they fill with history data.
The number of rule instances at any single time.	This reaches its maximum when each rule in your knowledge base that involves multiple items (those containing the word any) has concurrently fired for the maximum possible number of such items.
The amount of history kept by all variables.	You can use the Inspect facility to find every variable that keeps history, and run your knowledge base until the histories are completely full.
The number of elements in all arrays.	Use the expression the sum over each g2-array of (the array-length of the g2-array)
The number of elements in all lists.	Use the expression the sum over each g2-list of (the length of the g2-list)
The total count of new symbols G2 will encounter during a process, regardless of whether all symbols are in use at the same time.	If your KB contains any statement that creates a new symbol at every call, there is no maximum number of symbols, and G2 will eventually run out of memory.
The number of messages you keep on the message board.	The maximum number of messages is set in the Message Board Parameters system table.
The number of logbook pages you keep.	The maximum number of logbook pages is set in the Logbook Parameters system table.
The number of ICP connections you make.	Activate every ICP interface and send data through it. G2 thereafter needs no more memory for ICP connections.

Measuring the Maximum Memory Allocation

When you have run your knowledge base until it has maximized its memory use, you can measure the memory that it uses in any of three ways:

- Create G2 memory meters.
- Obtain measurements from allocation reports on the console.
- Execute operating system commands that print memory usage.

These techniques differ somewhat in the nature and accuracy of the measurements they yield, as described in the rest of this section.

Measuring Memory with G2 Memory Meters

You can use a G2-meter to measure the amount of memory G2 uses in any region, or in all regions together. All G2-meter memory measurements are in 8-bit bytes. The relevant meters are:

Meter Name	Memory Measured
region-1-memory-usage	Memory used in Region 1
region-2-memory-usage	Memory used in Region 2
region-3-memory-usage	Memory used in Region 3
memory-usage	Total memory used in all regions combined

These measurements provide very precise information about memory usage, but give no information about code memory. For information on how to create G2-meters and use them to measure memory, see G2-Meters.

Measuring Memory with Allocation Reports

When G2 launches, it prints an **allocation table** that lists the memory measurements for each region. The appearance of this table varies with the system on which G2 runs, but is always similar to this:

region#	minimum	default	desired	measured
1	4,750,000	10,000,000	unsupplied	10,000,000
2	3,000,000	3,000,000	unsupplied	3,000,000
3	400,000	2,500,000	unsupplied	0
	---	---	---	---
Totals:	8,150,000	15,500,000		13,000,000

All numbers represent 8-bit bytes of memory. The columns are:

Column	Measurement
region#	The number of the region to which the row applies.
minimum	The smallest size possible for that region.
default	The size of the region if you specify no other size.
desired	The size (if any) that you specify for the region; or unsupplied.
measured	The actual size of the region.

The numbers in an allocation table may vary on different systems. The example above shows a typical allocation table when G2 starts with no explicit memory specification; hence the desired value of unsupplied for all three regions.

G2 does not assign Region 1 memory to Region 3 unless there is a specific need for it, so the measured size of Region 3 is always zero in an allocation table.

If G2 needs additional memory, it prints an **allocation message** on the console for each allocation request. The appearance of such a message varies with the system, but is always similar to this:

```
2000/03/25 08:21:16 Obtaining more memory (region 2 at 2949120)
```

The second line of the message lists the region that needed more memory, and the size in bytes of that region after the allocation. If any allocation message(s) appear for a region during KB execution, the current size of that region is given by the last such message. If no such message appears, the region continues to have the size indicated in the measured column of the allocation table.

If the memory allocated to G2 was sufficient, no Region 1 or Region 2 allocation messages appear during KB execution. Since Region 3 is actually a cache within Region 1, Region 3 allocation messages never appear. If Region 3 needs more memory than Region 1 can provide, Region 1 obtains more memory and assigns it internally to Region 3.

Allocation tables and allocation messages are collectively called **allocation reports**. The measurements obtained from allocation reports are less precise than those obtained via memory meters, because G2 may not actually have used all of the memory that it was allocated. As with memory meters, allocation reports give information only about data memory, not about code memory.

Measuring Memory with Operating System Commands

Operating systems typically offer various commands that tell you how much memory executing programs use. Such measurements include all memory allocated to a process for any purpose, but do not show any subdivisions. The relevant commands are:

- **UNIX:** `ps -l`
- **Windows:** The Process tab of the Task Manager

Unlike G2-meters and allocation reports, operating system memory measurements include both code and data memory. Depending on the platform, they may or may not also include the memory G2 uses for storing graphics. See your system documentation for information on these commands, and to identify the analogous command (s) on other platforms.

Note On some UNIX systems, measurements printed by `ps -l` omit memory that has been allocated but has never been used. However, if your KB has done everything it ever will do, G2 will have used all memory it needs to use, and this omission will not distort `ps -l` measurements.

The measurements obtained with operating system commands are useful for determining how much RAM you need, but cannot be used to decide how much memory to allocate to the two memory regions. Use G2-meters and/or allocation reports to obtain that information.

Monitoring Instance Creation Count

Historically, G2 has had an upper limit on the number of instances that it could create in a single session, both explicitly and as a side-effect of other actions. In previous releases, this limit was $2^{29} - 1$, which was the largest integer that G2 could represent. In some cases, this limit is known to have been reached, in which case G2 behaves unpredictably.

To address this problem, beginning with G2 Version 6.0, this limit was increased to 2^{58} - 1, an increase of more than eight orders of magnitude over the previous limit, which allows for virtually unlimited object creation.

G2 Version 5.1 Rev. 9 introduced a g2-meter named instance-creation-count-as-float, which allows you to monitor the creation of instances. The g2-meter is also available in G2 Version 6.0 and higher; however, you should no longer encounter this limit and, therefore, should not need to use the meter to monitor the creation of items.

In addition, G2 issues warnings on the console or to the log file on Windows platforms when 95, 96, 97, 98, and 99% levels of utilization are reached.

Determining Region 1 and Region 2 Memory Requirements

To determine how much memory to preallocate to Regions 1 and 2, use the measurement techniques listed under Measuring G2 Memory Usage. If the memory usage in either region never stops growing, see Causes of Unbounded Memory Requirements and Correcting Unbounded Memory Requirements.

When your KB uses memory correctly, and your measurements have satisfactory accuracy and reliability, follow the guidelines in this section and the instructions in Specifying G2 Memory Allocation.

Excess Memory Preallocation

G2 uses no more memory than it needs to represent all the objects in a running KB, so preallocating more memory than a KB needs does not increase performance. Neither does it degrade performance, because virtual memory systems page out unused memory and never page it in again.

However, needless memory preallocation should be avoided because it wastes memory that other programs could use, and inflates the figures for G2 memory usage obtained with operating system commands. Such inflation can produce the impression that more RAM is needed, when the real problem is that the preallocation should be reduced.

Safety Factors

To be sure that unexpected events will never require more memory that any test run showed to be necessary, you should add a safety factor of at least 10% to the memory sizes that you obtain by measurement. If your KB has a history of unexpected increases of memory requirement, the safety factor should be even higher.

Allocating Less Than the Default

The allocation table that G2 prints when invoked shows the default size of each region. If your KB needs less memory in any region than G2 provides by default, you can preallocate less memory than the default. However, you cannot allocate a region less than the minimum memory size shown for the region in the allocation table. G2 increases any such specification to the minimum value.

Restricting Region 3 Memory

Since Region 3 is a cache within Region 1, preallocating Region 3 memory is a special case of preallocating Region 1 memory. Sufficient Region 3 memory is available when no Region 1 allocation messages appear during KB execution.

Region 3 exists to cache workspace and icon background images. Such images require large amounts of memory. Every time a background image is rescaled, G2 computes a new image of the appropriate size and stores it in Region 3. If Region 3 becomes full, G2 does not obtain more memory, but recycles the memory already available.

If you have many background images and/or frequently rescale them, and find that background images display too slowly, you might obtain faster display by increasing the maximum size of Region 3, thereby reducing time spent recycling and recalculating background images. To prevent run-time allocation from resulting, be sure to increase the preallocation for Region 1 accordingly.

Conversely, if your application is nearing the limits of available memory, devotes much memory to Region 3, and can accept slower background image display, reducing the size of Region 3 can free memory for other uses within Region 1, or allow its preallocation to be reduced.

G2 does not assign Region 1 memory to the Region 3 cache unless the memory is actually needed, so a Region 3 maximum need not be reduced just because it is greater than necessary. It only needs to be reduced when Region 3 is actively using more memory than is desired.

Specifying G2 Memory Allocation

If you do not explicitly specify the size of Region 1 or 2, G2 initially gives the region the default size shown for it in the memory allocation table. If you do not explicitly specify the size of Region 3, G2 restricts its size to be to be the default size shown for it in the table.

You can override the default memory size for any region. Though the meaning of a memory size specification is different for Region 3 than for Regions 1 and 2, both types of specification use the same syntax. You can specify memory size in two ways:

- Give arguments to the G2 command when you invoke G2. This technique is the same on all platforms.
- Set environment variables before you invoke G2. This technique differs somewhat on different platforms.

This section shows you how to use both of these techniques. Additional reference information appears under:

- rgn1lmt
- rgn2lmt
- rgn3lmt

Specifying Memory in the G2 Command Line

The syntax for specifying memory in the command line that invokes G2 is the same on all platforms, regardless of the syntax characteristic of the host operating system, because G2 processes its own command-line options.

When you specify memory in the command line, the specification overrides any specification given by an environment variable, and applies only to the particular G2 invocation.

The memory specification for each region is controlled by a separate command-line option. Any or all of these can be given at each G2 invocation.

`-rgn1lmt`: The number of bytes to preallocate to Region 1

`-rgn2lmt`: The number of bytes to preallocate to Region 2

`-rgn3lmt`: The maximum number of bytes in Region 3

To specify memory using command-line options:

➔ Invoke G2 by executing:

```
G2 [-rgn1lmt size] [-rgn2lmt size] [-rgn3lmt size]
```

where *size* is the number of 8-bit bytes to specified for the region. Do not include commas in *size*.

For example:

```
G2 -rgn2lmt 6000000 -rgn3lmt 6000000
```

specifies that Region 2 is to be allocated 6 MB when G2 is invoked, and Region 3 is to be limited to 6 MB, irrespective of any environment variable that may be defined for either region. Since Region 1 memory is not specified on the command

line, if an environment variable exists for Region 1, the variable will determine the region's size. If not, Region 1 will have the default size shown for it in the allocation table.

Specifying Memory with UNIX Environment Variables

When you invoke G2 under UNIX, G2 checks the environment for the following three variables:

`G2RGN1LMT`: The number of bytes to preallocate to Region 1

`G2RGN2LMT`: The number of bytes to preallocate to Region 2

`G2RGN3LMT`: The maximum number of bytes in Region 3

If any of these is defined, and no contradictory specification appears as an argument to the G2 command, G2 specifies the number of bytes indicated by the variable for the corresponding region.

To specify memory using a UNIX environment variable:

➔ Define the variable by executing:

```
setenv G2RGNnLMT size
```

where:

n is 1, 2, or 3, representing Region 1, Region 2, or Region 3

size is the number of bytes to specify for Region *n*. Do not include commas in *size*.

For example:

```
setenv G2RGN2LMT 6000000
```

specifies that Region 2 is to be allocated 6 MB when G2 is invoked.

To cancel memory allocation specified with a UNIX environment variable:

➔ Undefine the variable by executing:

```
unsetenv G2RGNnLMT
```

where:

n is 1, 2, or 3, representing Region 1, Region 2, or Region 3

For example:

```
unsetenv G2RGN2LMT
```

cancels the specification in the previous example.

Specifying Memory with Windows Environment Variables

When you invoke G2 under Windows, G2 checks the environment for the following three variables:

`G2RGN1LMT`: The number of bytes to preallocate to Region 1

`G2RGN2LMT`: The number of bytes to preallocate to Region 2

`G2RGN3LMT`: The maximum number of bytes in Region 3

If any of these is defined, and no contradictory specification appears as an argument to the G2 command, G2 specifies the number of bytes indicated by the variable for the corresponding region.

To specify memory using a Windows environment variable:

➔ Define the variable by executing:

`set G2RGN`*n*`LMT=`*size*

where:

n is 1, 2, or 3, representing Region 1, Region 2, or Region 3

size is the number of bytes to specify for Region *n*. Do not include commas in *size*.

For example:

```
set G2RGN2LMT=6000000
```

specifies that Region 2 is to be allocated 3 MB when G2 is invoked.

Note You can also set Windows environment variables in the Environment section of the System Control Panel.

To cancel memory allocation specified using a Windows environment variable:

➔ Undefine the variable by executing:

`set G2RGN`*n*`LMT=`

where:

n is 1, 2, or 3, representing Region 1, Region 2, or Region 3

For example:

```
set G2RGN2LMT=
```

cancels the specification in the previous example.

Note You can also delete Windows environment variables in the Environment section of the System Control Panel.

Causes of Unbounded Memory Requirements

No preallocation of memory for a KB can suffice unless the KB's need for memory is bounded. If a KB's memory requirement grows without limit during execution, the increase will exhaust any preallocation, then consume additional memory until no more is available, forcing G2 to shut down.

Two types of problems can cause unbounded increases in a KB's memory requirement:

- The KB creates and retains an indefinitely increasing number of permanent items, symbols, incomplete calls, or other things that occupy memory.
- The KB recurrently creates transient items in procedures or methods, or in some other way, and fails to explicitly delete them.

Unnecessary Retention of Storage

Any system can exhaust memory by creating ever more things and retaining them indefinitely, or by initiating ever more actions and failing to complete them. G2 does not differ from other systems in this regard.

Failure to Delete Transient Items

To understand why failure to delete transient items is a problem, you must understand how G2 manages the memory that such items occupy.

Some programming environments allow you to obtain storage as needed and abandon it when you are done with it. Such an environment includes a capability that periodically identifies all abandoned storage and reclaims it for reuse. Such reclamation is often called **garbage collection**.

Garbage collection entails moving data as needed to consolidate free memory into a single block. Such consolidation must be done carefully, or concurrent changes to memory will result in data loss. Some garbage collection algorithms suspend all other processing while they consolidate memory. Other algorithms are incremental, but this just distributes their overhead more finely.

G2 does not perform garbage collection, because pausing or slowing to do so would impair real-time performance. After G2 uses a block of memory, it does not return the memory to a generic pool, but retains it intact for future reuse when G2 again needs a block of that size. This technique manages memory with minimal impact on performance, which helps G2 provide real-time response.

When G2 allocates memory, as for a procedure call, it knows how much it has allocated, so it can reclaim and reuse the memory after the procedure returns. However, G2 does not attempt to manage memory that it allocates when a procedure creates transient items, because such management would slow performance.

When a procedure or method creates transient items (create action) but does not delete them (delete action) before returning to its caller, the item continues to exist after the procedure returns. Unless something else deletes the item, the memory that it occupies remains unavailable until you reset G2. Chronic failure to delete transient items causes an insidious loss of memory as G2 executes. Such a problem is called a **memory leak**.

G2 also creates transient items when you pass objects between different G2's, using the G2-to-G2 interface, or between G2 and a G2 Gateway bridge. To prevent memory leakage, you must explicitly delete all such items when you are done with them.

Correcting Unbounded Memory Requirements

Unexpected increases in KB memory requirements can occur in three different ways. In order of frequency, these are:

- The KB takes longer than expected to reach a memory requirement that actually is bounded.
- The KB contains an error that wastes or leaks memory.
- The current release of G2 contains a bug that leaks memory.

If your KB appears to require indefinitely increasing memory as it executes, you have two options:

- You can obtain a diagnostic KB from Gensym Customer Support. This KB performs automated tests that find most memory problems.
- You can use the techniques described in this section to perform the same tests manually.

You will probably find it easier to use the diagnostic KB, because making the changes required to perform the tests manually can be time consuming. Do not hesitate to call Gensym Customer Support at (781) 265-7301 (Americas) or +31-71-5682622 (EMEA) if your KB experiences unbounded memory increase.

The diagnostic KB contains complete instructions for performing the tests that it provides. However, you will find the KB easier to use, and the data it provides easier to interpret, if you understand the tests that it performs. Therefore you should read this section even if you intend to use the diagnostic KB.

Caution Some tests that diagnose memory problems change a KB in ways that slow down its execution and are difficult to undo. When you use the diagnostic KB, or manually apply any technique listed in this section, be sure to work with a copy of your KB, not the original.

Checking Region 1 Memory Increases

If your KB appears to experience unbounded memory increase in Region 1, check each of the following.

Accumulating Items

To determine whether your KB is accumulating items, put up a display of the count of each item. If this increases with time, your KB is creating items and not deleting them.

To detect accumulations of permanent items, put up a display of the count of each *class-name* for every class whose instances may be accumulating. If you detect any class that accumulates without limit, use the Inspect facility to locate statements that instantiate that class, and correct the error(s).

To detect accumulations of transient items, use the Inspect facility to locate statements that create them, and rewrite the KB to explicitly delete them after it is done with them.

For more information, see Monitoring Instance Creation Count.

Non-Returning Procedures

G2 allocates memory when it invokes a procedure, and reclaims that memory when the procedure returns. If a procedure never returns, the procedure invocation memory will not be reclaimed. If such a procedure is invoked recurrently, unbounded memory increase results.

To detect such a problem, first be sure that uninterrupted-procedure-execution-limit in the Timing Parameters system table has a low value, say 30 seconds (the default). This will cause any procedure that neither returns nor allows other processing to time out. G2 prints a message in the Operator Logbook identifying any such procedure, allowing you to locate and correct it.

Timeouts do not catch procedures that do not return because they enter wait states and never emerge. To identify such procedures, change the class-of-procedure-invocation of every procedure that allows other processing (directly or by waiting) to procedure-invocation. Then put up a display of the count of each procedure-invocation.

If this number increases, use Inspect to locate the procedures that aren't returning, then modify the KB as needed to ensure that they eventually return.

Accumulating Transient Class Definitions

G2 allows you to create transient class definitions programmatically. G2 must allocate new memory for every such definition, and this memory cannot be reclaimed. If your KB creates an unlimited number of transient class definitions, it will consume memory without bound. To test for this problem, remove the creation of transient class definitions from your KB, and see if G2 still consumes memory.

If your KB creates an unlimited number of transient class definitions, there are two possible solutions:

- Rewrite the KB so that it creates only a bounded and acceptably small number of transient definitions.
- Substitute a hierarchy of definitions that is fixed before G2 starts, eliminating the need for transient definitions.

Such rewrites are also likely to improve the KB's clarity and efficiency.

Accumulating History

In general, G2 allocates memory in one block at start time for variables/parameters that keep history with number of data points, but allocates it incrementally for variables/parameters that keep history with maximum age. The stepwise pattern in the latter could look like a leak.

Some histories kept by number of data points also show a stepwise allocation pattern, namely, text variables/parameters and quantitative variables/parameters whose histories contain a mixture of integer and float values. Given these facts, make the following changes to your KB as a test:

- Arrange for all history-keeping quantitative variables/parameters in your KB to have a full history, consisting either entirely of floats or entirely of integers, but not a mixture of the two.
- Arrange for all text variables/parameters in your KB to have a full history. Since G2 stores text by length, arrange for all text in the history to have the same length, or a finite number of different lengths.
- Use the Inspect facility to locate and reduce the maximum age of all history-keeping variables/parameters to a small interval like five minutes. Allow your KB to run past that interval.

The following Inspect command finds all variables/parameters keeping the age of datapoints:

```
show on a workspace every variable-or-parameter VAR such that
(is-contained-in-text ("maximum age", the history-keeping-spec of VAR))
```

Accumulating Process IDs

When you use a system procedure to spawn a process, G2 allocates memory to hold the process ID. G2 does not automatically reclaim this memory after the process completes. A KB that spawns an indefinitely large number of processes without reclaiming the storage holding their IDs will eventually consume all memory.

To reclaim the memory that holds a process ID, you must explicitly kill the process after it completes. To kill a local process spawned with g2-spawn-process-to-run-command-line or g2-spawn-process-with-arguments, call g2-kill-process. To kill a remote process spawned with g2-spawn-remote-process-to-run-command-line or g2-spawn-remote-process-with-arguments, use g2-kill-remote-process.

Accumulating Log Book Pages

Check maximum-number-of-pages-to-keep-in-memory in the Logbook Parameters system table. The default is 200 pages. If this number is large, G2 may not have created that many logbook pages yet. Until G2 creates the maximum number of logbook pages, it continues to use more memory for each page that it creates.

As a test, reduce the maximum number of pages to a small number like 2, and see if G2 still consumes more memory.

Accumulating Message Board Entries

Check maximum-number-of-entries in the Message Board Parameters system table. The default is 10 entries. If this number is large, G2 may not have created that many messages yet. Until G2 creates the maximum number of messages, it continues to use more memory for each message that it writes.

As a test, reduce the maximum number of entries to a small number like 2 to see if G2 still consumes more memory.

Generic Rules

G2 uses memory to create each instance of a generic (for any) rule. This could look like a memory leak if such rules are run on increasing numbers of items.

As a test, arrange to run all generic rules simultaneously on the maximum number of items for each rule. This will cause G2 to create the maximum number of rule instances it will ever create in your KB. See if G2 still consumes memory after this test.

Multiple Data Service Requests

If you are running G2 Gateway or G2-to-G2 data service, G2 uses memory to queue up requests for each data point. (G2 does not allocate memory for more than one request per data point, however.) If you have many data points, or

several groups of data points whose values are requested at overlapping times, the queueing of requests could look like a memory leak.

As a test, arrange for G2 to request information simultaneously for all data points in your KB. See if G2 memory use increases after that.

Lagging Priorities

Check the scheduler-lag G2-meters to see if any priorities are lagging. G2 uses memory to store the tasks it must run at each priority. If G2 is running behind, this could appear to be a memory leak.

As a test, reduce the number of rules, formulas, and procedures active at any one time, or otherwise decrease the load on G2 so that scheduler lag no longer occurs, and see if G2 memory use still increases.

Checking Region 2 Memory Increases

If your KB appears to experience unbounded memory increase in Region 2, check each of the following.

Accumulating Symbols

To determine whether your KB is accumulating symbols, check statements containing the change the text of action and calls to the symbol function.

Every time you execute change the text of with a new value that is not a number, truth-value, or text string, the action creates a new symbol. Developers sometimes miss this case, because change the text of suggests text strings rather than symbols.

Every time you call symbol with a value it has never received before, the function creates a new symbol. For example, an expression like symbol ("TEMP-[the current time]") uses different text in each call. Recurrent execution of such an expression causes unbounded memory consumption.

Accumulating Text Strings

G2 stores text strings grouped by the number of characters in the text. If your KB intermittently creates text with new lengths, or uses an indeterminate number of text strings of the same length at the same time, memory consumption increases without limit.

Run your KB until you are sure that it has created the maximum number of text strings that will exist simultaneously at each length you expect. Or, as a test, alter the text-manipulation machinery in your KB to produce text of a single length. After taking either of these steps, see if G2 memory use still increases.

If All Else Fails

The tests provided by the diagnostic KB described in Correcting Unbounded Memory Requirements and the tests described in this section can help you resolve almost any problem that causes unbounded memory increase. If the problem persists, you should produce a series of statistics files that contain details of memory use for examination by Gensym Customer Support.

You can use these system procedures to generate G2 statistics:

- g2-measure-memory
- g2-write-stats

For details on these procedures, see Memory Operations in the *G2 System Procedures Reference Manual*.

Caution For very large applications, generating memory statistics can take a very long time to execute, effectively requiring you to kill the G2 process. In addition, you should be aware that memory statistics can be very inaccurate.

To create statistics files:

1. Start with a version of your KB that you have fully tested with the diagnostic KB or the techniques described in this section.
2. Run the KB until you believe that it should not require any more memory (all histories filled, logbook pages at maximum, and so on).
3. Choose Save KB from the Main Menu.
4. Use Control + x to delete the pathname from the dialog that appears.
5. Enter write g2 stats as *pathname*, where *pathname* is the name of the statistics file to be written. If you supply a filename, G2 puts the statistics file in the directory that holds the executing KB.
6. Click End to save the statistics file.
7. Similarly produce several statistics files at intervals that make the problem apparent, say once every half hour.
8. After you have written all statistics files, save a copy of the KB itself.

Note that you do not need to pause a KB in order to gather statistics on it. Pausing the KB would yield statistics that all reflect the same instant, while letting it run yields statistics gathered over a brief interval, but the difference is insignificant for diagnosing memory leaks.

When the statistics files are complete:

➔ Contact Gensym Customer Support.

What's inside the statistics file:

1. Overall Memory Usage Statistics
2. System Object Pool Usage Statistics
3. LRU Queue Statistics
4. Short Simple Text String Pools
5. Long Simple Text String Pools
6. Adjustable Text String Pools
7. Byte-vector-16 Pools
8. Simple Vector Pools
9. Frame Vector Pools

Meaning of some columns in the statistics file:

- **Out**: the number of objects currently being taken "out" of the pool for actual use.
- **Out%**: the percentage of used objects in the pool (= Out / Count * 100)
- **Count**: the total number of allocated objects from OS for the pool, this number will never decrease
- **dCount**: the changes (difference) of Count since last time you write G2 stats (= Count - lastCount)
- **Memory**: the total memory (in bytes) of all objects allocated in this pool, (for Type CONS, Memory = Count * 8 in 32-bit G2, or Count * 16 in 64-bit G2)
- **dMemory**: the changes (difference) of Memory since last time you write G2 stats (= Memory - lastMemory)

56

Task Scheduling

Describes the G2 scheduler, the G2 clock, and task queues.

Introduction

The **scheduler** directs task processing in G2. While a user never interacts with it directly, the scheduler controls all of the activity that the user sees, as well as many of G2's background activities.

G2 activity occurs within a main processing cycle, of which the scheduler is a major part. The scheduler is the G2 time keeper and task master; it is responsible for:

- Scheduling and prioritizing all tasks
- Executing tasks between clock ticks
- Ticking the G2 clock

The remaining sections describe G2's main processing cycle and the scheduler's functions.

The Main Processing Cycle

All KB processing occurs by G2 performing a set of tasks within a continuous **main processing cycle**. The scheduler is a part of the main processing cycle.

The main processing cycle is constantly active as long as G2 is running as a process on your system. For instance, when you first start G2, it is the processing cycle that enables G2 to respond to a mouse click in the background area and display the Main Menu. In G2, a mouse click is a user interface request, one of the requests that the processing cycle services continuously.

Ticking the G2 Clock

Tasks are time-related, they are either executing currently or awaiting execution at a future time. For G2 processing, time passes according to the **G2 clock**.

A **clock tick** is the fundamental unit of time within G2. The scheduler advances the G2 clock and, between every clock tick, completes tasks scheduled for that particular time segment.

Caution The G2 clock has a limit of 17 calendar years. Reaching that limit, for example when using the as fast as possible mode for simulation purposes, will abort G2.

The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices.

The minimum-scheduling-interval attribute of the Timing Parameters system table determines the interval between the scheduling of each clock tick. The value of the attribute is either a specific time interval or continuous. The default value is 1 second.

Within the main processing cycle, the scheduler constantly checks to see if it is time to tick the G2 clock. If it is not time to tick, the scheduler continues processing any outstanding tasks in the task queue. If it is time, the scheduler ticks the G2 clock and continues the subsequent tasks in the processing cycle.

The G2 clock is completely separate from the computer's system clock, known as the **real-time clock**, even though both may be set to the same time and advance at the same rate as the real-time clock.

Major Events in the Processing Cycle

These are the six major events that occur during each processing cycle. Shaded events are those that the scheduler completes.

1	Check to see if it is time to tick the G2 clock based on the values of the minimum-scheduling-interval and the scheduler-mode attributes, and the current real time.
2	Schedule any tasks from the future time queues for the current clock tick. All tasks are placed in the current task queue according to their priorities.
3	Execute tasks in the current task queue. Any tasks not completed are deferred to later in the current clock tick or to the next.
4	Service network packets. G2 sends and receives network packets during each processing cycle. Any packets not serviced during the current cycle are deferred to the next processing cycle.
5	Service user interface requests, including drawing and mouse events. G2 receives input from and sends output to the user interfaces of all users logged into the current G2 process, including any Telewindows users.
6	Prepare to loop. G2 checks to see if it was active in this clock tick. If it was, it returns to step 1. If it is not yet time to tick, G2 goes directly to step 3 (or 4) to complete any tasks. If G2 was not active and there are no tasks left to execute, G2 idles its process for up to one second, or the time of the next future task, whichever is shorter.

The G2 Scheduler

The G2 real time scheduler manages a set of tasks which execute atomically without preemption. Tasks are selected by G2 time and priority, strictly and in that order. G2 time is driven by the scheduler mode you specify in the schedule-mode attribute of the Timing Parameters system table. You can specify one of three modes:

- **Real Time**: G2 time is synchronized with the operating system time.
- **Simulated Time**: Time intervals are allowed to expand in order to accommodate more tasks than could fit in a given interval.
- **As Fast As Possible**: All tasks are allowed to complete in the desired interval. G2 does not attempt to synchronize with the operating system.

The G2 scheduler breaks tasks into two main categories, current tasks and future tasks. Current tasks are tasks which are ready to run immediately. The G2 start action start p1() will enqueue a current task. Future tasks are tasks that should be executed with some non-zero delay. The G2 start actions start p1() after 10 seconds will enqueue a future task.

G2 attempts to execute all current tasks before any future tasks. One could also think of current tasks as future tasks with a delay of 0 seconds. The set of current and future tasks forms the outermost sorting for the G2 scheduler.

At a given G2 time (including the current time) priority is used to resolve scheduling conflicts. G2 has 12 priorities for task execution, 0 through 11. Priorities 1 through 10 can be used by user tasks, and by system tasks, for example, G2 internal tasks. Priority 0 is reserved for system tasks that must occur before user code runs. Priority 11 is reserved for low priority system tasks that should not be given favor over any user code.

Priorities usually come from the attributes of computable G2 Items. They can also come from overrides in explicit task creation, for example, start p1() at priority 7. Priorities are contagious through recursion. For example, when p1 is started at priority 4 and calls p2, which has a default priority of 9, if p2 enters a wait state, its resumption is scheduled at priority 4.

Wait States

A computational task is either running or in a wait state in which all wait states are equal. Whenever a task enters a wait state, the G2 scheduler determines the next task to execute by using the methods described above.

Most G2 tasks make use of time slices to help determine when to go into a wait state. By default, time slices are 100 milliseconds in G2 4.x and 50 milliseconds in G2 5.0 and higher. Because G2 is not pre-emptive, the time-slice interval can vary to some degree.

Task Scheduling

G2 has four principal kinds of tasks: computation tasks, network tasks, UI tasks, and miscellaneous internal tasks. Because G2 is non pre-emptive, the scheduled interleaving of tasks is completely determined by the conditions that cause tasks to be placed on the schedule, to go into wait states, or to complete.

Computation Tasks

Computation tasks execute G2 procedures, methods, expressions, and rules.

Procedures and methods will create a task if called from a local or remote start, or from a remote call. Expressions are normally embedded in other items and are run periodically.

The scheduling of rules (the actions in the rule consequent) is more involved:

1. If the conditions in the antecedent of a rule are met, a rule context is created that contains the set of items that satisfy the conditions.

   For example if G2 has the rule whenever the level of any tank receives a value then start p1(), and some other part of the KB concludes a value into the level of my-tank, the rule context is the set of one item, my-tank.

2. This context is combined with the rule itself to form a key into a table of all pending rule computation tasks. This table is a secondary index into the G2 scheduler data structures.

3. If the key indicates that there is currently no pending task to execute the rule, a new task is created to execute the actions in the rule consequent.

4. Otherwise, the new rule context will update any values in the existing rule task.The rule-context is then discarded and no new rule task is created.

Wait States in Computation Tasks

Embedded expressions do not go into wait states. A procedure or method task runs until an unhandled error occurs, the task completes, or a statement executes that causes a wait state. The following statements can cause a wait state:

- wait *for some condition or period of time*

  Waiting for 0.0 seconds does *not* cause a wait state.

- allow other processing

  Allows the task to wait if its time slice has expired, or if a higher-priority task is ready to run.

- call P across (remote call)
- collect data
- do in parallel [until one completes]
- on error

  Causes a wait state in G2 Version 4.0, but *not* in Version 5.0 and higher.

Rules execute their consequents in order or in parallel. Wait states can happen in scheduled rule tasks in order to obtain the values of variables used in the consequent. The actions themselves are atomic.

Network Tasks

Network tasks handle network messages and perform connection management, remote procedure calls, and data service. The receipt of remote calls or starts always enqueues tasks. Data-service requests are enqueued in a separate queue to be handled at the priority of data service specified in the Data-server-parameters

system table. The network task itself limits itself to a time slice whose expiration is checked after handling all requests for a given socket.

The data service task runs in a loop, performing each data service request. After each tasks is atomically handled, two things are checked:

1. The data service task will put itself into a wait state if its time slice has expired.
2. The G2 scheduler is queried to see if any new higher-priority tasks have been enqueued as a result of processing the data service request.

For example, a gsi-return-values request could cause the invocation of some rule. If such a task is found, the data service task goes into a wait state.

UI Tasks

Input handling is done in time slices that are distributed round-robin to the G2 windows. Input handling tasks do not execute KB code directly, but schedule the execution of procedures and the rule consequents for buttons and user-menu-choices.

When the Drawing-parameters system-table attribute, allow-scheduled-drawing is set to yes, UI output is scheduled as well, instead of being done in the execution of state changes in G2 directly.

Miscellaneous Internal Tasks

Saving KBs, printing, and trend-chart updates, are examples of other tasks that the G2 scheduler manages. Most of these tasks time themselves out after their time slice expires.

Procedural versus Rule-Based Tasks

KB writers have synchronous (procedural) and asynchronous (rule-based) means of reacting to changes in the outside world. The former is straightforward but sometimes requires more code. The latter can be more elegant, but exposes the developer to some subtle interactions between rules, certain procedure statements, the G2 scheduler, and data service. Care must be taken to ensure that rules are given the opportunity, through wait states and priority, to fire at the appropriate times.

Default Task Priorities

The default priorities of some common G2 tasks are:

Task	Priority	Comments
Action and update buttons	2	Includes radio buttons, check boxes, type-in boxes, and sliders.
Remove or unhighlight messages	2	
Update displays (readout tables, meters, dials, graphs, trend-charts, charts, and freeform-tables) Graphs are a superseded capability. For more information, see Appendix F, Superseded Practices.	2	Updating screen displays consists of two parts. First, the value of the display is found; second, the screen display is changed. You can override the default in an individual display's attribute table.
Complete data service	4	When G2 cannot receive all of the values from a data server in the time that is allotted, the scheduler schedules a task to finish reading that data. This task has priority 4. You can change this priority with the priority-of-data-service option in the Data Server Parameters system table.
Update variable values	4	Variable values are obtained through backward chaining and other data-seeking.
Invoke rules	6	You can set an individual rule's priority in its attribute table.
Start procedures	6	You can set a procedure's priority in its attribute table.
Reply to outside requests for data from G2	6	

Task	Priority	Comments
Detect variable failure, and retry variables	8	For a description of failed variables, see Handling a Variable Failure.
Drawing and printing	8	You can change the printing priority using the printing-priority attribute of the Printer Setup system table. There is no way to change drawing priority.
KB Save operation	8	The priority for saving a KB using the KB Save option on the Main Menu. The priority

The scheduling priority 8 is considered a background process. Whenever tasks are scheduled at a higher level priority, such as rules and procedures, background tasks with a priority level of 8 proceed slowly.

Optimizing Task Scheduling

Optimum task scheduling requires a fine balance between:

- The total number and management of future time queues.
- The scheduling interval, which determines the amount of time G2 has to perform tasks between each clock tick.

Two factors determine the number of future time queues:

- The number of tasks scheduled for future processing.
- The scheduling interval.

The number of tasks scheduled for future processing is an arbitrary value, determined by the individual requirements of a KB. You control the scheduling interval by adjusting the time between clock ticks in the minimum-scheduling-interval attribute.

You can set this attribute in one of these ways:

- A scheduling interval, which may be subsecond or not.
- Continuous mode.

The scheduler runs in one of three modes: real time, simulated time, and as fast as possible. Each of these modes is described in Defining the Scheduler Mode. Because simulated time and as fast as possible modes are used primarily for simulation or demonstration purposes, and this discussion centers around

scheduling for runtime applications, this section assumes that the scheduler is in real time mode.

The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices.

Using the Default Scheduling Interval

In most cases, the default 1 second scheduling interval provides an optimum balance between the number of tasks processed during each processing cycle and the number of future time queues that G2 has to manage. The default scheduling interval limits the amount of overhead in moving tasks from future queues to the current task queue.

If your KB does *not* require any subsecond processing, the 1 second scheduling interval usually supplies satisfactory KB performance. In general, we do not recommend setting the scheduling interval to a value greater than 1 second.

Using a Subsecond Timing Interval

You can set a subsecond timing interval up to .05 seconds. A future time queue exists for each subsecond interval at which tasks are scheduled, up to 20 per second.

When subsecond timing is in effect, tasks scheduled for less than a second are processed at the subsecond interval. Tasks cannot be processed at an interval less than the scheduler's setting.

For example, if you set the minimum-scheduling-interval attribute to .5 seconds, setting the default-update-interval of a variable to .05 seconds is allowable, but has no effect. G2 will update the variable at .5 seconds, since that is the smallest increment of time at which processing occurs.

During subsecond scheduling, G2 incurs some processing overhead from the additional future time queues it has to manage and the calls required to move those tasks to the current task queue. Also, a subsecond processing cycle means there is less time between clock ticks in which to execute tasks.

For KB's requiring subsecond updates, intervals or scanning, use the subsecond interval that fulfills your requirements, keeping in mind the overhead of managing more future time queues and having less time per cycle to complete tasks.

Using Subsecond Interval Expressions

When subsecond timing is in effect, the KB can use subsecond time intervals to control processing. You can specify a subsecond time interval for many interval expressions that determine when the scheduler controls a task, including:

- Scan-interval attribute in rules.
- Display-update-interval and display-wait-interval attributes in dials, graphs, meters, and readout-tables.

  Graphs are a superseded capability. For more information, see Appendix F, Superseded Practices.
- Initial-scan-wait-interval attribute in freeform table.s
- Default-update-interval attribute in variables.
- The wait for *time-interval* statement in procedures.
- The start *procedure* after *time-interval* action.
- Default-simulation-time-increment of the Simulation Parameters system table.

The G2 Simulator is a superseded capability. For more information, see Appendix F, Superseded Practices.

Using a Continuous Scheduling Interval

When a continuous scheduling interval is in effect, future time queues are created for each interval at which tasks are scheduled, and the clock ticks to tasks, rather than to a set interval. This can incur clock ticks of variable lengths, depending on when tasks are scheduled. G2 idles between clock ticks in continuous mode.

Because continuous mode ticks the clock to tasks, rather than to an interval, task processing can occur at a faster speed than the minimum scheduling interval that G2 allows, if a task is set to less that .05 seconds.

Consider the two extreme examples presented next, one of infrequent scheduling demands, once every 2 minutes, and critically fast subsecond scheduling, at one hundredth of a second.

Infrequent Scheduling Demands

If KB tasks are scheduled less frequently than once every 2 minutes, in continuous mode, the clock would tick at that speed, and G2 would idle between clock ticks.

Critically Fast Subsecond Scheduling

When using continuous mode, you can schedule tasks at a rate that is faster than the minimum subsecond interval of .05 seconds. For example, G2 permits the use of subsecond scan intervals to an infinitesimal value, though computer speed and other factors would likely preclude an unreasonable subsecond value from taking effect.

As an example, setting the scan interval of a rule to .01 seconds causes G2 to create a future time queue for one-hundredth of a second, and to place the scan task on that queue. In continuous mode, the scheduler would then have one-hundredth of a second to complete current tasks before ticking the clock to the next scheduled task, one-hundredth of a second later. While such task scheduling may be critical for some rules, it is likely not the most efficient scheduling for the remainder of the KB.

Continuous mode is useful if, for instance, you have a very small number of rules that require a high scanning frequency, and you set that frequency to a reasonable rate for G2 to be able to complete executing other tasks.

1886

Part VIII

Application Deployment

1888

57

Package Preparation

Describes removing a KB's source code and making a proprietary KB.

Introduction

Before distributing a completed knowledge base (KB) to customers and end users, you can hide source code, or limit access to certain knowledge by making workspaces proprietary.

Hiding the source code you have developed protects the integrity of the KB design and methodology from unauthorized reuse. Making workspaces proprietary limits access to certain functionality, and can be used to make an entire KB proprietary. Proprietary KBs are inaccessible without Gensym authorization codes. For a description of how to create a proprietary workspace, see Making Workspaces Proprietary.

G2 refers to the process of hiding source code or making workspaces proprietary as **package preparation**.

Preparing a KB for Customer Distribution

You prepare a KB for customer distribution by using G2's **package preparation mode.** The purpose of package preparation mode is to let you perform these tasks:

- Mark items for text stripping.
- Remove change logging and version information.
- Configure proprietary workspaces.

The processes are independent, but can be combined. You can strip the text from items, remove change logging and version information, make workspaces proprietary, or complete all three tasks.

The process of preparing a KB for distribution is one in which the original KB is irreversibly changed into a different version, which may not have source code, or which may be proprietary. To refer to these two KB versions, this chapter uses the terms *source*, referring to the original, comprehensive version of the KB, and *target*, indicating the KB version that will exist once package preparation is complete.

Saving a Copy of the Source KB

The changes that you make to a KB in package preparation mode are irreversible. Before making such changes, save a copy of the source KB in its completed form for future development.

To save a copy of the source KB:

1 Select Main Menu > Save KB.

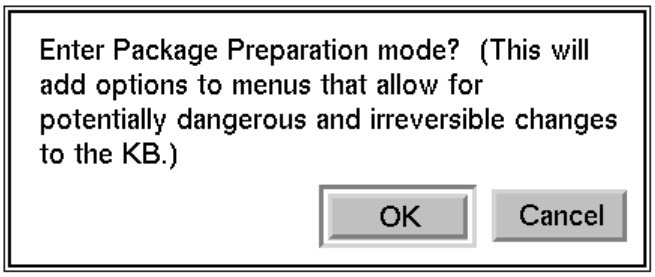
2 Enter the name of the source KB to save for future use.

Entering Package Preparation Mode

To enter package preparation mode:

1 Select Main Menu > Miscellany > Enter Package Preparation Mode.

This dialog appears:



2 Click OK to enter package preparation mode.

You can now mark items for text stripping and mark any workspaces you want to make proprietary.

Text Stripping Items

Text stripping means to remove source code from compiled attributes. Upon command, G2 strips the text of any items marked for text stripping.

When the KB is in package preparation mode, the menus of each item can include either of the following menu choices:

- Mark To Strip Text or Mark Not To Strip Text
- Remove Strip Text Mark or Remove Do Not Strip Text Mark

The menu choices in each pair are mutually exclusive. Choosing one replaces the original menu choice with another to reverse the marking, as follows:

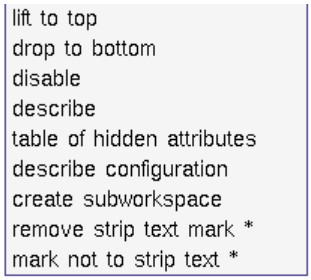
Marking an item this way...	Replaces the menu choice with...
Mark to Strip Text	Remove Strip Text Mark
Mark Not To Strip Text	Remove Do Not Strip Text Mark

You mark each item in the KB with one of the choices, so that G2 knows what to text strip. You then have G2 perform text stripping on the items on a marked workspace.

To text strip KB items:

1 Select each item in your KB and choose one of the text stripping menu choices. If you make a mistake, select the appropriate menu choice to undo it.

Marking a workspace as Mark To Strip Text changes the item menu of every item upon the workspace, and each of its subworkspaces and their items. The text stripping menu choices of each of these items appear with an asterisk next to them, such as this partial menu:

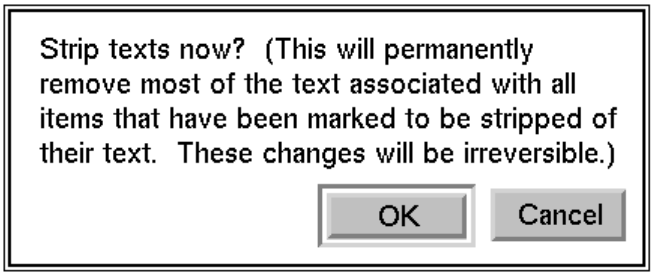


When text stripping occurs, G2 text strips all of the compiled attributes of any item upon the workspace, along with any of its subworkspaces and their items. To retain the text of any individual items upon a workspace marked for text stripping, choose the Mark Not To Strip Text option from its item menu.

2 If you did *not* save the source KB before starting text stripping, save it now before completing the target KB.

3 Pause the KB and select Main Menu > Miscellany > Strip Texts Now.

This dialog appears:



4 Click OK for G2 to text strip every marked item.

The Operator Logbook displays the messages: Stripping texts now! and Finished stripping texts to let you know when G2 has finished.

5 Select Main Menu > Miscellany > Leave Package Preparation Mode to exit package preparation mode.

6 Save the target KB with a different name than the source KB, or save the target KB in a different directory.

Removing KB Change Logging and Version Information

You can remove KB change logging and version information from a KB whenever necessary. For example, you would probably remove change logging and version information from an application before KB deployment.

Since flushing a KB of its change log and version information is a permanent and irreversible change, it is accessible only when the KB is in package preparation mode.

When your KB is in package preparation mode, these menu choices are available for flushing logging information:

- The change-log attribute submenu of rules and other definition items includes an additional choice:

  flush change log

  Choosing this option permanently deletes all entries in the item's change log.

- The kb-version-information-for-change-logging attribute submenu of the Saving Parameters system table includes an additional choice:

  flush version information

  Choosing this option permanently deletes all of the KB version information.

- The Main Menu > Miscellany menu includes the choice:

  Flush Change Log For Entire KB

  Choosing this option flushes the change log of every KB item.

Caution Choosing the Flush Change Log For Entire KB menu choice flushes change log entries for *every* module in the current KB, not just the top level module or any other single module. Be very cautious about choosing this option.

Making Workspaces Proprietary

Making workspaces proprietary confines the behavior of the workspace, its items, and their subworkspaces and items. G2 restricts the functionality of proprietary workspaces so that users cannot:

- Transfer items on or off the workspace.
- Clone items on the workspace.

- Edit the item configuration of the workspace or items upon the workspace.
- Edit the instance configuration of any class definitions upon the workspace.

In addition to these limits, G2 provides configuration statements that affect *only* proprietary items. When in effect, such configurations cannot be overridden by any user mode, including administrator mode.

Use proprietary workspaces to:

- Hide information from users and provide configurations that cannot be overridden by administrator mode.
- Create a proprietary KB that requires special authorization codes from Gensym, to use knowledge in the KB.

When proprietary restrictions are in effect, the user can load the KB; however, attempting to start the KB results in an error message such as:

> G2 is not licensed to run with the package <package>.
> G2 cannot be started or resumed.

Creating a Proprietary KB

A proprietary KB is one that requires special authorization codes from Gensym. You create a a **proprietary KB** while making proprietary workspaces, by configuring one or more workspaces with a proprietary package name.

To make a proprietary workspace:

1. Open the attribute table of the workspace to make proprietary by selecting KB Workspace > Table.
2. Edit the proprietary-package attribute of the attribute table.

   This attribute is only available when the KB is in package preparation mode. For details, see Entering a Proprietary Statement.
3. Complete the attribute with a proprietary statement.

Entering a Proprietary Statement

In the proprietary-package attribute, enter the statement for a proprietary workspace using this syntax:

> {none | not proprietary | potentially private | potentially *package-name* }

Proprietary Statement	Description
none	Indicates that the workspace inherits the value of its superior workspace.
not proprietary	Specifies that the workspace is not proprietary. When you direct G2 to make workspaces proprietary, this value changes to none.
potentially private	Specifies that the workspace is proprietary, but does not require special authorization. The term private is a system-defined package name that is always authorized. Once workspaces are proprietary, configuration statements that restrict proprietary items will be in effect.
potentially *package-name*	Indicates the name of the package for which users require authorization to use this KB. Once workspaces are proprietary, this value causes the KB to become proprietary, requiring authorization to access it.

Specifying private for the proprietary-package attribute on your workspaces lets you protect proprietary information without requiring end users to modify the `g2.ok` file for authorization. To obtain appropriate authorization codes for a *package-name* that you provide, end users must call the Production and Licensing Department at Gensym.

Tip When restricting knowledge, never make a top-level proprietary workspace non-deletable, unless you want to prevent end users from hiding one of your workspaces. Typically, end users may wish to use the parts of the KB that they created in your delivered application. To do this without authorization, they must be able to delete your proprietary workspaces.

Creating and Configuring Proprietary Items

An item is considered **proprietary** if it resides upon a proprietary workspace.

Making items proprietary is a simple way to limit item behavior within your KB. For example, to make all instances of an automobile class proprietary, do so by making a workspace proprietary, and then placing the automobile class definition

upon it. By adding a configuration statement to either the proprietary workspace or the class definition, you could then further restrict the object's behavior.

You enter a configuration for proprietary items, using the restrict proprietary items as follows configuration clause. While you can include this configuration statement in any item's item-configuration attribute, such statements take effect *only* if the item, or its definition, resides upon a proprietary workspace, or its subworkspaces or items. For example, if you click on an automobile object residing on a proprietary workspace, and see a configuration statement such as this:

```
restrict proprietary items as follows: selecting any automobile does nothing
```

then the configuration is in effect and clicking on the item does nothing. For a detailed description of configuration statements, see Configurations.

Testing a Proprietary KB before Completion

Before completing the process of making workspaces proprietary, you can simulate the behavior of your target proprietary KB or workspaces.

To test a proprietary KB before compiling:

➔ Select Main Menu > Miscellany > Enter Simulate Proprietary Mode.

This option simulates the behavior of your KB as if your computer were authorized to use the named package. You can enter and leave simulate proprietary mode as many times as you need to change the KB behavior until you are satisfied.

To exit from the simulated proprietary mode:

➔ Select Main Menu > Miscellany > Leave Simulate Proprietary Mode.

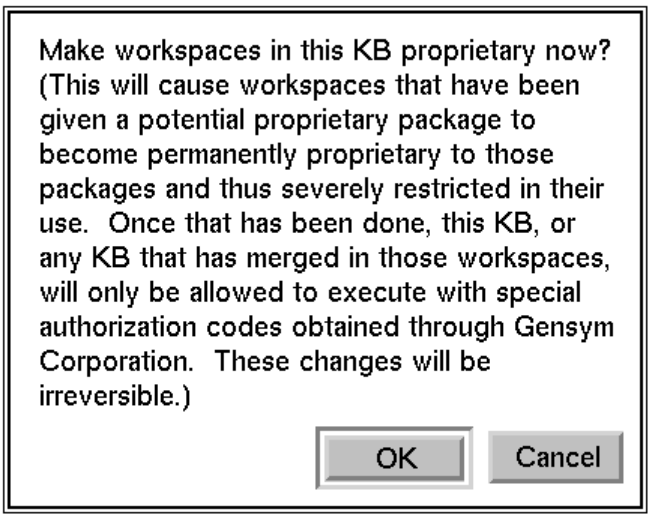
Tip You do not have to be running the KB in package preparation mode to simulate proprietary mode. You can simulate proprietary mode at any time.

Completing Proprietary Workspaces

To finish making workspaces proprietary:

1. If you did not save your source KB before starting to make proprietary items, save the KB now for future development.
2. Pause the KB and select Main Menu > Miscellany > Make Workspaces Proprietary Now.

This dialog appears, reminding you that the changes you are about to make are irreversible:



3 Select OK for G2 to make the marked workspaces proprietary.

4 Save the target KB using a different name than the source KB, or save the target KB in a different directory. The target KB is the one for distribution.

5 Leave package preparation mode by selecting Main Menu > Miscellany > Leave Package Preparation Mode.

Distributing a Proprietary Application Package

If you distribute a proprietary KB to your end users, you must register the name of the application package with Gensym. If the proprietary-package attribute or any workspace specifies the package name (instead of private or none), your end users must modify the `g2.ok` files to be authorized to use the KB.

1898

58

Licensing and Authorization

Presents licensing and authorization for G2.

Introduction

This chapter provides information about licensing and authorization in G2 and Telewindows, beginning with an overview of G2's licensing options. It describes licensing and authorization in a non-secure and secure G2. It also describes the Telewindows licensing structure.

G2 Licensing

G2 provides two types of licenses and two options for each type. Licenses are designed to provide the level of access and functionality that you, or your users and customers require. For example, as a G2 developer, you may require a fully functional, online developer's license. An end user of your finished KB, however, would typically require a license offering only minimal access to and manipulation of the KB.

G2 License Types

G2 is available in either an **offline** or an **online** license.

While the two types of licenses provide similar capabilities, they differ in their ability to access or communicate with other systems:

- A G2 offline license is intended for development environments where interfacing to external real-time data systems, or supervisory systems is *not* required.
- A G2 online license is intended for use in applications that require a high-performance interface to databases or external real-time data systems such as PLCs, DCSs, or supervisory systems. It provides offline license capabilities, with the *addition* of an external device interface.

Both offline and online licenses are available in development or deployment licensing options. A summary of the capabilities and features of both licenses is:

Licensing Capabilities and Features	Online	Offline
G2 object code for one CPU	✓	✓
Structured natural language editor	✓	✓
Object-oriented knowledge base development tools—fully portable KB	✓	✓
Integrated, animated color graphics	✓	✓
Real-time inference, procedure, and rule execution	✓	✓
Dynamic simulator and modeling	✓	✓
External C source-code interface (via foreign functions)	✓	✓
Warmboot facility	✓	✓
G2 GUIDE for building Motif-like graphical user interfaces	✓	✓
Online documentation	✓	✓
ICP application layer network protocol for TCP/IP	✓	✓
G2 file interface (GFI)	✓	✓
G2 Gateway high performance data server for live data	✓	

GFI and the G2 Simulator are superseded capabilities. For more information, see Appendix F, Superseded Practices.

G2 License Options

You can purchase one of two license options for both G2 online and G2 offline licenses:

- Development
- Deployment

You can run your KBs under either G2 license option.

The development license is the most powerful license, providing unlimited G2 capabilities and unrestricted system access. A G2 online license with the development option includes G2 Gateway and GFI. All G2 license options include **ICP**. The other license option permits KBs to run, but it limits interactive access to G2 capabilities.

GFI is a superseded capability. For more information, see Appendix F, Superseded Practices.

The functionality of each license type is as follows:

This license...	Includes...
Development	Full functionality.
Deployment	All functionality of a development licence except: • Inspect: Menus that enter the Inspect facility are disabled. • Profiling: Calls to use the profiler are disabled and signal an error. • Rule/Procedure editing: Menus and mouse gestures to enter the Text Editor on the text of rules, procedures, and methods are disabled.

Note The capability of editing item icons and attributes, available under run-time and embedded license options, refers to transient items only.

Finding License Types and Options in a KB

Within G2, license types and options appear as values of the authorized-optional-modules attribute, located in the KB Configuration system table.

Within G2, the representation of licensing options differs slightly from when you purchase licenses and options. For example, you can purchase either an online or an offline G2 license, then choose one of the two available license options, as G2 License Types explains. However, within the KB Configuration system table, the attribute values online and offline are interpreted as follows:

- When listed alone, an online or offline value means an online or offline G2 license with the development license option.
- When listed with a license option, a runtime or restricted-use and an online or offline value means an online or offline G2 license with whatever option follows.

For example, this value indicates an online license type with the restricted option:

online, restricted-use

G2 Authorization and the g2.ok File

While G2 as a product includes all license options, each Gensym customer requires both the license and the corresponding authorization codes to use each option. The `g2.ok` file provides this authorization, based on your license type. When you install G2, the installer generates authorization codes, based on your license type, and automatically updates the `g2.ok` file to use these codes. This file must exist with the proper codes in order to run G2.

Note G2 provides another authorization file named `gsi.ok` for authorizing the following bridge products: G2 WebLink, G2-Oracle Bridge, G2-Sybase Bridge, G2-ODBC Bridge, and G2-OPCLink. For more information on this authorization file, see the *G2 Database Bridge User's Guide*.

When G2 is not secure, there is no need to edit the `g2.ok` file. When G2 is secure, you must edit the `g2.ok` file, either manually or programmatically, to specify named users and user passwords. For information, see Authorizing Users at a Secure Site.

How G2 Locates the g2.ok File

Each time G2 starts, it searches for the `g2.ok` file in these locations:

- The directory from which you started G2.
- The directory indicated with either of the optional command-line options `-ok`, or `-v11ok`, which let you specify a location of your choice for the OK file.
- The directory specified in the `G2V11_OK` environment variable.

On operating systems with file versioning, G2 always uses the most recent version of the OK file.

If G2 does not find the `g2.ok` file, it is unable to start.

For information about using environment variables, see Using Environment Variables. For information about using the `-ok` and `-v11ok` command-line options, see `ok`, and `v11ok`.

Description of the g2.ok File

The `g2.ok` file is a text file containing:

- Machine authorizations for each machine that is licensed to run G2. This file includes the following information:
  - The machine name and ID.
  - The license type and expiration date.
  - The authorization codes, which are automatically generated as part of the installation process.
  - The authorized packages and their authorization codes, which are automatically generated during the installation process.
  - The maximum number of concurrent floating Telewindows clients.
  - Whether the G2 is secure.
- When the G2 is secure, the authorized named users, user modes, passwords, and password validity.

Here is a sample g2.ok file that specifies a machine authorization for a machine named my-machine. The machine authorization is not secure; therefore, the g2.ok file does not contain any named users.

```
begin g2-ok-file
-- This generated from a G2 Knowledge Base file-format-version: 2;
-- Machine Authorizations
begin machine
   name: my-machine;
   machine-id: "123abc45";
   authorized-products: (online jl);
   expiration-date?: none;
   authorization: (68019 2369 592 3623 311214);
   make-g2-secure?: false;
   authorized-kb-packages:
       ((gensym-corbalink-runtime-v1 6949 317947 510403)
        (gensym-corbalink-dev-v1 483151 483480 439561)
        (gensym-activexlink from 1 oct 2002 to 25 mar 2003
          372873 339152 480929)
        (gensym-gqs5 482834 185618 511204)
        (gensym-gda4 160092 253521 111643)
        (gensym-protools5 526378 181648 32301));
   number-of-processes-authorized: 1;
   maximum-number-of-concurrent-floating-telewindows-allowed: 8;
   maximum-number-of-concurrent-floating-tw2-allowed: 8;
end machine
-- There were no named users in the KB.
-- End of file marker
end g2-ok-file
```

How G2 Uses the g2.ok File

G2 maintains a record of the g2.ok file that it uses during the launch process. It uses the file to monitor the number of licenses and license types under which developers are running. Monitoring occurs on a per G2 process basis. For example, if your company purchases two development licenses and two deployment licenses, and three developers are working with two development and one deployment license, a fourth developer could use G2 in deployment mode, but not in development mode.

Authorizing Users at a Secure Site

A G2 site is secure when the make-g2-secure? attribute of the g2.ok file that authorized G2 to run has the value true. At a secure site, a user must provide a password before G2 grants access to a KB. User names and passwords are stored in the g2.ok file that authorizes G2 execution.

Passwords are stored in encrypted form. In most cases, encrypted passwords generated on one platform will work on other platforms, but such transportability

is not guaranteed, because differences between platforms can affect the encryption algorithm.

Note G2 has a single built-in user mode: Administrator. To start secure G2 as a specific user, a KB with that user defined must be provided. See kb, for information about using the -kb command-line options.

How G2 Uses a Secure g2.ok File

When the g2.ok file indicates that a G2 process is secure, G2 displays a login dialog when there is an attempt to login to G2. G2 uses the g2.ok file data to validate the user name and mode information.

For each user that connects with G2, G2 dynamically creates a g2-window item and associates it with the user. For each authorized user, G2 sets the following g2-window item read-only attributes to true:

- g2-window-user-is-valid
- g2-window-mode-is-valid

Secure G2 OK File Syntax

The syntax of an OK file that specifies a secure G2 is as follows:

```
begin g2-ok-file
   version-element
   machine-element
   [machine-element]
   user-element
   [user-element]
end g2-ok-file
```

The machine elements are syntactically the same as in a non-secure G2: the only difference is that the make-g2-secure? attribute of the machine element is true. Each user element authorizes one user to access G2. The syntax of a user element is:

```
begin user
   name: name;
   password: "";
   permitted-user-modes: (list-of-modes);
   password-validity: numer-of-days
end user
```

where:

name: A symbol that is the user name of the user authorized by the element.

list-of-modes: A parenthesized list of symbols that specify user modes that the user can set in G2. The modes are separated by blanks. Do *not* separate the modes with commas.

number-of-days is the number of days since Jan. 1, 1900 that the password should remain valid, where 1/1/1900 is day 0. For example, to configure the password to expire on Jan 1, 2006, use 38690 (365 x 106).

The value of `password` is initially an empty string. The `g2passwd` program fills in an encrypted value, as specified under Specifying a Password in a G2 Authorization File.

Version Element

The version element allows files produced for different versions of the OK file parser to be distinguished. For G2 Version 5.1 and later OK files, the version is 2. The version number is not the same as the G2 version.

The syntax of a *version-element* is:

```
file-format-version: integer;
```

For example, the *version-element* in a G2 Version 2011 OK file is:

```
file-format-version: 2;
```

User Name and Password Syntax

Both the user name and the password that identify an authorized user of a secure G2 must have the following syntax:

- The first character is alphabetic.
- All subsequent characters are alphanumeric and can include hyphen, underscore, period, and question mark characters.
- Alphabetic characters are case insensitive.

If these requirements are not met, the user will be unable to use G2 until the error is corrected.

Secure G2 OK File Example

An OK file with two machine elements and two user elements could look like this before passwords were added:

```
begin g2-ok-file
   begin machine
--     Authorization Information Appears here
   end machine

   begin machine
--     Authorization Information Appears here
   end machine

   begin user
      name: kanti;
      password: "";
      permitted-user-modes: (user);
   end user

   begin user
      name: john;
      password: "";
      permitted-user-modes: (administrator developer);
   end user
end g2-ok-file
```

Adding User Elements to the Authorization File Interactively

To add a user element to an OK file, you must first determine the user name and the user mode(s) that the user can set.

To add user elements to the authorization file interactively:

1. Open the `g2.ok` file using any text editor.
2. Edit the `make-g2-secure?` attribute to be `true`.
3. Add the necessary element(s), using the syntax given previously.

Specifying a Password in a G2 Authorization File

Gensym provides a program, `g2passwd`, for setting or changing a user's password. This program is in the same directory that holds the G2 executable file. Before a user can set or change a password, a user element that names the user must be defined in the relevant OK file, as described previously in this section.

Caution Do not attempt to set a password except via `g2passwd`, as described in the following instructions. A password entered literally would be unencrypted, and would not match the encrypted password that G2 uses for validation.

To specify or change a password:

1. Select or launch a command-line interpreter.
2. Connect to the directory that contains the G2 executable file.
3. Execute:

   `g2passwd` *ok-file-name*

   where *ok-file-name* is the pathname of the relevant OK file.
4. Supply information in response to the program's prompts, as follows:

Prompt	Response
User Name	Enter the name of the user whose password is to be set or changed. The password must have the syntax described under User Name and Password Syntax.
Current Password	Enter the existing password of the user. If no password has previously been registered, press Return without entering anything.
New Password	Type the password to be set in this invocation of `g2passwd`. The password must have the syntax described under User Name and Password Syntax.
New Password Again	Retype the password to verify it.

The program does not echo the characters that constitute any password typed. If no errors occur, `g2passwd` changes the user's password as indicated, then quits. If any error occurs, the program prints an error message, then quits, leaving the OK file unchanged. To try again, relaunch `g2passwd`.

Updating the g2.ok File

Site administrators can freely change the `g2.ok` file and reinstall it from within G2 without having to relaunch G2. A site administrator can update the `g2.ok` file for any of several reasons, including:

- Adding users.
- Deleting users.
- Changing the valid modes of users.
- Resetting a user's password.

An administrator can also make changes that affect the status of a user who is currently logged in. Such a change can remove:

- A currently logged-in user's entry.
- The authorized mode of a currently logged-in user.

To edit the OK file interactively:

➔ Open the `g2.ok` file using any text editor and edit the user element portion of the file, as needed.

A number of system procedures also exist for editing the `g2.ok` file. For the syntax of these system procedures, see User and Security Information Operations in the *G2 System Procedures Reference Manual*.

Adding and Deleting Users Programmatically

System administrators can use the following system procedures to add and delete users to and from the `g2.ok` file programmatically:

- g2-add-user
- g2-delete-user

Resetting a User's Password Programmatically

System administrators can use the following system procedures to reset user passwords:

- g2-set-user-password
- g2-change-password-expiration-date

If a user forgets his or her current password, a G2 site administrator can reset an existing password by changing the encoded string in the `g2.ok` file to an empty string ("").

Since a user cannot log in to G2 with a password that has no characters, the user cannot change the password online. Instead, the user must run the `g2passwd` program from a command console to enter a new password for the first time.

Any new password entered through the g2passwd program remains unusable until the g2.ok file is reinstalled and the new password information communicated to G2.

Reinstalling the g2.ok File

After changing the g2.ok file while G2 is executing, you must reinstall it from within G2 for the changes to take effect.

Reinstalling a g2.ok file updates user:

- Names
- Modes
- Passwords

G2 ignores machine ID and other g2.ok information when you reinstall a g2.ok file.

You can reinstall your g2.ok file interactively or programmatically.

To reinstall the g2.ok file interactively:

1 Select Main Menu > Miscellany > Reinstall Authorized Users.

A dialog appears describing the action about to occur and naming the file to be loaded.

2 Click OK to start the reloading process.

Note The Reinstall Authorized Users menu option only appears on the Miscellany Menu of secure G2 sites.

To reinstall the g2-ok file programmatically:

➔ Execute the g2-reinstall-authorized-users system procedure.

This procedure takes no arguments and does not return a value. It directs G2 to reread the OK file for the G2 process and install the authorized users.

When G2 is reinstalling the g2.ok file, messages are sent to the logbook and console indicating that the process has started. Other messages are sent when the process is complete.

If G2 encounters a syntactic error in the g2.ok file while it is attempting reinstallation, processing stops and none of the changes take effect. An error report is sent both to the logbook and the console.

How Reinstalling a g2.ok File Affects Current Users

When you reinstall the g2.ok file, G2 validates the contents of the file you are reinstalling against information gathered during the launch process. Specifically,

G2 checks the user-name and user-mode entries of both files, noting any changes to current users and information about new users.

Reinstalling an `g2.ok` file from which you have removed user names or modes can affect the values of the g2-window items corresponding to current users, and has the following effect upon users:

If you remove the...	G2 changes to false...	And the user...
User name of a currently logged in user	The g2-window g2-window-user-is-valid attribute	Can remain logged in as a current user, but will be unauthorized to login again after logging out from the current session.
User mode that a user is currently using	The g2-window g2-window-mode-is-valid attribute	Is not affected unless or until he or she changes modes, at which point the originally authorized mode becomes inaccessible.

Monitoring g2.ok File User Changes

While G2 takes no immediate action as a result of changes either to the g2-window-user-is-valid or g2-window-mode-is-valid attributes, developers can use whenever rules to monitor updates. By reasoning about value changes to these attributes, a secure G2 KB could take any appropriate action to deal with invalid users or user modes.

For example, this rule monitors changes to the g2-window-mode-is-valid attribute and invokes a procedure to check user modes upon any change of value in a particular window:

```
whenever the g2-window-mode-is-valid of any g2-window W
    receives a value then start check-current-mode(W)
```

These two g2-window item attribute values are updated as necessary each time the `g2.ok` file is reinstalled. Thus, if a user name or mode is removed erroneously from the `g2.ok` file, the administrator need only restore the information to the `g2.ok` file and reinstall it once more.

Additional System Procedures for Verifying User and Security Information

System administrators can also use the following system procedures to verify user modes, licenses, and login attempts:

- g2-floating-client
- g2-get-floating-licenses-remaining

- g2-get-modes-for-authorized-user
- g2-get-window-license-type
- g2-register-login-handler
- g2-set-maximum-login-attempts
- g2-validate-user-and-password

For the syntax, see User and Security Information Operations in the *G2 System Procedures Reference Manual*.

Changing User Passwords Interactively

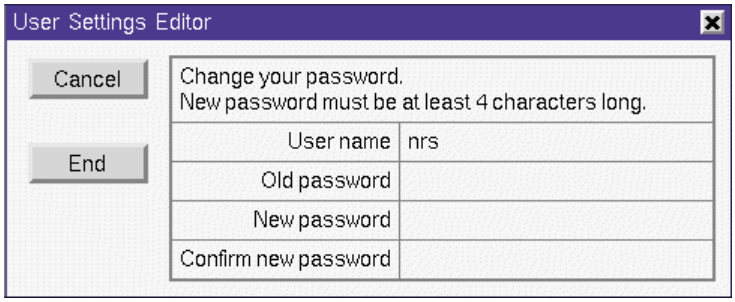
Users at secure G2 sites can change their password from within G2, provided that the administrator configures relevant user modes to have access to this option.

When a user changes his or her password from within G2, G2 spawns the `g2passwd` program with the new password information. If an error occurs while spawning the `g2passwd` process, G2 signals an error. If a problem occurs while the spawned process is executing, a message appears on the command console, but G2 does not signal an error. If G2 cannot find the `g2passwd` program, it writes an error in the log file.

To change a password interactively:

1 Select Main Menu > Miscellany > Change Password.

G2 displays the Change Password dialog with your current user name, which you cannot change:



Note The Change Password menu option only appears on the Miscellany menu of secure G2 sites. Also, the `g2passwd` program must be in the `g2` directory for the Change Password menu option to appear.

2 Enter your old password in the Old password field.

3 Enter your new password in the New password field.

4 Enter your new password a second time in the Confirm new password field.

5 Click End to save your new password in the site OK file.

If the first and second passwords to not match, the dialog prompts you to enter your password again.

Localizing the G2 Password Change Dialog

You can localize each element of the G2 Password change dialog, including:

- Button labels
- Attributes
- Simple messages

The following three sections briefly outline the steps for localizing the password facilities. Refer to Localizing Menu Choices and G2 Facilities for more information on localization.

Localizing the Dialog Buttons

You can localize these buttons:

- Cancel
- End

Note The buttons on the G2 Password change dialog do not include the Text Editor buttons Paste, Undo, and Update that appear when a user edits one of the dialog attributes.

To specify a G2 Password change dialog button:

1 Create a language translation definition by selecting KB Workspace > New Definition > language translation.

G2 invokes the text editor for you to enter the translation.

2 Enter your translation using this grammar:

in *my-lang*, as a button-label in the password-change-dialog:
button = *local-name*

For example:

```
in local-language, as a button-label in the password-change-dialog:
end = done
```

Specifying or Localizing Dialog Messages

You can change or localize the message that appears as a directive at the top of the G2 Password change dialog. The grammar identifies this message as a simple-message. By default in G2, this message reads:

Change your password.

To specify a simple message in the dialog:

1. Create a language translation definition by selecting KB Workspace > New Definition > language translation.

   G2 invokes the text editor for you to enter the translation.

2. Enter your translation using this grammar:

   in *my-lang*, as a simple-message in the password-change-dialog :
   g2-password-change-message = "*local-text*"

The symbol that denotes this particular message on the Password change dialog is g2-password-change-prompt, so to modify it, enter a definition such as:

In local-language, as a simple-message in the password-change-dialog :
g2-password-change-prompt = "Use this dialog to change your password."

Note You can enter messages such as this as a text value using quotation marks (") or as a symbol using hyphens to separate words.

Simple Messages for the Password Change Facility

The following simple messages are part of the password change facility, and can all be changed using the technique shown above:

- g2-password-change-prompt
- password-required
- new-password-required
- duplicate-new-password-required
- new-password-must-be-at-least-4-characters-long
- new-password-too-long
- new-password-has-invalid-characters
- new-passwords-do-not-match
- cannot-find-g2-ok-file
- problem-saving-password-in-g2-ok-file

Localizing Dialog Attributes

You can localize all attributes of the G2 Password change dialog, which is an instance of a password-change item. The attributes of a password-change item are:

- user-name
- old-password
- new-password
- confirm-new-password

To localize the G2 Password change dialog attributes:

1 Create a language translation definition by selecting KB Workspace > New Definition > language translation.

G2 invokes the text editor for you to enter the translation.

2 Enter your translation using this grammar:

in *my-lang*, as an attribute of a password-change :
{*g2-password-change-attribute*} = "*local-text*"

For example:

in local-language, as an attribute of a password-change :
User name = "Name:"

Telewindows Licensing Structure

Gensym offers two separate license options for its client/server-based Telewindows product—**floating** and **dedicated**. Dedicated Telewindows licenses are available in two options: development and deployment.

This Telewindows license...	Connects to...
Floating	Any G2 license option authorized for one or more floating Telewindows connections.
Dedicated development	Any G2 license option.
Dedicated deployment	Restricted G2 license option.

The two versions of dedicated licenses allow the Telewindow clients to correspond either directly with the G2 server's license option, or with a less powerful option offering fewer capabilities. For example, a dedicated deployment Telewindows client cannot connect to a G2 running a development license option, because the G2 license is more powerful than the Telewindows license.

Floating Telewindows licenses receive their level of authorization from the G2 server to which they connect. There is no distinction between development and deployment for such a license.

The Telewindows license options can connect to G2 running with a particular authorization level as follows:

	Can connect to a G2 with this authorization...			
This client...	**Development**	**Restricted**	**Development (Floating)**	**Restricted (Floating)**
Floating			✓	✓
Dedicated development	✓	✓	✓	✓
Dedicated deployment		✓	✓	✓

An exception to these corresponding license types occurs when the G2 server *also* has authorization for one or more floating Telewindows connections. In this case, if a Telewindows client with a dedicated deployment license attempts a connection to a G2 server with development authorization, G2 detects that the client has a less powerful license than it requires. Instead of rejecting the connection, however, G2 tries to use one of its floating Telewindows licenses for the dedicated deployment client.

If a license is available, G2 permits the Telewindows system to connect as a floating Telewindows client. Such a connection effectively provides a less-powerful, dedicated Telewindows client with connection to a more powerful authorization level running on the G2 server.

For more information about using Telewindows, see Accepting a Connection from a Telewindows Process.

Floating Telewindows

Floating Telewindows provides a flexible licensing scheme in which a G2 server maintains and provides authorization for a set number of Telewindows connections.

Within a floating Telewindows environment, you purchase G2 with licensing for a specific number of concurrent Telewindows connections. To increase the number of Telewindows connections, G2 requires new authorization codes. The client Telewindows software accompanying G2 can be installed on an indefinite number of client systems.

Authorization for floating Telewindows occurs in the `maximum-number-of-concurrent-floating-telewindows-allowed` attribute of the `g2.ok` file of the server G2. The limit on floating Telewindows licenses 2047.

To require Telewindows clients to provide a user name and password when connecting to G2, which restricts access to a particular user mode, configure G2 to be secure, as described in Authorizing Users at a Secure Site.

Whenever a Telewindows client attempts to log in to a G2 server, G2 first determines the client's authorization level. When the Telewindows client is a floating license, and if the new client will not exceed the total number of authorized connections, G2 permits login. If a new Telewindows client will exceed the allowable number of connections, G2 rejects the Telewindows login attempt and informs the user that no more licenses are available.

Note A dedicated Telewindows client connecting to a G2 with authorization for a finite number of floating Telewindows clients does not decrease the number of available licenses. Dedicated Telewindows clients require a different authorization process, as the next section describes.

Dedicated Telewindows

In a dedicated Telewindows environment, you purchase a separate license for each system on which you install the Telewindows software. Each license is authorized for use on a specific client system by a specific user. The concept of dedicated Telewindows licensing is the model in use for all Telewindows licenses prior to G2 Version 5.0.

Each client requires its own `tw.ok` file, authorizing that system to use Telewindows with its particular authorization level.

Once authorized, the client is capable of connecting to any available G2 server operating with the same or less powerful authorization level. Unlike floating Telewindows clients, the G2 server has neither knowledge of nor authorization capabilities for named Telewindows clients.

Simulating License Types

You can simulate the licensing options available for G2 through the KB Configuration system table, simulate-optional-modules attribute. For information about this attribute, see Simulating Optional Modules.

Simulating optional modules lets you test functionality within a KB for any less powerful authorization. For instance, if you are developing a KB for a manufacturing operator who will run the KB with a Run-time G2 license option, you can simulate that mode of operation.

Simulating optional modules cannot give you more functionality than your current license provides. For example, if you have purchased a Run-time G2 license option, you cannot simulate a developer's license option.

Part IX

Networking and Interfacing

Chapter 67: Interfacing with TCP/IP Sockets

Describes the system-defined items that allow communication with TCP/IP sockets.

Chapter 68: Foreign Functions

Describes how to call C or C++ foreign functions from within G2.

Chapter 69: Windows Services

Describes how to run G2 and G2 bridges as a service under Windows.

59

Network Security

Describes how to limit network access to a KB.

Introduction

G2 provides security for network access to G2. This chapter discusses these topics:

- Level of network security.
- Defining network security for a KB.

Determining the Level of Network Security

Network security permits you to prevent network access to a KB. Network access refers to:

- Another G2 process
- G2 Gateway
- Telewindows

G2 supports the TCP/IP protocol only.

Network security lets you secure any KB from network access at any level you choose. At the most restrictive level, you can secure an entire KB, preventing any

network access, or at a more lenient level, you can restrict one or more classes of items to have limited types of network access.

Defining Network Security for a KB

G2 provides network access security by using configuration statements such as this:

set up network access as follows: *configuration-statements*

where *configuration-statements* define the level of access you permit.

As with all configuration statements, network access configurations can affect items within the workspace hierarchy that use item-configuration, or items in a particular class hierarchy that use instance-configuration.

Using Configuration Statements for Network Access

By using the set up network access as follows: configuration statements, you can allow or prohibit different kinds of network access, using several clauses as follows. For a complete description of using configurations, see Configurations.

This configuration clause...	Allows or prohibits...
connect	Other G2 processes, G2 Gateway, or Telewindows from connecting. This configuration can be set only in the KB Configuration system table.
read	Another G2 process from reading variable values. You cannot allow or prohibit read access to or from G2 Gateway and Telewindows.
write	Another G2 process from writing variable values. You cannot allow or prohibit write access to or from G2 Gateway and Telewindows.
execute	Execute access to any item from another G2 process or from G2 Gateway.
inform	Messages being sent to the operator (the message board) from another G2 process.

Allowing or Prohibiting Connect Access

The allow/prohibit connect clauses are different from other configurations in these ways:

- You can set them only in the KB Configuration system table.
- They refer to the entire KB and are all inclusive.

For example, if you prohibit connect access to an entire KB by adding the configuration statement to the KB Configuration system table, you cannot override that prohibitive state with a subsequent configuration statement.

When you require network access with security, you can configure the KB to allow network access at a broad level, and prohibit access at specific levels. For example, to implement network access with security, you could allow connect access but restrict reading, writing, and executing objects that need to be hidden. In this way, you can provide a restricted KB *view* to an external connecting process, while still allowing network access.

Prohibiting access can be absolute, by including a prohibit absolutely clause in the configuration statement, which indicates that no other configuration clauses anywhere in the class hierarchy can override the configuration.

1924

60

Secure Communication and Authentication (SSL)

Describes how to encrypt communication and connect to TCP/IP sockets securely.

Introduction

G2 and Telewindows provide command-line options for encrypting communication on TCP/IP connections, using SSL on both Windows and UNIX platforms.

G2 supports secure communication between G2 and G2 Gateway, G2 and G2 ActiveXLink, and G2-to-G2 connections.

When connecting to TCP/IP sockets programmatically, you can also specify a secure connection.

Encrypting Communication between G2 and Telewindows

To encrypt communication on TCP/IP connections, you use the following command-line options:

`-secure`

Use SSL on all TCP/ICP connections.

`-cert` *name* | *file*

Specifies the name of the SSL server certificate to use.

On Windows, you specify the Common Name (CN) of the certificate in the local machine's `my` certificate store.

On UNIX, you specify a file containing a private key and a certificate in PEM format, which consists of the DER format base64 encoded with additional header and footer lines.

You can also use the `G2_CERT` environment variable to provide the default certificate name.

For example, the following command creates a self-signed certificate, suitable for testing, named `test`. `Makecert` is included in the Platform SDK.

```
makecert -r -pe -n "CN=test" -e 01/01/2036 -len 2048
-eku 1.3.6.1.5.5.7.3.1 \

-ss my -sr localMachine -sky exchange \

-sp "Microsoft RSA SChannel Cryptographic Provider" -sy 12
```

When the connection is encrypted, the padlock icon appears in the status bar.

Encrypting Communication between G2 and G2 Gateway

G2 Gateway supports the `-secure` and `-cert` command-line options, which are available for G2.

To access SSL, you need to include the following new libraries, depending on your platform:

UNIX	Windows
`libgsec.a`	`libgsec.lib`

If you do not want to use SSL, you need to include the following new libraries instead:

UNIX	Windows
`libnogsec.a`	`libnogsec.lib`

Failure to include one of these libraries or attempts to include both results in link errors.

In addition, you must also provide the following platform-specific libraries:

- Windows: `crypt32.lib`, available with your Microsoft compiler.
- Solaris, Linux, HP-UX, IBM AIX: `libssl.a` and `libcrypto.a`, which are supplied with G2 Gateway. Note that you must supply these two libraries in exactly this order; failure to do so will result in link errors.
- HP-UX: You must also include `libgcc.a`, also provided with G2 Gateway.

On the Windows platforms, the default `gsi.dll` is linked without SSL support; a separate library `gsi_ssl.dll` is provided to include SSL support as a DLL.

Currently, G2 Gateway does not support SSL on the alphaosf platform, but `libnogsec.a` must be linked in anyway. The example is not present.

The example `makefile` for G2 Gateway compiles most of the examples without SSL support. The `skeleton_ssl` example includes SSL support.

Attempting to give the `-secure` option to a G2 Gateway bridge that has not been linked with SSL support results in a warning message; however, the bridge will start up normally, but without SSL support.

Upon startup, a bridge gives the port number with /SSL appended when `-secure` is requested and available. For example:

```
GSI Version 2011 Rev. 0 IBM POWERstation (JA28)
2007-01-30 15:00:05   Waiting to accept a connection on:
2007-01-30 15:00:05   TCP_IP:cs-aix4:22000/SSL
```

To establish a secure connection and test the secure status, use these procedures, described in the *G2 Gateway User's Guide*:

```
gsi_int gsi_current_context_is_secure()

gsi_int gsi_establish_secure_listener
   (network, port, exact, certificate)

gsi_int gsi_initiate_secure_connection
   (interface_name, class_name, keep_connection, network, host,port,
   rpis)

gsi_int gsi_initiate_secure_connection_with_user_data
   (interface_name, class_name, keep_connection, network, host,
   port, rpis, context_user_data)
```

Note that if G2 is not listening for secure connections, this connection fails and G2 Gateway becomes inoperative. We recommend that you determine whether G2 is listening securely before executing either of these procedures.

To establish a GSI connection with security, use the secure yes option in the gsi-connection-configuration attribute, after the host and port number. For example:

```
tcp-ip host "localhost" port-number 22044 secure yes
```

For G2-G2 connections, use the icp-connection-specification attribute.

Specifying the secure yes option attempts to make a secure connection to the port number on the specified host. Note that if the host is not listening for secure

connections on the specified port, this connection fails and G2 becomes inoperative. If no host is listening at the port, then the connection simply fails.

In addition, the gsi-interface class defines the gsi-interface-is-secure attribute, and the g2-to-g2-interface class defines the interface-is-secure attribute, whose value is yes or no, which determines whether or not security was established on the connection from the remote system.

Note that you cannot make a secure G2-to-G2 connection to the same G2. This condition is detected, and an insecure connection is created instead, with a warning on the logbook.

Connecting to Sockets with SSL Security

When connecting to TCP/IP sockets using the g2-tcp-connect system procedure, you can use SSL security by specifying the secure attribute in the *options* structure as true. For example, this code fragment makes a secure connection to my-host on port 1111:

```
socket: class g2-socket;
socket = g2-tcp-connect ("my-host", 1111, structure(secure: true))
```

When listening to TCP/IP sockets using the g2-tcp-listen system procedure, you can use SSL security and specify the certificate to use by specifying these attributes in the *options* structure:

- secure — Whether to accept SSL security for clients that connect to this port. This option does not require the client to use SSL; it also accepts insecure connections. The new connection is reported as connected if it is insecure, and connected-secure if it is secure. The default is false.
- certificate — A string that identifies the SSL certificate to be used if the secure option is set to true. If the `-cert` G2 command line option has been given, it overrides the certificate option in the structure. Also, if another certificate was used to establish security, either for general G2/Telewindows communication or in another g2-tcp-listen call, that certificate is used instead. Thus, only one certificate may be active in a G2 session at one time, and once established, it is used for the remainder of the session.

For example, this code fragment creates a secure listener connection to my-host on port 1111:

```
socket: class g2-socket;
socket = g2-tcp-listen ("my-host", 1111, structure(secure: true,
    certificate: “CN=test”))
```

For details on these system procedures, see Network Connection Management in Network Operations in the *G2 System Procedures Reference Manual*.

1930

61

Telewindows Support

Describes G2's features that support Telewindows connections.

Introduction

Gensym's Telewindows product allows more than one user to access the same G2 independently. Each Telewindows user can open a **telewindow**, or remote view, into a running G2 process.

A Telewindows process can run on the same computer as the G2 process, or on a different computer to which the Telewindows connects across a network. You can also run Telewindows as a plugin in a Netscape browser or as an ActiveX control inside any COM-compliant container, such as Internet Explorer or Visual Basic.

Each Telewindows user has all the capabilities of a G2 user; in fact, to a user, a Telewindow can behave just like G2's own window seen by a G2 user. With sufficient access privileges, and depending on the configurations declared in the current KB's items, a Telewindows user can not only access the KB workspaces shown in a Telewindow, but also create rules, edit attributes, even save knowledge bases (KBs), using the remote G2 process.

Tip Associating User Modes with G2-Window Items describes how configurations determine how the current KB's knowledge appears in a G2 window.

A G2 interacts with a connected Telewindows by means of a g2-window item in the current KB. When the Telewindows process successfully connects with the G2 process, G2 automatically associates a g2-window item with that Telewindows process's own window. The G2 then updates what appears in the Telewindow through its associated g2-window item.

This chapter gives an overview of interacting with Telewindows from the G2 side. For information on using Telewindows itself, see the *Telewindows User's Guide*.

Accepting a Connection from a Telewindows Process

To start a Telewindows process, the Telewindows user must indicate which G2 to connect with by specifying:

- The machine name on which the G2 is running.
- The network address across which to make the connection.

For example, to connect to a G2 running on the machine `athens` and connected to TCP/IP network port 1122, the Telewindows user enters:

```
tw athens 1122
```

Tip When G2 starts, it displays its own TCP/IP network port number in the title bar of the window and in the title block.

You can also specify a number of command-line options for starting Telewindows, which are similar to those you use to start G2. For details, see the *Telewindows User's Guide*.

The maximum number of TCP/IP network connections to or from a single G2 process on Windows platforms is 2048.

Displaying the Telewindow

After G2 has accepted a connection from a Telewindows, the Telewindows displays its own window on the Telewindows user's machine. This window represents a view into the connected G2's current KB.

If specified in the command that launches Telewindows, the `-height` and `-width` command-line options determine the height and width of the Telewindows own window. If the size of this window is less than the size of the screen, the window

manager on the Telewindows user's machine determines the user-interface controls that enclose the Telewindow.

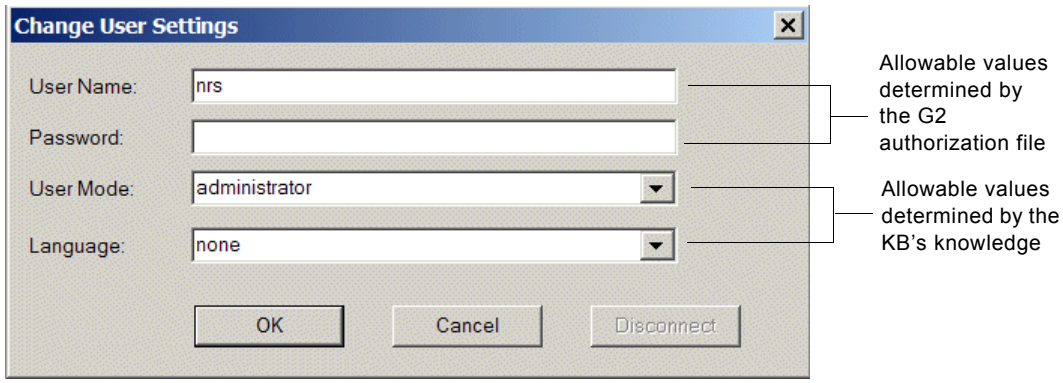
When Telewindows fails to connect to G2 on a particular host and port, a dialog appears indicating the failure.

Connecting with a G2 that is Not Secure

If the G2 and Telewindows product licenses are compatible, and if the G2 is not secure, the G2 allows a connection with the Telewindows. After G2 has accepted a connection from a Telewindows, the Telewindows displays its own window on the Telewindows user's machine. This window represents a view into the connected G2's current KB. The window manager on the Telewindows user's machine determines the appearance of this Telewindow.

Connecting with a Secure G2

If the product licenses of the G2 and Telewindows processes are compatible, and if the G2 is secure, the G2 first displays a login dialog such as:



Tip The login dialog shown above presents custom values in its fields. The current KB's own knowledge can determine the default values that display in the fields in the login dialog. For more information about how to initialize the values in the login dialog's fields, see Using the Login Dialog.

By displaying the login dialog, this G2 indicates that it requires the Telewindows user to log in before allowing a connection. This is called a **secure G2**. Whether a G2 is secure depends upon information supplied in the G2 authorization file, which is typically named `g2.ok`.

To login, the user must type a user name and password, and can optionally type a G2 user mode, G2 window name or class, and G2 window specific language.

After six unsuccessful attempts to login to a secure KB, a G2 refuses to accept any further attempts to connect by a Telewindows process.

Remember these facts about the fields in the login dialog:

- The authorization file for this G2 determines the acceptable values for the user name, password, and G2 user mode fields.
- The knowledge found in the G2's current KB determines whether the user must enter values in the G2 user mode, G2 window name or class, and G2 window specific language fields, and what are the allowable values for these fields.

For information on configuring a secure G2, see Authorizing Users at a Secure Site.

Logging Login Activities

You can register a login handler that is to be run whenever a user attempts to log into your secure G2. The login handler is a G2 procedure you write that specifies the actions you want performed when successful or failed login events occur. Changing a user's password is not considered a login event and will not call your login handler.

You must register this procedure with G2, using the system procedure g2-register-login-handler. Its single argument is the procedure, and its return value is a truth-value indicating whether the procedure has been successfully registered. See the *G2 System Procedures Reference Manual* for a description of this system procedure.

The login handler must accept a structure as an argument. The structure is returned by the system login function and has the following attributes:

Attribute	Value
success	true if the login succeeded, false otherwise.
system	The symbol tw for Telewindows.
status	A symbol describing the event.
user-name	A symbol.
user-mode	A symbol.
network-info	The `icp-connection-name` string for connections over the network and false otherwise.

The `icp-connection-name` string provides information about the protocol of the connection and the host name of the machine attempting to connect. Your login handler can use this information in any way desired.

The following example shows a login-handler that simply prints the information in the structure to the Message Board:

```
default-login-handler(info-structure: structure)
msg: text;
begin
    if (the success of info-structure)
        then msg = "succeeded"
        else msg = "didn't happen [the status of info-structure]";
    post "Login [msg] in system [the system of info-structure]
        for user [the user-name of info-structure]
        in mode: [the user-mode of info-structure]
        from [the network-info of info-structure]"
end
```

Accepting a Password

We recommend that G2 application administrators direct users to establish acceptable passwords. Note that, depending on the operating system and international character set that is in use, using the `-password` command-line option to specify passwords might constrain which specific passwords are possible for the application's users.

Associating the Telewindow with a G2-Window Item

For each connection with a Telewindows process that a G2 accepts, it also associates that connection with an item in the current KB of the class g2-window or a user-defined subclass of g2-window, as follows:

1 If the Telewindows user enters a class name in the G2 window name or class field:

- If an item of that class (but not of a subclass) exists and is not associated with a Telewindow, G2 assigns that item to the new Telewindow. If the current KB contains more than one such item, G2 picks one of the items at random.
- If the specified class exists and no item of that class exists or is available, G2 automatically creates an item of that class and associates it with the new Telewindow. G2 does not place this new item on any workspace; however, your KB's processing can do so.
- If the specified class does not exist, the Telewindows displays an error message and allows the user to enter another G2 window name or class.

2 If the Telewindows user enters the name of an item of the g2-window class, or any user-defined subclass of g2-window, in the G2 window name or class field:

- If an item of that name and of the g2-window class exists and is not associated with a Telewindow, G2 associates that item with the new Telewindow.
- If an item of that name and of the g2-window class exists but is associated with a Telewindow, the Telewindows process displays an error message and allows the user to enter another G2 window name or class.
- If no item with that name and of the g2-window class exists, the Telewindows process displays an error message and allows the user to enter another G2 window name or class.

Tip See The G2-Window Class for more information about the attributes of g2-window items. See Associating an Existing G2-Window with a Telewindow about whether your G2 application should utilize user-defined subclasses of the g2-window class.

Establishing a Window Style for Your Telewindows Process

You can override the default window-style value defined for the G2 process. G2 makes it easy for you to find the g2-window item associated with your connection with G2.

To access the g2-window associated with your Telewindows process:

➔ Select Main Menu > System Tables > This Window.

To specify a default window style for your interaction with G2:

➔ Edit the g2-window-style attribute on your g2-window item to one of these four values:

default, standard-large, g2-5.x, or standard

The g2-window-style of the Server Parameters system table determines your window-style when you specify default. You can also configure the default window style when you start Telewindows, using a command-line option.

For information on G2 window styles, see Specifying Window Styles of The Developer's Environment.

Logging Out from a Secure G2

When a Telewindows user is finished using a secure G2, he or she should log out from that G2. This is true regardless of whether the user intends to close the Telewindows connection.

To log out from a secure G2:

➔ Select Main Menu > Miscellany > Logout.

This menu choice is available only under a secure G2.

Alternatively, your G2 application can provide a different user-interface object, such as an action-button or dedicated user-defined object that, in turn, invokes the Logout menu choice.

When the user logs out, the Telewindows process continues to run and remains connected to the G2, although it is no longer associated with a g2-window item (or other item of a subclass of the g2-window class) in that G2's current KB.

After the user logs out, the Telewindows process again displays the G2 Login dialog.

Closing a Telewindows Connection

To close a Telewindows connection:

➔ Select Main Menu > Miscellany > Close Telewindows Connection.

Alternatively, your G2 application can provide a different user-interface object, such as an action button or dedicated user-defined object that, in turn, invokes the Close Telewindows Connection menu choice.

Selecting Close Telewindows Connection causes G2 to log out the Telewindow (that is, to break the association between the Telewindow and its associated g2-window item or item of a subclass of g2-window). The Telewindows process then shuts down on the user's machine.

Tip To close a particular Telewindows connection programmatically, use the g2-system-command system procedure, as described in the *G2 System Procedures Reference Manual*.

Rerouting Telewindows Connections

You can organize your G2 application so that a client Telewindows on the user's computer provides a front-end interface to one or more G2s, each of which runs a KB, and whose collective knowledge is distributed (using, for instance, G2's G2-to-G2 data service features).

One G2 can switch or reroute a Telewindows connection to another G2 process. That is, the KB running on one G2 can pass the information associated with a Telewindows connection to another G2, where the same Telewindows attempts a new connection to that G2. To the user, the new connection represents a continuation of the previous application session that took place via the (now rerouted) connection to the previous G2.

To accomplish this, G2 supports opening and closing Telewindows connections in a programmatic manner:

- Two system procedures (g2-reroute-window and g2-system-command) support connecting and disconnecting a Telewindows session programmatically.
- Attributes for the g2-window class support rerouting Telewindows connections.

These features enable a G2 application to provide a seamless means of routing a client Telewindows connection among more than one G2, subject to the availability of network resources and to the number of Telewindows connections authorized for those G2s.

Tip The sample KB `twtour.kb`, shipped with your G2 product, demonstrates the basic features that a distributed G2 application should support, while also supporting reroutable Telewindows connections. See the *Telewindows User's Guide* for an overview of this KB.

A G2 process represents a Telewindows connection as a g2-window item whose g2-window-management-type attribute has the value remote. Such a g2-window item has other attributes whose values include, in order:

1. When rerouting a Telewindows connection, the first G2 process passes the attribute values in that connection's G2 window to the second G2. This information, which was used to open the original Telewindows connection to the first G2, enables a new Telewindows connection with the second G2.
2. If the attempt to open a new Telewindows connection with the second G2 is successful, the active KB on that G2 responds to the new connection as if the Telewindows user had just connected directly.

If the second G2 is, in fact, running on a particular machine name and network port, the attempt to open a new Telewindows connection to the second G2 may not be successful under these circumstances:

- The attempted new connection exceeds the number of simultaneous authorized Telewindows connections allowed for the second G2.
- The KB running in the second G2 does not accept the login information passed to it from the first G2 process.
- A conflict exists in the licensing options between the attempted connection and the second G2.

If the attempt to reroute the Telewindows connection to the second G2 is unsuccessful, the first G2 attempts to re-establish the connection that it attempted to reroute. For this attempt to succeed, the KB running on the first G2 must include an application-specific capability to accommodate this situation.

Rerouting a Telewindows Session to a Secure G2

There are two ways to start a Telewindows session to a secure G2:

- Enter user information on the command line, using the command-line options (`-user-name`, `-user-mode`, `-password`, `-language`).
- Enter user information in the login dialog after starting a Telewindows session.

Any user information entered in the login dialog is retained and passed to the receiving G2 when rerouting a Telewindows process. Password information is encrypted. Retaining and passing user information to a G2 process has the advantage that users need not enter their user information at the command level when starting the initial Telewindows process.

A Telewindows command-line option, `-discard-user-settings`, exists to override the transfer of user information during rerouting. Telewindows users who do not want to transfer their user information should start the initial Telewindows session with this command-line option:

```
% tw -discard-user-settings
```

Using this command-line option causes a rerouted Telewindows process to discard user information.

Using System Procedures

You can use the g2-system-command system procedure to close a Telewindows connection programmatically. This procedure allows your G2 application to perform several G2 system-level operations that are otherwise available only using configurations.

- To close a Telewindows connection programmatically, pass to the g2-system-command system procedure the symbol close-telewindows-connection as the first argument and the g2-window item (associated with the Telewindows connection itself) as the second argument.
- To open a new Telewindows connection that uses knowledge from another Telewindows connection, use the g2-reroute-window system procedure. g2-reroute-window opens a new Telewindows connection and the new target G2 associates that connection with a g2-window item in its own current KB, using the attribute values of the existing Telewindows connection's associated g2-window item.

Using G2 Window Attributes

The attributes in a g2-window item contain the knowledge that your application can pass between Telewindows sessions connected to different G2 processes.

Two attributes of a g2-window item support rerouting Telewindows connections:

- The g2-window-initial-window-configuration-string attribute contains a text value that the KB running in the target G2 uses to set up the user's access to that KB.

  For instance, in a G2 application designed to support access by users via reroutable Telewindows sessions, the KB running on one G2 can hand off a user's processing to another KB running on another G2. The donor KB can log the user (via Telewindows) into another G2 and pass to its current KB a g2-window-initial-window-configuration-string value that represents the state of that user's activity within the application.
- The g2-window-reroute-problem-report attribute is a read-only attribute that presents an error message returned from an unsuccessful rerouting (via the g2-reroute-window system procedure) of a Telewindows user's session to another G2.

Applications that Reroute Telewindows Connections

The g2-system-command and g2-reroute-window system procedures, and the relevant attributes of the g2-window class, provide very specific capabilities. The KBs that comprise your distributed G2 application must perform the bulk of the

work required to reroute a Telewindows connection. The following subsections outline these requirements and offer recommendations on how to proceed.

Knowledge External to the KB

A G2 application that reroutes a Telewindows connection must acquire and manage knowledge of the application's system environment, such as:

- The machine IDs of the computers on which the application's related G2s run.
- A technique for acquiring the network locations, that is, the TCP/IP port numbers of the application's related G2s.
- Knowledge, in the form of rules, assumptions, or otherwise, about the licensing of the Telewindows that you want to connect to the application.

Application Requirements

Overall context: The application must present the application user with a context (or application metaphor) within which the rerouting of the Telewindows sessions takes place.

For example, the application can represent each G2 to which the application user can connect as a room in a virtual building that represents the scope of the application. This helps the user to understand intuitively that, in an actual building, he or she cannot occupy more than one room at a time.

Available candidates for rerouting: The application must support presenting to the application user a list of G2 processes that are suitable targets for rerouting.

Failed attempts at rerouting: The application must manage a failed attempt at rerouting a Telewindows connection, as in the case where no more available Telewindows connections were authorized for a particular G2 on a particular machine.

Depending on the licensing arrangements for G2 and Telewindows that the application users' organization has purchased, you might authorize one machine that runs G2 to allow connections with a maximum number of Telewindows users, while authorizing other machines that support your application to allow fewer concurrent Telewindows connections.

Continuity: The application can provide a continuing context (such as a history list of connections) for the application user, as the application opens and closes Telewindows connections on one machine after another.

For instance, the application can use the g2-window-initial-window-configuration-string attribute to accumulate information that the distributed KBs can use to reroute the application user's Telewindows back to some starting point.

1942

62

G2-to-G2 Interface

Describes how to connect two G2 processes and pass data between them.

Introduction

The G2-to-G2 interface lets two or more G2 processes connect for the purpose of exchanging data. G2 supports the TCP/IP protocol. Once two systems are connected, you can exchange various types of data. This chapter describes how to connect two (or more) G2 systems, and how to prepare a KB for remote data service or value and item passing across that connection.

Two sample KBs that demonstrate item and object passing are:

- `itempass.kb`
- `objpass.kb`

Both are available in the `samples` subdirectory of the `kbs` directory.

Using the G2-to-G2 Interface to Exchange Data

The G2-to-G2 interface is the general facility that permits one G2 to communicate with another. Each of the G2s can send and receive data. This lets you create applications that use distributed information processing on distinct G2s.

To enable communications between two or more G2s, you create a g2-to-g2-data-interface item on the client G2, which is the G2 making a connection to a remote server G2. This object specifies the protocol to use, along with other useful information about the remote system. This chapter calls such an object a **data interface object**.

A data interface object acts as a doorway between two G2s. Through the interface object, the KB receives data from a remote G2 and sends data to that remote process. The client G2 requires only one data interface object for the connection, though multiple interface objects can be used in a single KB. The G2 server manages the connection with the client G2 internally and does not create or require a corresponding data interface object.

An active data interface object provides:

- A point-to-point connection, allowing one process to talk to a single other process. A G2 data interface is not designed to broadcast to multiple processes.
- Data service to variables, remote expression evaluation, and remote procedure (RPC) execution.

Once you create and complete the data interface object and start both G2 systems, communication begins. The local system requests a connection to the remote system. If the connection is valid, the two systems connect and the status of the interface object is running. Once both G2 processes are started and the connection is running, the two systems can exchange data.

You should explicitly check the connection status before communicating across a network because a network connection cannot happen immediately or within a set amount of time.

Two G2s can pass knowledge between each other in these ways:

Type of Exchange	Description
Remote data service	A remote server G2 provides a value for a variable, using either another variable or a parameter, or through evaluating an expression on the remote system.
Item passing as a reference only	Passes any KB item as an integer reference, called a **network handle**.
Value or item passing	Passes any value or item, with some set of its attributes, and optionally also as a reference, to a remote G2 process. The reference passed is an integer network handle.

All data service and value or item passing requires the use of at least one data interface object for each remote G2.

You can use G2's publish/subscribe facility for event notification in distributed applications. For details, see Publish/Subscribe Facility.

Using the G2-to-G2 Interface

Using the G2-to-G2 interface requires that you configure the local knowledge base (KB) with one or more data interface objects, which are items of the g2-to-g2-data-interface class. These objects inform the local G2 of the remote G2 process (or processes) with which to communicate. You then need to configure the KB for one or more types of data service or value or item passing.

Note Though you need to configure only the client G2 with a G2 data interface, the remote server G2 process must allow the correct form of network access. Network access is described in Network Security.

Creating Data Interface Objects

To create a data interface object:

➔ Select KB Workspace > New Object > network-interface > g2-to-g2-data-interface.

Naming the Interface Object

For data service to operate properly, you must name each data interface object and that name must be unique within the KB. You use the name of the interface object to identify it within other objects and items. For example, if you use remote data service for a variable, use the name of the data interface object as the value of the variable's g2-to-g2-interface-name attribute.

Identifying Attributes

The identifying-attributes attribute is only used with GSI interfaces and is not applicable to G2-to-G2 interfaces as this time.

Setting the Warning Message Level

The interface-warning-message-level attribute sets the severity level for error and warning messages that G2 will provide for the data interface object.

Warning message levels range from 0 to 3. Level 0 is the lowest severity level, and provides the least information. Increasing the warning message level causes G2 to provide more information about errors and failures that are otherwise only detectable through the value of the interface-status attribute. Messages are posted to the Operator Logbook.

For example, when the warning message level is at 0 or 1, a failure to connect to the remote G2 process causes the interface-status to change to failed, but no information is available about why the failure occurred. If the warning message level were set to 3 and the same connection failure occurred, G2 would post a message to the Operator Logbook describing why the connection failed.

The values for this attribute are:

Attribute Value	Description
default to warning message level	Error message defaults to the system-wide message level set in the warning-message-level attribute of the Debugging Parameters system table.
0 (no warning or error messages)	Provides no information about errors and failures for the data interface object.
1 (serious error messages only)	Provides additional information about serious errors.

Attribute Value	Description
2 (all error messages)	Provides additional information on all error messages, including loss of a connection for active interfaces.
3 (all error and warning messages)	Provides additional information on all error and warning messages provided by level 2, and other messages about connection attempt failure.

During KB development and testing, it may be useful to set the value of this attribute to 3 to detect all data interface error and warning messages, and then to reset the value to 0 for KB deployment.

Defining the Connection Details

The icp-connection-specification attribute defines information the KB needs to connect to a remote system. The connection information includes:

- The protocol to use.
- The name of the remote system.
- Either a host-machine name or port number.
- Whether the connection is secure.

G2 supports the TCP/IP protocol. You can also create a data interface object that connects to the G2 process itself. Such a connection is referred to as a **local emulator**. You can use the local emulator to test a KB running on a single G2 process that eventually will be connected to one or more remote G2s.

The syntax for the connection specification is:

```
{local emulator |
  tcp-ip host "host-machine-name" port-number port-number [secure yes] }
```

Element	Description
"*host-machine-name*"	The *host-machine-name* is a case-sensitive string that you enter within quotation marks.
port-number	The unique port identifier used in TCP/IP systems.
secure yes	Attempts to make a secure connection to the port number on the specified host, using SSL (Windows) or OpenSSL (UNIX). **Note:** If the host is not listening for secure connections on the specified port, this connection fails and G2 becomes inoperative. If no host is listening at the port, then the connection simply fails. **Note:** You cannot make a secure G2-to-G2 connection to the same G2. This condition is detected, and an insecure connection is created instead, with a warning on the logbook.

To obtain the host machine name and either port number or task name:

➔ Select Main Menu > Miscellany > Network Info.

Alternatively, you can obtain this information programmatically, using the system procedures g2-get-host-name and g2-get-port-number-or-name, available in the `sys-mod.kb`.

Setting the Interface Timeout Interval

The interface-timeout-period attribute controls how much time should elapse before the local G2 assumes that the G2-to-G2 connection is inoperative and times out. You can create whenever rules in your KB to take appropriate action for a data interface object timeout.

The time specified here refers to the timeout limit for the network communications link, *not* to the update interval of any variable being used. Even if the update interval of every variable exceeds the interface timeout period, the local data interface object will not time out unless the network connection is lost.

Obtaining the Current Connection Status

The interface-status attribute indicates the current status of the G2-to-G2 connection. This is a read-only value, which changes with the state of the connection. Possible values are:

This value...	Indicates that...
inactive	The interface is either on an inactive workspace, has no name, is otherwise not OK, or the local G2 has not started yet.
attempting	The interface is trying to make a connection to a remote G2, but has not yet completed the process and is not ready to transmit or receive data. Typically, the interface is in this state only briefly before obtaining either a connected or failed status.
connected	The interface has successfully connected to a remote G2 and is ready to transmit or receive data.
failed	The interface has attempted to make a connection to a remote G2, but the attempt was unsuccessful.
paused	The remote G2 is paused.
running	The remote G2 has been started or restarted and is not paused.
reset	The remote G2 has not been started or restarted.
timed-out	The local G2 has received no data from the remote G2 within a time interval given by the interface-timeout-interval attribute of the interface object.

When you are creating a data interface object, the value of this attribute is always inactive until the interface object is activated.

Using Whenever Rules That Refer to the Connection Status

You can refer to the value of the interface-status attribute to obtain the connection status, but you cannot change this value either interactively or programmatically. For instance, you could write a rule such as the following, to test for and take action upon a particular status.

```
whenever the interface-status of world-connection receives a value
    and when the interface-status of world-connection is running
    then change the background-color of the
    subworkspace of world-connection to green
```

Starting the G2 Processes

To start the G2-to-G2 connection, start G2 on both computers. When the local G2 needs data from the remote G2 process, it obtains it in whatever way the KB is configured to pass data: data service, or value or item passing through remote procedure execution.

Activating Data Interface Objects

Locating data interface objects upon an activatable subworkspace lets you control the objects programmatically. By activating or deactivating the subworkspace upon which a data interface object resides, you can activate or deactivate the object.

You can also use a conclude action to control activation. Concluding the icp-connection-specification attribute to have no value closes the connection. Here are two states that use conclude actions to activate and deactivate the connection.

This statement closes a connection:

```
conclude that the icp-connection-specification of connection3 has no value
```

This statement activates a connection:

```
conclude that the icp-connection-specification of connection3 =
    structure(network-transport: the symbol TCP-IP,
              hostname: "GHWSYS", port: 1111)
```

The G2-to-G2-Data-Interface Class

The class-specific attributes of g2-to-g2-data-interface items are:

Attribute	Description
names	The name of the interface object.
Allowable values:	Any unique name
Default value:	none
identifying-attributes	For GSI interface only.
Allowable values:	Not applicable.
Default value:	none

Attribute	Description
interface-warning-message-level	The severity level of error and warning messages about which G2 provides information.
Allowable values:	0 - 3
Default value:	default to warning message level
ICP-connection-specification	The information required for connecting, including the remote system, the protocol, and host name.
Allowable values:	See Defining the Connection Details.
Default value:	none
interface-timeout-period	The length of time G2 waits before timing out after attempting to connect to the remote system.
Allowable values:	use default any *time-interval*
Default value:	use default (10 seconds)
interface-status	The current status of the interface object.
Allowable values:	See Obtaining the Current Connection Status.
Default value:	inactive
interface-is-secure	(Read-only) Whether the connection is secure. See Defining the Connection Details.
Allowable values:	yes | no
Default value:	N/A

Creating Data Interface Subclasses

If you need additional user-defined attributes for your data interface object or want to provide a specific icon-description, create a new subclass by using a class definition. Specify the g2-to-g2-data-interface class as the direct superior class.

Once you have created a G2 data interface subclass, you can create any number of instances of it, each of which can specify a different remote G2 process. As a result, you may want to define a standard interface class as part of a KB and merge it into other KBs whenever they require the G2-to-G2 interface facility.

If you are creating a subclass that uses multiple inheritance, the G2 data interface can be either a primary or a secondary superior class. For more information about creating subclasses, see Creating Class Definitions. For a description of primary and secondary superior classes, see Specifying the Superior Class(es).

Using Remote Data Service

One use of a G2-to-G2 connection allows remote data service to one or more variables.

To configure the KB for remote data service:

1. Create at least one data interface object, as described in Creating Data Interface Objects.
2. Create one or more **G2-to-G2-variables** as described next, which are variables that can receive values from (and, in some cases, send data to) the remote G2 process.

Creating a G2-to-G2 Variable

A g2-to-g2 variable is a variable subclass that includes the g2-to-g2-data-service class as one of its direct superior classes.

To create a g2-to-g2 variable:

1. Define a subclass of any of the system-defined variable classes, and include the mixin class g2-to-g2-data-service as one of the direct superior classes. The mixin provides the additional attributes needed for G2-to-G2 data service. Give the subclass any unique name.
2. Edit the Attribute-initializations validity-interval attribute to specify any time interval, or indefinite. Data servers other than the Inference Engine cannot have a validity interval of supplied, which is the default.
3. Optionally, customize the new class in any other way.
4. Create an instance of the new class and open its table.

Using the g2-to-g2-data-service mixin class sets the data-server attribute of the variable to G2 data server, and adds two additional attributes, g2-to-g2-interface-name and remote g2-expression.

Specifying the G2 Data Interface

The g2-to-g2-interface-name attribute specifies the data interface object through which you want G2 to obtain values for the variable.

Enter the name of the data interface object that this variable should use.

Defining the Remote G2 Expression

Edit the remote-g2-expression attribute to indicate what expression you want G2 to evaluate on the remote G2.

G2 uses this expression to compute a value for the variable. The expression can include references to any item in the remote G2, but it cannot contain references to items on the local system. For example, to get the current time from the remote machine, you could enter the current time for the remote-g2-expression attribute.

Considering Network Access Configurations

Be aware that all items can include configuration statements that can affect network access. For example, an item could be configured with this statement:

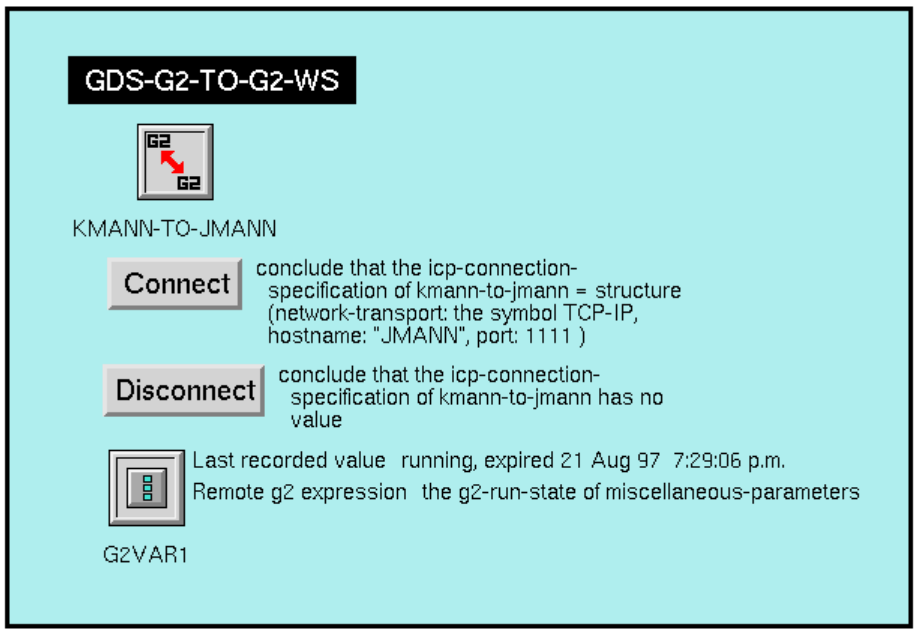
```
set up network access as follows: prohibit read access to this item by g2
```

If the expression in this attribute references a remote item configured this way, G2 will be unable to compute a value for the variable. To the local G2, an item with such a configuration statement is indistinguishable from a deactivated item.

Examples of Remote Data Service

The next figure illustrates how a symbolic variable has been defined and is displaying the current run state of the remote G2, available as an attribute of the Miscellaneous Parameters system table, using the expression:

```
the g2-run-state of miscellaneous-parameters
```



When the variable requires a value, it receives one by evaluating its expression through the kmann-to-jmann object, which is a g2-to-g2-data-interface object.

A variable in the local KB can evaluate to a variable in the remote G2, simply by referring to the remote variable by name. When a local variable refers to a remote variable in this way, the two variables are further associated. If the local G2 executes a set action on the local variable, the remote G2 concludes that the remote variable is equal to the value of the local G2 variable. In this way, the local G2 can send information to (as well as receive information from) the remote G2 process.

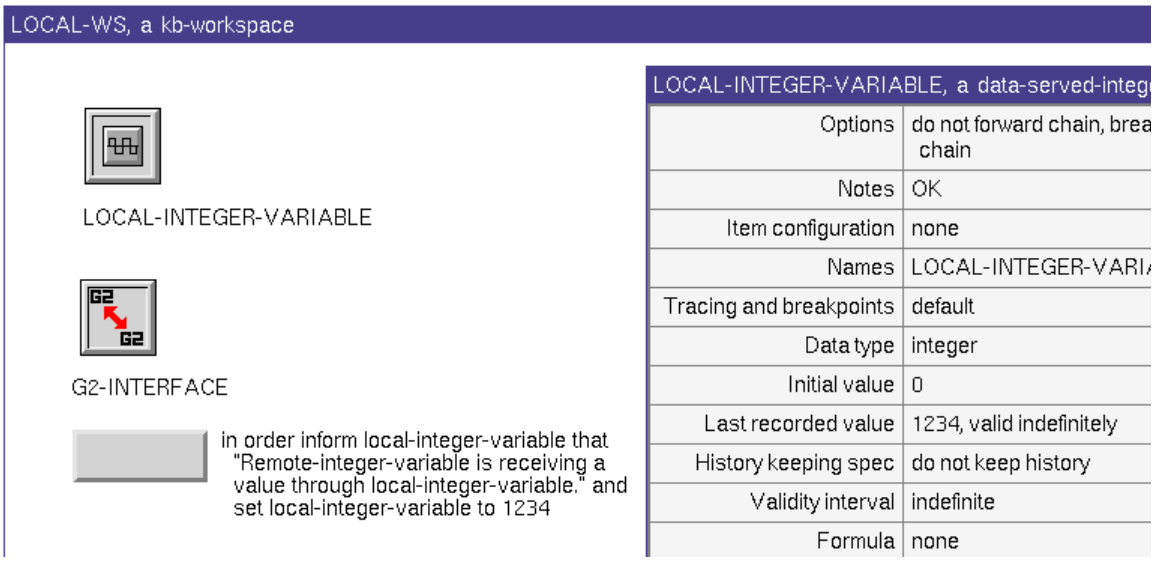
Also, multiple variables in the local KB can use a single G2-to-g2 data-interface object through which to obtain their values, or you can create multiple data interface objects to provide remote data service to different variables.

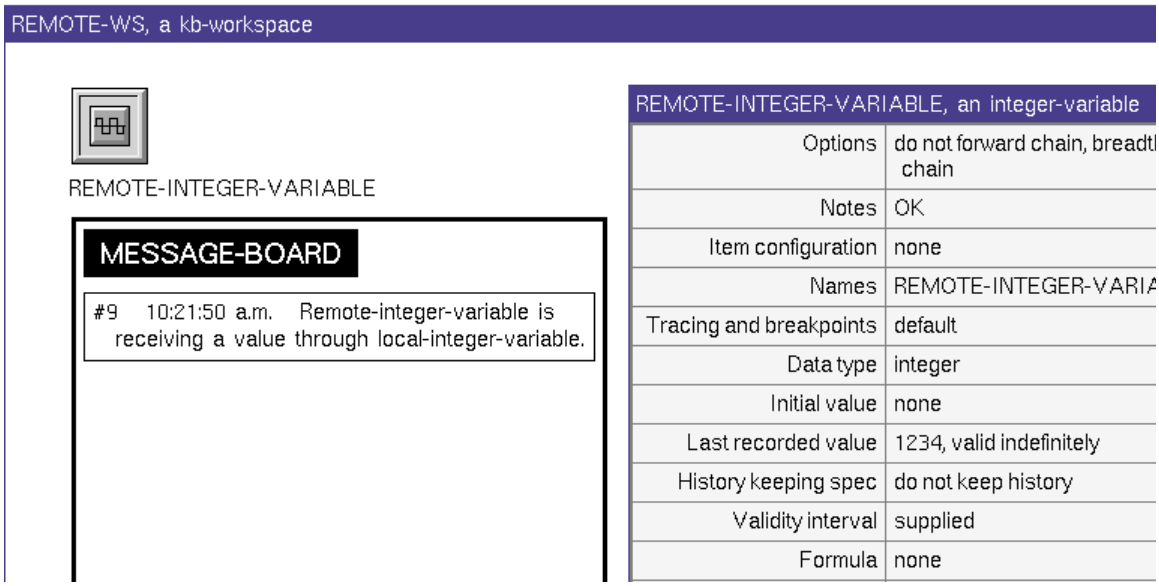
The following examples demonstrate inform and set actions evoked from a KB running on a local computer connected to a to a KB running remotely. The local KB contains local-integer-variable a g2-to-g2 data-served integer variable whose Remote G2 expression attribute has remote-integer-variable as its value. Evoking the inform and set actions through local-integer-variable results in a message appearing on the Message Board of the remote KB and the value of remote-integer-variable being set to the value in the set expression.

The local KB invokes this action code from an action button:

```
in order
    inform local-integer-variable that "Remote-integer-variable is receiving a
        value through local-integer-variable."
    and set local-integer-variable to 1234
```

The actions have these effects on the items on a local KB workspace and the items on a remote KB workspace:





Using Remote Procedure Calls

You can use remote procedure calls (RPCs) in any application that requires G2 to execute a procedure in another G2 across an ICP interface object. In particular, you can use RPCs for value- and item-passing, and to build interfaces to external devices and database systems. You can also use RPCs to allow G2 Gateway to call procedures in G2 and receive return values.

The G2 that issues the remote procedure call is the client. The G2 that executes that call is the server. To use RPCs, the client G2 declares that a particular procedure is remote by using a remote procedure declaration. The remote procedure declaration specifies the name, argument types, and return types of the procedure as it exists on the remote G2.

Depending on the arguments of the remote procedure, and thus how you declare the remote procedure on the client G2, you can send or receive:

- Variable or parameter values.
- A reference to any item.
- A copy of any item with any number of its user-defined, and user-accessible system-defined attributes.

Note For RPC calls, the order is stable when passing sequences, structures, g2-list elements, and g2-array elements; but the order is not stable for the attributes of items.

The next three sections describe how to create, declare, and call or start a remote procedure. Subsequent sections present the valid arguments for RPCs, and various ways to use RPCs to pass values and items to and from remote systems.

Creating and Declaring a Remote Procedure

To create a remote procedure declaration:

➔ Select KB Workspace > New Definition > remote-procedure-declaration.

G2 invokes the Text Editor immediately so that you can complete the remote procedure. After completing the declaration, you can place the remote procedure onto the workspace and then open its attribute table.

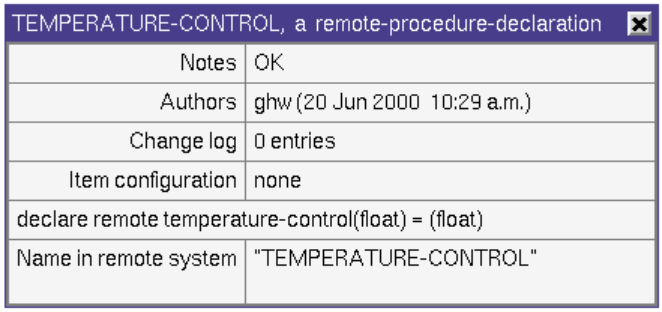
You complete the remote procedure declaration by stating the name of the procedure you will call from the local KB, but which resides on a remote G2 process. The syntax is:

declare remote *remote-procedure-name* ([*argument*] [,...]) = (*return* [,...])

Element	Description
remote-procedure-name	The name of the procedure on the remote system.
argument	One or more arguments that the procedure accepts. You can specify up to 200 arguments within the parentheses. For a complete description of RPC arguments, see Value and Item Passing Arguments and Return Types for RPCs.
return	Indicates the return value (or values) of the procedure.

Using an Alternative Procedure Name

The name you enter in the declaration statement is duplicated as a string in capital letters in the name-in-remote-system attribute of the remote procedure declaration's attribute table, as shown in this diagram.



You can use the name-in-remote-system attribute to declare a remote procedure with a different name so that you do not have two procedures with the same name in your local KB.

Once you edit this attribute, G2 decouples the name from the remote-procedure-name in the remote procedure declaration syntax. This means that further edits to the declaration will not affect the name-in-remote-system attribute. You will have to edit that attribute directly to change it.

The name-in-remote-system attribute of the remote procedure declaration is a case-sensitive string. For example, if the name of the remote procedure is my-proc, and you enter my-proc, G2 will be unable to locate the appropriate procedure on the remote system. By default, G2 procedure names are uppercase.

Invoking Remote Procedures

You can invoke a remote procedure from the client G2 in one of two ways:

- A start action with the across *g2-to-g2-interface* phrase.
- A call statement with the across *g2-to-g2-interface* phrase.

You can use the start action for remote procedures in both rules and procedures. When you use start, the client G2 continues to execute the calling rule or procedure.

You can use the call statement only in procedures. When you use call, the client G2 waits until the remote procedure completes before continuing with the calling procedure.

It is possible for a call to a remote procedure to be aborted at the client, for example, when a different branch of a do in parallel statement within a procedure finishes first. In this case, the client procedure continues processing and the remote procedure is aborted.

Starting a Remote Procedure

To use the start statement for a remote procedure, the syntax is:

start *remote procedure* (*argument* [,...])
[at priority *integer-expression*] [after *time-interval*]
across *g2-to-g2 interface*

Element	Description
remote-procedure	An expression that returns a procedure.
at priority *integer-expression*	Specifies the priority of the procedure. This priority setting overrides the default priority only within the client; within the server, the execution of the remote procedure call begins at that procedure's default priority.
after *time-interval*	Specifies a time interval after which the remote procedure should start in the client.
across *g2-to-g2 interface*	Indicates the ICP interface through which the RPC is being called. If the ICP interface object is not connected, an G2 signals an error in the client process executing the start action, but the executing rule or procedure continues processing.

If the remote procedure does not exist in the server, or if an error occurs in starting the remote procedure, G2 signals an error in the client.

If an error occurs during the execution of the remote procedure after it has started, the server G2 handles the error as if the procedure were called locally. The client G2 does not signal an error. If the remote procedure was called from a higher-level procedure, that procedure continues processing.

Calling a Remote Procedure

To use the call statement for a remote procedure, the syntax is:

[return [,...] =] call *remote procedure* (*argument* [, ...]])
  across *g2-to-g2 interface*

Element	Description
remote-procedure	An expression that returns a procedure.
across *g2-to-g2 interface*	Indicates the ICP interface through which the RPC is being called. If the ICP interface object is not connected, an error occurs in the client process and G2 signals an error on the client.

In the client G2, the processing to initiate the call occurs at the same priority as that of the procedure containing the call. This priority does not affect the priority of the call within the server G2, where the execution of the remote procedure call begins at the default priority of the remote procedure.

If an error occurs during the execution of the called remote procedure, the remote procedure is aborted in the server G2 and the error message is propagated back to the client. On the client G2, the procedure that initiated the remote call is aborted and a message is displayed on the client logbook.

Value and Item Passing Arguments and Return Types for RPCs

All item instances have system-defined attributes, such as notes, names, and item-configuration. User-defined classes usually include user attributes, defined by the user in the class definition.

Using a remote procedure declaration, you can specify which values, user-defined attributes or user-accessible system-defined attributes you want to pass to or from a remote G2.

You can pass any value to a remote G2. Since structure and sequence values can consist of items as their attributes and elements, and those items include their own attributes and values, the RPC grammar for specifying a value argument lets you specify attributes and values. For more information about value passing, see Value Passing.

G2 can pass an item in various ways, including:

- As a handle only.
- With all of its user attributes and no system attributes (the default).
- With all user attributes, except those in a specified set, and zero or more allowed system attributes.
- With only a specified set of user attributes, and zero or more allowed system attributes.
- All of the above, and with a handle.

You can specify one or more of these possibilities as part of the remote procedure declaration syntax.

The following figures illustrate the grammar that you can use in G2 remote procedure declarations:

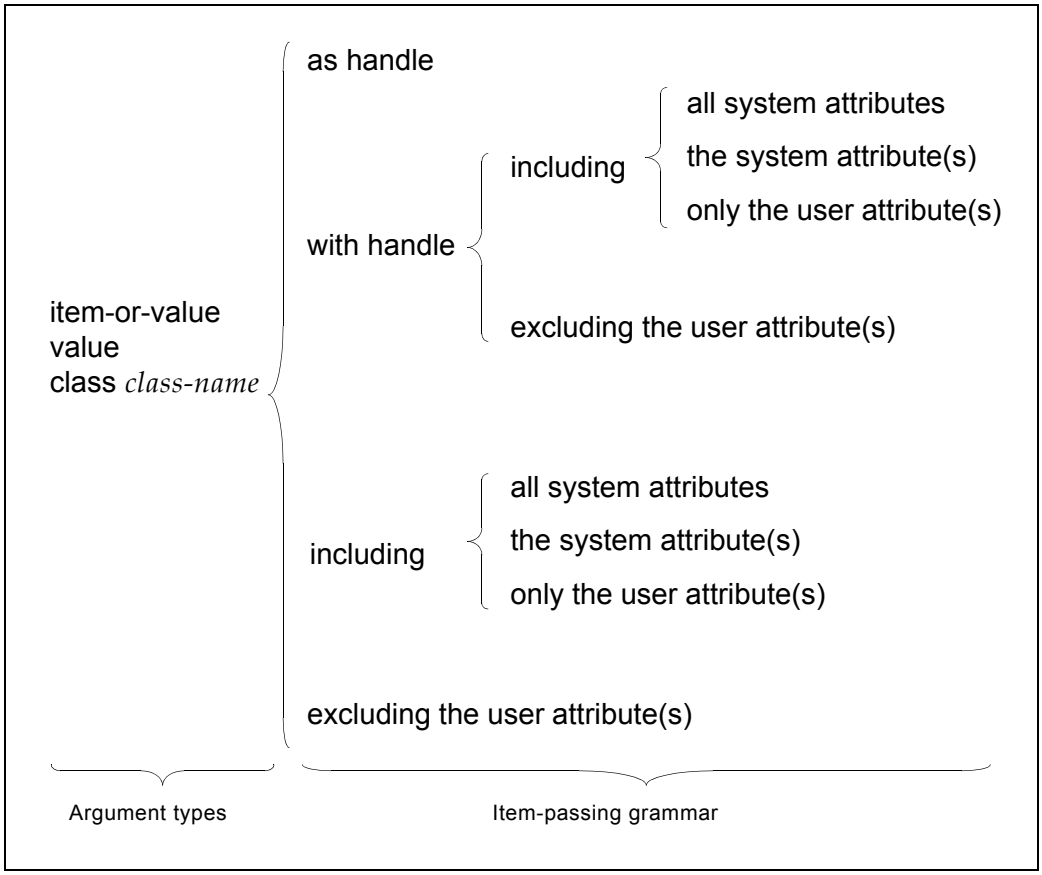
Grammar for Remote Procedure Declarations

declare remote *procedure-name*
([*argument* [,...]] [all remaining *argument*]) =
([*argument* [,...]] [all remaining *argument*])

Argument Grammar for Remote Procedure Declarations

integer
float
quantity
text
truth-value
symbol
item-passing-argument
(*item-passing-argument*)

Item-Passing Argument Grammar for RPC Declarations



Note In a list of argument- or return types, specifying class as an argument followed by anything more than a *class-name*, requires that you enclose the argument or return type in parentheses, followed by a comma if other arguments follow.

Considerations for Item Passing

Before completing the required steps for item passing, consider these things:

- Both G2 processes must have compatible definitions for user-defined classes.
- The type and name of user-defined attributes of items must align.
- Transient items may accumulate through item passing.

Creating Compatible Definitions

Compatible definitions must exist on both the client and the server G2 when you pass user-defined items between two G2s. The system defined items have compatible definitions automatically.

For user-defined classes, the degree of definitional compatibility can vary. At the very least, the two definitions must have the same class name. For example, to pass an item called PC, a definition for an item called must PC exist in both the client and the server KBs.

Compatible definitions do not have to be identical, and the attributes in one definition can be a subset of the other. For a receiving definition to contain a subset of the sending definition's attributes, you can use the statements for including or excluding particular attributes in the remote procedure declaration.

Aligning User-Defined Attributes

User-defined attributes passed from one G2 process to another must be of the same name and type in both definitions. For instance, you could not define a user-attribute in one definition like this:

```
temp is given by a quantitative-variable
```

and specify the attribute as temp is an integer, in the other definition. The attribute temp must be either an integer or given by a quantitative variable in both definitions to pass that attribute successfully between two G2 processes.

Accumulating Transient Items

Item passing creates a transient item on either the local or remote G2. As with any transient item, you can handle placing the item on a workspace, or manipulating it in any other way, programmatically.

Keep memory resources in mind when using item passing. All transient items consume memory, so you must explicitly delete them when they are no longer needed. For additional information, see Memory Management.

If your KB needs to maintain the items that are passed to it, make the transient items permanent, using the make permanent action. For more information on this action, see make.

Value Passing

The G2-to-G2 interface lets you pass values to and get values from a remote G2 process.

The argument and return types for a remote procedure declaration can be of any value type:

- value

- quantity
- integer
- float
- text
- symbol
- truth-value
- sequence
- structure

When passing an argument or return value declared as a structure or sequence, G2 passes the value in its entirety. You cannot specify that some attributes and values of a structure, or certain elements of a sequence, be passed.

Since structure and sequence values can include items, which include user- or system-defined attributes, the RPC argument and return value grammar lets you specify one or more user- or system-defined attributes and their values, just as you would when passing an item, rather than a value.

For the grammar to specify user- or system-defined attributes, see Passing User- and System-Defined Classes.

In a list of arguments, each must be separated by a comma, or parentheses and a comma when specifying a class, as described in Value and Item Passing Arguments and Return Types for RPCs.

Configuring the KB for Value Passing

You configure your KB to pass a value to a remote G2 by using a remote procedure declaration (RPC).

To configure a KB for value passing using an RPC:

1. Create a data interface object and complete it as explained in Creating Data Interface Objects.
2. Create a procedure on the remote G2 that declares one or more values as an argument and/or as a return value.
3. In the client G2, create a remote procedure declaration with the correct arguments of the procedure on the remote G2.
4. In the client G2, create a rule or procedure that starts or calls the remote procedure across the appropriate data interface object to obtain a value.

Example of Passing an Integer Value

Here is a simple example of passing an integer value.

1 A procedure in the local G2 that calls a remote procedure across the kmann-to-jmann data interface object.

```
get-remote-items()
total-items: integer;
begin
    total-items = call get-remote-item-count() across connection3;
    post "The total number of items on the remote system is [total-items]."
end
```

2 The remote procedure that returns the number of items on one of its workspaces.

```
get-remote-item-count() = (integer)
begin
    return the count of each item upon gds-g2-remote-workspace
end
```

Example of Passing a Structure Value

Whenever you pass a structure or sequence to or obtain a structure or sequence from a remote system, G2 includes all of the attributes or elements of that value. If a structure attribute consists of an item, you can optionally choose which user- or system-defined attributes of that item to pass.

Here is a basic example of passing a structure value:

1 A procedure in the local G2 that calls the remote composite-value-proc procedure and displays the results.

```
get-remote-structure()
history-spec: structure;
begin
    history-spec = call composite-value-procedure() across connection3;
    post "The history-keep-spec of a remote variable is [history-spec]."
end
```

2 The remote procedure, which obtains a structure representing the history-keeping-spec of a variable called V1, and returns that structure to the local G2.

```
composite-value-procedure() = (structure)
S: structure;
begin
    S = the history-keeping-spec of v1;
    return S
end
```

Passing an Item as a Network Handle

You can configure your KB to pass an item as a network handle only.

To pass an item as a handle:

➔ declare remote procedure (class *class-name* as handle) = (*return* [,...])

where *class-name* is any system- or user-defined G2 class.

You can pass an item as a handle as one of two or more arguments or return types by entering the item argument in parentheses, followed by a comma if any arguments follow.

To pass an item as a handle with other arguments:

➔ declare remote procedure (float, (class *class-name* as handle), integer) = (*return* [,...])

Configuring the KB for Item Passing as a Network Handle

To configure a KB for item passing as a handle:

1. Create a data interface object and complete it as explained in Creating Data Interface Objects.
2. Create a procedure on the remote system that declares an integer (for the network handle) as an argument or a return value.
3. Create a remote procedure declaration with the correct syntax for passing an item as a handle, as described in Value and Item Passing Arguments and Return Types for RPCs.
4. Start or call the remote procedure from the local G2 across the appropriate interface.

Obtaining Network Handles

A network handle is an integer used to refer to an item. If you register the item in the local G2, the remote G2 can then refer to it as a reference.

An item can acquire a network handle in three ways. The third way is applicable only to GSI interfaces.

- Automatically, by declaring that an item is a handle in the argument specification of a remote procedure declaration.

  During remote procedure invocation, if an item does not have a network handle, G2 registers it automatically, associating the item with the ICP interface object across which the procedure has been invoked. The item's handle is then passed to the remote procedure.

- Manually, by using the g2-register-on-network system procedure, described in the next section.

  The system procedure returns a network handle for any item passed to it, associating the item with the specified interface object. The interface object could be either a data interface object, or a GSI interface object, because item passing works through GSI interfaces as well.

- Automatically, through the GSI data service. This applies only to items that are a subclasses of GSI-data-service being data served through a GSI interface object.

For more information about data service and item passing in GSI, see the *G2 Gateway Bridge Developer's Guide*.

Using a System Procedure to Obtain a Network Handle

To obtain a network handle for an item manually:

➔ g2-register-on-network
(*item-to-register*: class item, *icp-interface*: class item)
-> *network-handle*: integer

Element	Description
item-to-register	The item for which you need a network handle.
icp-interface	The network interface you are using.

Example of Obtaining a Network Handle

For example, the next procedure accepts any item and any data interface object as its arguments, and returns a handle.

```
get-item-handle(register-item: class item,
                        data-interface: class g2-to-g2-data-interface) = (integer)
handle: integer;
begin
    handle = call g2-register-on-network(register-item, data-interface);
    return handle
end
```

You cannot maintain network handles across interface activations. You must register items each time you activate a data interface object.

You can typically acquire network handles automatically, using the remote procedure argument declarations. If, however, the local process requires information about network handles before they are acquired automatically during a remote procedure call, you can acquire them manually, using the system procedure.

The following system procedures work in conjunction with item passing and network handles:

To...	Use this system procedure...
Register an item on the network to obtain a network handle associated with the item and the ICP interface object.	g2-register-on-network
Deregister an item from the network handle number assigned it, and from the interface. G2 may use that network handle again to assign to a new item during network registration.	g2-deregister-on-network
Obtain the item associated with a network handle number and its interface.	g2-get-item-from-network-handle
Obtain the network handle associated with an item already registered with an interface, whether through the g2-register-on-network system procedure or through a remote procedure invocation.	g2-get-network-handle-from-item

For more information on the network registering, and other system procedures, see the *G2 System Procedures Reference Manual*.

Example of Passing an Item as a Handle

The statement below shows an example of how you declare a remote procedure with a float argument type, followed by a procedure class as a handle (in separate parentheses), with an integer return type value.

```
declare remote adding-procedure(float, class procedure as handle)) = integer
```

Passing Variables and Parameters

Within G2, in almost all cases when you refer to a variable or parameter, the expression returns the value of the variable or parameter. When passing variables and parameters across a G2-to-G2 connection, the value is not passed unless you explicitly state that it should be.

These are the ways in which you can pass variables and parameters:

- As an item, which passes a copy of the variable or parameter. As with all items being passed, by default, passing a copy includes all user-defined attributes but no system-defined attributes. Thus, the variable or parameter value is not passed automatically.
- As a handle. Passes the handle number to the remote system, not the variable or parameter value.
- As a value by using the RPC grammar:

  including the system attribute current value of variable-or-parameter

Passing a Variable or Parameter as a Copy or Handle

You can pass a copy of a variable or parameter with all of its user-defined attributes and none of its system-defined attributes by using a declaration such as:

declare remote variable-copy ((class *variable-or-parameter*)) = (*return* [,...])

You can pass a variable or parameter as only a handle. For example, use a declaration such as this to pass a variable or a parameter as a handle:

```
declare remote syria-proc((class integer-variable as handle),
                          (class integer-parameter as handle)) = (integer)
```

Passing the Current Value of a Variable or Parameter

You can pass the current value of a variable or parameter by specifying that in the RPC grammar.

To pass a variable or parameter value:

➔ declare remote *procedure* ((class *variable-or-parameter* including the system attribute current value of variable-or-parameter)) = (*return* [,...])

Here is a basic example of passing a variable value:

1 A procedure in the local G2 that calls the remote variable-value procedure across the data interface object and displays the return results.

```
get-variable-value(VAR: class integer-variable)
T: text;
begin
    T = call variable-value(VAR) across connection203;
    post "[T]"
end
```

2 The remote procedure declaration indicating that the *value* of the local variable should be passed to the remote procedure.

```
declare remote
    variable-value((class integer-variable including the system attribute
                        current value of variable-or-parameter)) = (text)
```

3 The remote procedure that:

- Transfers the variable passed from the local G2 onto the workspace of the procedure.
- Names the variable.
- Performs a collect data statement to get the current value of the variable.

c Returns a text string that includes the value, and which is what the local procedure displays:

```
variable-value(V: class integer-variable) = (text)
T: text = "The value of the passed variable is:";
current-value: class integer-variable;
begin
    transfer V to this workspace at (50, 50);
    change the name of V to the symbol RemVar;
    collect data
        current-value = V;
        T = "[T][V]"
    end;
    return T
end
```

Passing User- and System-Defined Classes

Item passing lets you copy an item from one G2 to another. The item may be a user-defined class, or any system-defined class, including definitions.

Most user-defined classes have user-defined attributes, and all classes have system-defined attributes, such as notes, item-configuration, and names. Item passing lets you determine which user- or system-defined attributes to pass from one G2 to another.

G2 supports item passing in several ways:

- As a handle only.
- As a copy with all of user-defined attributes and no system-defined attributes.
- As a copy with one or more user-defined attributes, and zero or more system-defined attributes.
- As a copy with all user-defined attributes and all user-accessible system-defined.

- As a copy with user- and system-defined attributes, and additionally with a handle.

Passing an item either as a handle or with an handle, means passing any item with an integer value network handle. Obtaining Network Handles describes how to get and use handles.

Note User-accessible system-defined attributes are those that are available through the attribute access facility, and which appear in the *G2 Class Reference Manual*.

Configuring the KB for Passing an Item with Attributes

To configure a KB for item passing using a remote procedure declaration:

1. Create a data interface object and complete it as explained in Creating Data Interface Objects.
2. For items of user-defined classes, ensure that both G2 systems have compatible definitions for the class or classes being passed as discussed in Creating Compatible Definitions.
3. In the remote G2, create a procedure that declares one or more items or values as its arguments and/or return type. The remote procedure can also include an integer argument if you wish to optionally include a handle for the item you are passing.
4. In the local G2, create a remote procedure declaration with the correct syntax for passing an item with user- and/or system-defined attributes, as described in Value and Item Passing Arguments and Return Types for RPCs.
5. In the local G2, start or call the remote procedure across the appropriate data interface object.

Passing an Entire Item or a Specific Attribute Set

By default, passing an item automatically includes all of its user-defined attributes and none of its system-defined attributes. You can include or exclude user-defined attributes explicitly. Including one or more user-defined attributes excludes the remainder. Excluding one or more user-defined attributes includes the remainder.

You can include all system attributes or those that you declare explicitly, but you cannot exclude system attributes as you can user attributes.

You can specify one or more of these possibilities as part of the remote procedure declaration syntax.

Item-Passing Examples

You use a remote procedure declaration to specify which attributes to pass to or from a remote G2.

The including or excluding of user- or system-defined attributes is specified through the grammar of a remote procedure declaration. Though a remote procedure declaration can include multiple variations of its grammar, this section describes the grammatical options separately. You can combine these options in any syntactically meaningful way. For example, see:

- Passing a Copy of any Item.
- Passing an Item Excluding User-Defined Attributes.

Within a remote procedure declaration, you can combine such statements, along with others, to include some attributes and to exclude others.

Passing a Copy of any Item

By default, an item is passed as a copy, with no handle, with all of its user-defined attributes, and none of its system attributes.

To pass an item with all user-defined attributes:

➔ declare remote *procedure-name* (class *class-name*)

You can pass an item with one or more of its user attributes, by specifying what to include. Explicitly including one or more user-defined attributes excludes the remainder.

Example of Passing Copies of Items

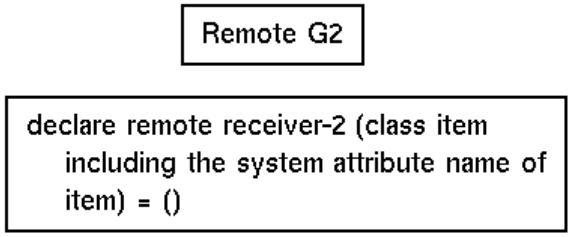
Compare the previous example with passing items by copy, using RPCs.

The following example shows that when declaring a remote procedure that takes an item as its argument, the receiver creates a copy of the original item. Here is the receiver procedure in the local G2:

The receiver-2 procedure takes an item as argument, posts the name and UUID of the item to the Message Board, and transfers the item to the workspace:

```
receiver-2(i: class item)
begin
    post "received an item named [the name of i] with UUID "[the text of the uuid of
        i]"";
    transfer i to this workspace;
end
```

The remote procedure declaration in the remote G2 takes as its argument an item and includes the name attribute only:

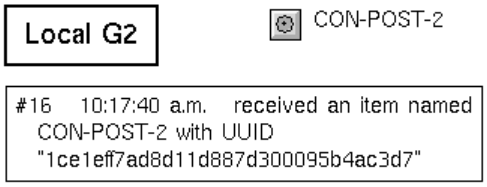


In the remote G2, the send-item procedure makes a remote procedure call to receiver-2 across the network interface, passing the item con-post-2 as the argument:

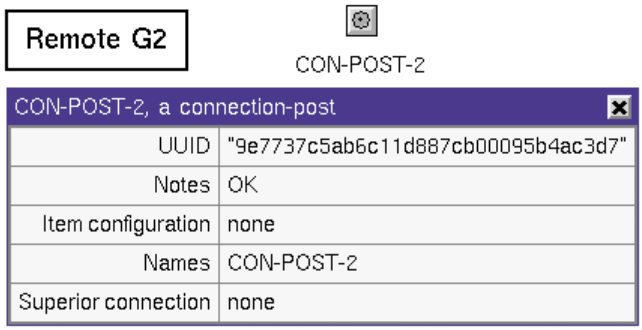
Here is the send-item procedure:

```
send-item(itm: class item)
begin
    call receiver-2(itm) across interface;
end
```

Clicking the button in the remote G2 calls receiver-2 across the network, which creates a new connection post named con-post-2 in the local G2 with this UUID:



This item is a copy of the original item in the remote G2, as the UUID for the original item in the remote G2 shows:



Including and Excluding Attributes

The next few sections describe the various ways in which you declare a remote procedure call to include or exclude user- and system-defined attributes through the item passing argument grammar.

An important point to note is that including or excluding any user- or system-defined attributes can be declared only once for each argument. Making such inclusions and exclusions can, therefore, be applicable to multiple items. As an example, an argument for a remote procedure xproc consists of a sequence. The sequence being passed to xproc consists of several instances of the same class:

```
sequence (my-object1, my-object2, my-object3, my-object4)
```

While the item passing argument grammar permits you to declare which attributes of the items in an item, sequence, or structure you wish to include or exclude, the specification applies to all items in the same argument. Thus, in the example given here, if the my-object class included attributes temperature and volume, and you specified the remote procedure declaration to include only the temperature attribute:

```
declare remote procedure xproc
    (sequence including only the user attribute temperature) = (truth-value)
```

all four objects would be passed with the temperature attribute, and none could include the volume attribute.

Passing an Item Including User-Defined Attributes

You can pass an item by explicitly including one or more of its user-defined attributes. Using the including only the user attributes grammar excludes all remaining user-defined attributes by default.

To pass an item including certain user-defined attributes:

➔ declare remote *procedure-name* (class *class-name*
including only the user {*attribute* | *attributes*:} *attribute-name* [,...]

Element	Description
class-name	The object class.
attribute-name	One or more user-defined attributes of the class, separated by commas. You can precede a single user attribute with the statement: including only the user attribute Such a declaration passes only the user attributes of the class that you list with the including only statement. It is incompatible with the excluding the user attributes: syntax.

When specifying more than one attribute, a colon (:) is required after the attributes statement, and the attributes themselves are separated with a comma (,).

Example of Passing an Item Including User-Defined Attributes

If you specify more than one class in the list of procedure arguments, separate the class and its attribute specifications within parenthesis, as in this example, passing a user-defined subclass of procedure as a procedure argument:

```
declare remote gds-proc-1
    (
        (class my-proc including only the user attribute height),
        (class my-proc including only the user attributes: height, width)
    )
    = (structure)
```

Passing an Item Excluding User-Defined Attributes

You can pass an item with all of its user-defined attributes, except for those you exclude. Explicitly excluding one or more user-defined attributes includes the remainder. By default, passing any item includes all of its user-defined attributes and none of its system-defined attributes.

To pass an item excluding certain user-defined attributes:

➔ declare remote *procedure-name* (class *class-name*
excluding the user {*attribute* | *attributes*:} *attribute-name* [,...]

Element	Description
class-name	The object class.
attribute-name	One or more user-defined attributes of the class, separated by commas. You can precede a single user attribute with the statement excluding the user attribute, omitting the colon. Such a declaration passes all of the user attributes of the class, except those listed with the excluding statement. It is incompatible with the including only statement.

When specifying more than one attribute, a colon (:) is required after the attributes statement, and the attributes themselves are separated with a comma (,).

Passing Attributes with Object Values

User-defined classes can include attributes whose value is an object. An attribute with an object value can be:

- A user-defined object.
- A list or array.
- A variable or a parameter.
- Any subclass of object.

Further, the object that is the value of an attribute can itself have attributes with object values.

When passing object values, keep in mind that:

- For objects, the remote G2 must include a compatible class definition for *every* object passed, including those included as object values. For a description of what constitutes a compatible definition, see Creating Compatible Definitions.
- For an object value that is given by a variable or parameter, G2 does *not* pass the value unconditionally.

To pass the value of one or more object attributes given by a variable or a parameter, you must explicitly include the statement:

with system attribute current value of variable-or-parameter

in addition to any other statements about which user- or system-defined attributes to pass. In the absence of this statement, G2 passes the object, but any attributes whose values are given by a variable or parameter do not have values.

Example of Passing an Attribute with an Object Value

In this example, the local G2 defines an auto class that includes a tire-pressure attribute, which is given by a float-variable. The purpose of the RPC is to pass:

- An auto object to the remote procedure, check-pressure.
- Include the float-variable object of the tire-pressure attribute and its current value.

To do this, use a remote procedure declaration such as the following:

```
declare remote
    check-pressure(class auto including only the user attribute tire-pressure
                        and including the system attribute current-value of
                            variable-or-parameter)
        = (class auto)
```

Passing an Item with System-Defined Attributes

You can pass one or more user-accessible system-defined attributes, or those that you include explicitly. The system attributes that you can pass are those that are accessible through the attribute access facility and that appear in the *G2 Class Reference Manual*.

While many system attributes are user-accessible, not all attributes for every item are available. For example, you can access most attributes of a workspace, but not the data structure that represents the items that reside upon a workspace. Thus, passing a workspace from one G2 to another results in a new workspace on the remote system, but without any of its associated items. Similarly, you can pass an item, but if the item has a subworkspace, its subworkspace is not passed to the remote system.

To determine which system attributes are accessible for each G2 item, see the *G2 Class Reference Manual*.

You can pass an item with one or more of its system-defined attributes. By default, passing any item *excludes* all of its system-defined attributes.

To pass an item including one or more system-defined attributes:

➔ declare remote *procedure-name* (class *class-name*
    including {all system attributes |
    the system {*system-attribute* | *system-attributes*:} *attribute-name* [,...]

Element	Description
class-name	The item class.
system-attributes	One or more user-accessible system-defined attributes. You can precede a single system attribute with the statement including the system attribute, omitting the colon. Such a declaration passes the system-defined attributes that you specify. You can specify these system-defined attributes explicitly: • name of item, which specifies the name of the item being passed • current value of variable-or-parameter • history of variable-or-parameter

If an item has more than one name and you specify:

```
including the system attribute: name of item
```

only the first name is passed.

Specifying a system attribute that is not applicable for the class, such as entering current value of variable-or-parameter when the item is not a variable or parameter subclass, passes the item to the remote G2 process with only its appropriate user- or system-attributes.

Note Using the special grammar for passing the name of an item, the current value of a variable or parameter, or the history of a variable or parameter, is the recommended way of passing each of these three system attributes.

Examples of Passing System-Defined Attributes

The next example specifies two arguments of a user-defined subclass of procedure. It passes all system-defined attributes with the first argument, and includes only two attributes with the second:

```
declare remote gds-proc-1
    (
        (class my-proc including all system attributes)
        (class my-proc including the system attributes:
            tracing-and-breakpoints, default-procedure priority)
    )
    = (integer)
```

This example shows the remote procedure declaration passing the current value and the history of an integer variable, and the name of a tank item:

```
declare remote special-attributes
    (
        (class integer-variable including the system attributes:
            current value of variable-or-parameter,
            history of variable-or-parameter),
        (class tank including the system attribute name of item)
    )

    = (sequence)
```

Passing Both User- and System-Defined Attributes

You can specify both user- and system-defined attributes for a single remote procedure argument, using any combination of values. Here is an example of how to pass history values along with a user-defined attribute.

```
declare remote
    pc-check(class pc including only the user attribute memory-size and
                    including the system attribute history of variable-or-parameter)
= (class pc)
```

Passing an Item with Attributes and a Handle

The ability to pass an item handle, in addition to any user or system attributes, has been implemented for use in G2 Gateway. While the grammar to support this functionality exists, using the optional [with handle] statement is not applicable to G2-to-G2 item passing.

Specifying One or More Remaining Arguments

After declaring the specific arguments of a procedure, you can optionally specify zero or more remaining arguments of one type using the all remaining statement as the last part of the remote procedure declaration.

To specify a remaining number of arguments:

➔ all remaining {*item-or-value* | *value* }

Example of Passing Remaining Arguments

This example declares a procedure with the first three arguments as:

- item-or-value
- value
- sequence

The remaining arguments are declared as all remaining item-or-value:

```
declare remote test-proc1
    (
        (item-or-value including only the user attribute foo),
        (value excluding the user attributes: volume, temperature
            and including all system attributes),
        (sequence including only the user attribute capacity),
        all remaining item-or-value
    )
    = (integer)
```

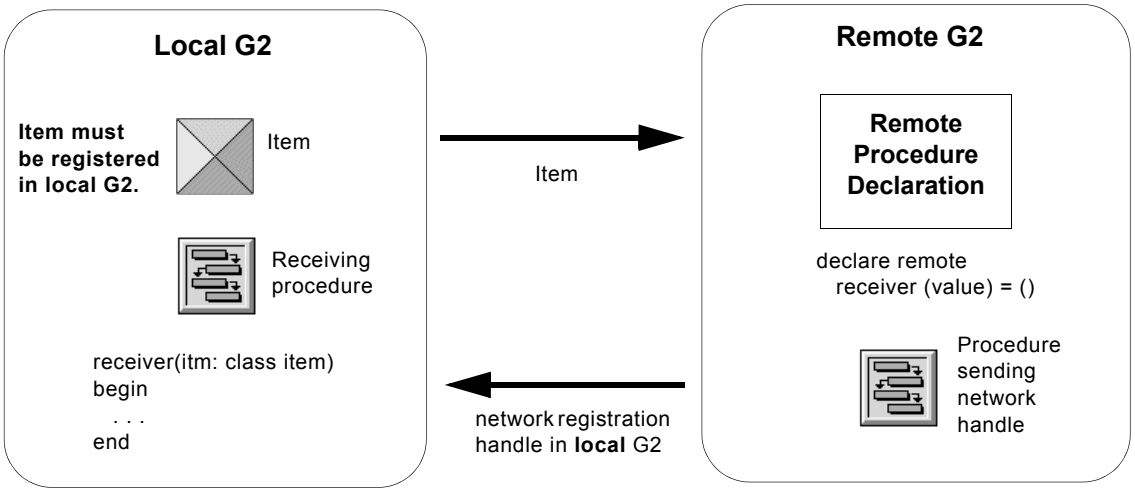
Passing Network Handles as the Class in RPCs

You can pass the network handle of an item as an argument to an RPC in remote G2, where the receiving procedure in the local G2 expects an item, and G2 attempts to replace the handle with the item before calling the procedure. If G2 does not find an item with that network handle, or if the handle is not of the class the procedure is expecting, it signals a type-mismatch error to the caller.

Note To call a G2 procedure with a network handle in order to rendezvous with an item, the procedure must declare its argument type to be a class of item; the procedure cannot declare it to be an item-or-value.

Note To use this feature, you must register the item in the local G2 and pass the local network handle as the argument to the RPC. For example, you might register a number of items in the local G2 and pass a list of network handles to the remote G2, which can then be used to call a procedure remotely in the originating G2 where item rendezvous can now occur.

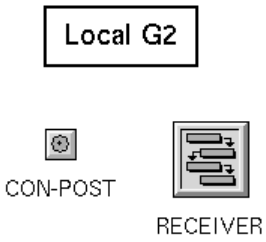
This figure illustrates how item rendezvous occurs when passing network handles:



This feature is similar to the feature whereby you can pass the UUID of an item to a G2 procedure via an RPC call when the receiving procedure expects an item. For details, see Passing UUIDs Referring to Items in RPCs.

Example of Passing Handles as the Class

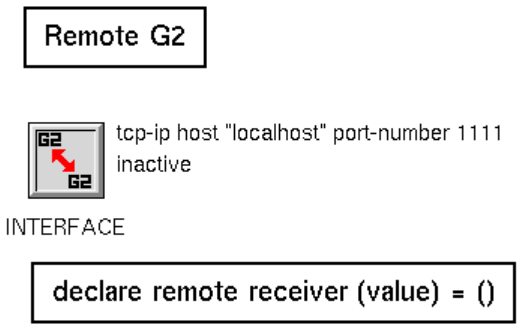
For example, in the local G2, suppose you have an item named con-post that you want to pass as the argument to a procedure named receiver, which you are calling from a remote G2:



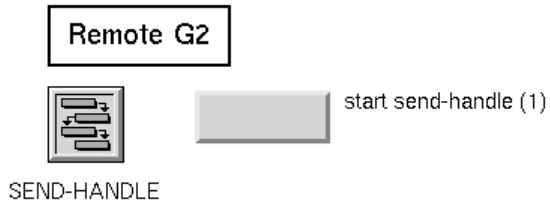
Here is the receiver procedure in the local G2, which takes an item as its argument and simply posts the name of the item to the Message Board. Notice that the procedure argument is declared to be a class of item, not item-or-value.

```
receiver(i: class item)
    begin
    post "received an item named [the name of i]";
end
```

In the remote G2, you would define a G2-to-G2 interface and declare the remote procedure. Notice that the remote procedure declaration takes as its argument a value, which is the argument type that is being passed to the RPC in the remote G2, namely, a network handle.



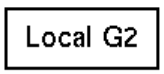
In the remote G2, you can make a remote procedure call to receiver across the network interface, passing the network handle as the argument, in this case, the integer 1. Note that this integer is the network handle of the item registration in the local G2, which you must generate locally and pass to the remote G2.



Here is the send-handle procedure, which makes the remote procedure call, passing the network handle as the argument, instead of the item:

```
send-handle(handle: integer)
    begin
    call receiver(handle) across interface;
end
```

Clicking the button in the remote G2 calls receiver across the network, replacing the network handle with the item, which posts the name of the item in the Message Board:

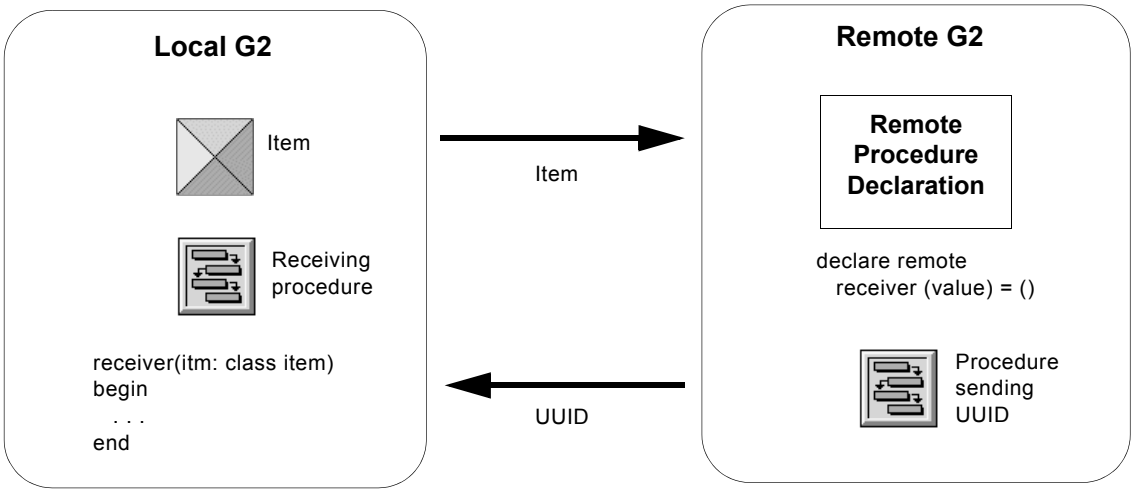


Passing UUIDs Referring to Items in RPCs

You can pass the text of the UUID of an item as an argument to an RPC in a remote G2, where the receiving procedure in the local G2 expects an item, and G2 attempts to replace the UUID with the item before calling the procedure. If G2 does not find an item with that UUID, or if the UUID is not of the class the procedure is expecting, it signals a type-mismatch error to the caller.

Note To call a G2 procedure with a UUID in order to rendezvous with an item, the procedure must declare its argument type to be a class of item; the procedure cannot declare it to be an item-or-value.

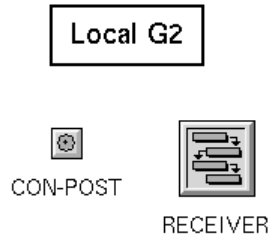
This figure illustrates how item rendezvous occurs when passing UUIDs:



This feature is similar to the previously undocumented feature whereby you can pass a network handle to a G2 procedure via an RPC call, where the receiving procedure expects an item.

Example of Passing UUIDs Referring to Items

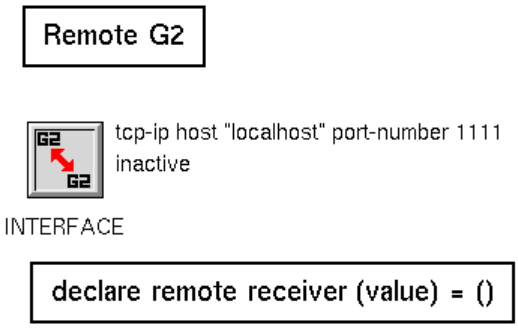
For example, in the local G2, suppose you have an item named con-post that you want to pass as the argument to a procedure named receiver, which you are calling from a remote G2:



Here is the receiver procedure in the local G2, which takes an item as its argument and simply posts the name of the item to the Message Board. Notice that the procedure argument is declared to be a class of item, not item-or-value.

```
receiver(i: class item)
    begin
    post "received an item named [the name of i]";
end
```

In the remote G2, you would define a network interface and declare the remote procedure. Notice that the remote procedure declaration takes as its argument a value, which is the argument type that is being passed to the RPC in the remote G2, namely, a UUID.



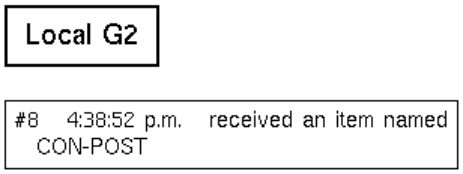
In the remote G2, you can now make a remote procedure call to receiver across the network interface, passing the text of the UUID as the argument:

Note You can also pass the UUID in compressed format; however, note that you cannot see the value of the UUID in compressed format like you can the text format.

Here is the send-uuid procedure, which makes the remote procedure call, passing the UUID as the argument, instead of the item:

```
send-uuid(uuid: text)
    begin
    call receiver(uuid) across interface;
end
```

Clicking the button in the remote G2 calls receiver across the network, replacing the UUID with the item, which posts the name of the item in the Message Board in the local G2:



G2 Gateway

Describes the system-defined items that permit GSI interfacing.

Introduction

The G2 Gateway standard interface (GSI) product is a network-oriented toolkit used for developing software interfaces, or **bridges**, between G2 and other external systems. G2 Gateway allows knowledge bases (KBs) to exchange various types of data between the G2 process and the bridge.

The G2 Gateway bridge is itself a process that communicates with G2 over the TCP/IP protocol, using a gsi-interface object.

This chapter describes the system-defined objects that enable data exchange between a G2 process and G2 Gateway. For a complete description of G2 Gateway and the objects that support it, see the *G2 Gateway Bridge Developer's Guide*.

Using G2 Gateway to Exchange Data

To enable communications between a G2 process and a bridge, you create one or more gsi-interface items, which specify the protocol to use, along with other relevant information about the remote process.

The GSI interface object acts as a doorway between G2 and the G2 Gateway bridge (formerly known as GSI). Through the interface object, the KB receives data from and sends data to a bridge process.

Both the G2 process and the bridge are capable of sending and receiving data. G2 has the ability to pass knowledge as follows:

Type of Exchange	Description
Value passing	Provides a value for a GSI variable from the bridge. GSI variables have GSI as their data server.
Item passing	Passes any item or object, with some set of its attributes, to the bridge process.

A G2 Gateway bridge process provides these additional capabilities:

- **Once-per-second G2 polling**, which permits G2 to obtain unsolicited data from the bridge process once every second.
- **External scheduling** (reporting by exception), indicating that value updates for variables occur automatically from the bridge process. G2 essentially turns off data seeking for the GSI variable, since it infers that updates will occur from the bridge. For a description of how G2 performs data seeking, see Obtaining Values for Variables.

Both of these capabilities are invoked by certain attribute settings in the GSI interface object, and are described fully in the *G2 Gateway Bridge Developer's Guide*.

From the G2 process, use remote procedure declarations to pass values, items, and objects to the GSI process. For a description of how to specify remote procedure declarations for data passing, see Using Remote Procedure Calls. The arguments are identical whether G2 is communicating to another G2 process or to a bridge.

G2 can also send text messages and acknowledgments to a bridge. Passing text and acknowledgments requires the use of GSI message objects.

You can use G2's publish/subscribe facility for event notification in distributed applications. For details, see Publish/Subscribe Facility.

Using GSI Interface Objects

The GSI interface object is an item of the gsi-interface class that lets you send values to, and receive values from, an external GSI process, using the TCP/IP communications protocol.

Creating a GSI Interface Object

To create a GSI interface object:

➔ Select KB Workspace > New Object > network-interface > gsi-interface.

The attribute table of a GSI interface object is:

a gsi-interface	
Notes	OK
Item configuration	none
Names	none
Identifying attributes	none
Interface warning message level	default to warning message level
Disable interleaving of large messages	no
Interface timeout period	use default
Interface initialization timeout period	unlimited
GSI connection configuration	none
External system has a scheduler	no
Poll external system for data	no
Grouping specification	no grouping
Remote process initialization string	""
GSI application name	default
GSI interface status	0
GSI interface is secure	no
Interval to poll external system	use default

For a complete description of each of the gsi-interface object attributes, see the *G2 Gateway Bridge Developer's Guide*.

Locating GSI Interface Objects on Activatable Subworkspaces

Locating GSI interface objects on an activatable subworkspace lets you control the objects programmatically. By activating or deactivating the subworkspace upon which a GSI interface object resides, you can activate or deactivate the object.

You can also use a conclude action to control activation. Concluding the gsi-connection-configuration attribute to have no value closes the connection.

Creating GSI Variables

A GSI variable is a variable subclass that includes gsi-data-service as one of its direct superior classes.

To create a GSI variable:

1. Define a subclass of any of the system-defined variable classes, and include the mixin class gsi-data-service as one of the direct superior classes. The mixin provides the additional attributes needed for GSI data service. Give the subclass any unique name.
2. Edit the Attribute-initializations validity-interval attribute to specify any time interval, or indefinite. Data servers other than the Inference Engine cannot have a validity interval of supplied, which is the default.

   You can also customize the new class in any other way.
3. Create an instance of the new class and open its table.

By using the gsi-data-service mixin class, the data-server attribute of the variable is set to GSI data server, and two additional attributes, gsi-interface-name and gsi-variable-status have been added.

Specifying the GSI Interface Name

The gsi-interface-name attribute specifies the name of the GSI interface object through which this variable will obtain values from the bridge. You must specify an interface object name to enable data services from the GSI process.

When creating GSI variables programmatically, complete this attribute last, since its completion causes the variable to become active immediately, unless the variable resides on an activatable subworkspace not yet activated.

Determining the Status of the Variable

The gsi-variable-status attribute indicates the status code of the external data point or the variable mapped to this GSI variable. You cannot change the value of this attribute since it is provided by the bridge process.

For information about using the attribute-initializations attribute and creating subclasses, see Specifying Default Values for Inherited Attributes.

Using GSI Message Servers

A GSI message server is a user-defined object or message class that includes gsi-message-service as one of its direct superior classes. Such items can send and receive text messages and acknowledgments from a bridge process.

To create a GSI message server:

1. Define a subclass of any item, object, or message class, and include the mixin class gsi-data-service as one of the direct superior classes. The mixin provides the additional attribute needed for GSI data service. Give the subclass any unique name.

   You can also customize the new class in any other way.

2. Create an instance of the new class and open its table.

By using the GSI message service class, the item includes the gsi-interface-name and the data-server-for-messages attributes. Enter the name of the GSI interface object through which you wish to transmit messages and acknowledgments. By default, the value of the Data-server-for-messages attribute is gsi-data-server.

Use inform actions to send messages to the bridge through the GSI message server. For example, if P1 is a GSI message server object with a temp attribute, a whenever rule such as this would inform the bridge about some event that occurred in P1.

```
whenever the temp of P1 receives a value and when the temp of P1 >= 200
    then inform P1 that "The temperature of [the name of P1]
        is above average."
```

1990

64

Interfacing with COM Applications

Describes the system-defined items that allow communication with COM appliations.

Introduction

G2 ActiveXLink enables you to establish communications between G2 and a COM-compliant application running under Windows XP, Windows 2003, or Windows 2000. This chapter discusses the `G2Gateway` control and the `WorkspaceView` control for use within COM-compliant containers.

For a detailed description of G2 ActiveXLink, see the *G2 ActiveXLink User's Guide*.

Using the G2Gateway Control

G2 ActiveXLink enables container applications and languages that support Microsoft COM, such as Microsoft Office, Microsoft Visual Basic, Visual C++, Microsoft Internet Explorer, and Active Server Page (ASP) to communicate with G2. G2 ActiveXLink provides the `G2Gateway` control, which:

- Enables users to invoke procedures in a G2 server, passing any number of arguments and returning any number of arguments with as little as a single line of code.
- Automatically maps data types.
- Supports both synchronous (blocking) and nonblocking calls.
- Can be used safely in multi-threaded applications because G2 ActiveXLink is thread-safe.
- Creates connections to multiple G2 servers at the same time.
- Automatically manages connections to the G2 server.
- Stores configuration information, such as the G2 server location as a visually configurable property.

Additionally, the G2 server can invoke logic in the COM-compliant container application with or without return arguments. Clients, the container applications, can post messages on the G2 Message Board.

The following Visual Basic code fragment shows how compact and powerful calls to `G2Gateway` can be:

```
Private Sub Form_Load()
   Call G2Gateway1.PostMessage("Hello from Visual Basic!")
   Call G2Gateway1.Call("My-Procedure",1,123,3.1415,True)
End Sub
```

The `Form_Load()` function automatically:

- Creates a connection to a G2 server.
- Posts a message to the G2 Message Board.
- Calls the G2 procedure my-procedure with four arguments.

The G2 server resides at the TCP/IP address specified in the `G2Location` property of the `G2Gateway1` object inserted in the Visual Basic form. The `G2Gateway1` object is an instance of the `G2Gateway` class in G2 ActiveXLink.

Managing G2 Items

G2Gateway is a control that you normally place on a form at design time, although it is not visible at run time. G2 ActiveXLink also defines a number of classes, which are not controls so they are not visible and are, therefore, only available at run time. These classes include:

- G2Symbol
- G2Structure
- G2Item
- G2List and G2Array
- G2Workspace
- G2Window

You use these classes to represent G2 items in you COM application. By default, a G2Item is a static copy of the item in G2. You can also create a G2Item so that is linked to the item in G2, which means the item updates automatically in both directions when changes occurs.

G2Item defines a number of methods for subscribing to various events on the item. These events occur on the G2Gateway to which the G2Item is linked. G2Gateway provides notification for these events: attribute changes, item deletions, icon color changes, variable or parameter value changes, and custom events.

Using the WorkspaceView ActiveX Control

The WorkspaceView ActiveX control allows you to view KB workspaces inside Microsoft COM-compliant containers, such as Internet Explorer or Visual Basic.

Note The WorkspaceView ActiveX control only works with Microsoft Internet Explorer, Version 4.0 or higher. We recommend that you use Version 5.5 or higher.

The WorkspaceView control connects to the G2 server through a G2Gateway ActiveX control, which is available as part of G2 ActiveXLink. Before you can use the WorkspaceView control, you must register the G2Com DLL. You must also register the control.

Each G2Gateway connection that displays a workspace view starts an embedded Telewindows, without a top-level window, and, therefore, consumes a Telewindows license. Each G2Gateway can display multiple workspace views. You are restricted as to the number of connections you can make, based on the number of Telewindows licenses you have.

The control provides properties and methods that allow you to:

- Connect to a `G2Gateway` instance.
- Specify the name or UUID of a KB workspace to show in the view.

65

Interfacing with Java Applications

Describes the system-defined items that allow communication with Java appliations.

Introduction

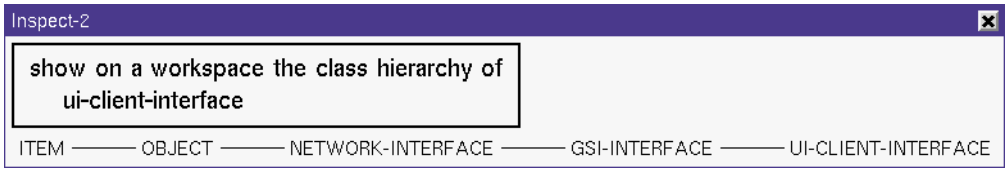
G2 JavaLink provides a set of Java components and classes that you can use to communicate with Java/RMI applications. To create G2 applications that interface with Java, you use the following classes:

- ui-client-interface
- ui-client-item
- ui-client-session

For information on how to use these classes, see the *G2 JavaLink User's Guide.*

Ui-Client-Interface

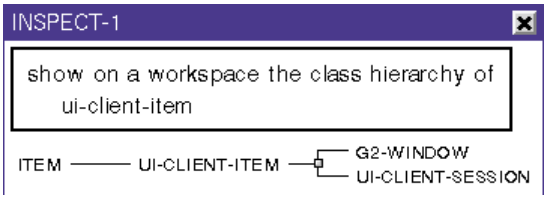
The ui-client-interface class is a subclass of the gsi-interface class as this hierarchy shows:



You create a ui-client-interface class to communicate between G2 and G2 JavaLink clients. For more information, see G2 Gateway.

Ui-Client-Item and Ui-Client-Session

The ui-client-item class is the superior class of the g2-window class and the ui-client-session class, as this hierarchy shows:



Typically, you refer to a ui-client-item as an argument to procedures and methods when you want to include the G2 window, as well as any G2 JavaLink clients. For example, the following G2 system procedure takes a ui-client-item as an argument to determine the remote process to kill:

```
g2-kill-remote-process
    (process-id: float, remote-win: class ui-client-item, timeout: value)
    -> process-killed: truth-value
```

When a G2 JavaLink client connects to G2, a ui-client-session is created for each connected client.

66

Interfacing with Web Services

Describes how to interface with Web service applications.

Introduction

According to the World Wide Web Consortium (W3C) Web Services Architecture Working Group Note (`http://www.w3.org/TR/ws-arch/`):

> A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

This chapter provides an overview and examples of how G2 can interface with Web service applications.

For detailed information on the system procedures used in this chapter, see Web Operations in the *G2 System Procedures Reference Manual*

Web Services

G2 can act as a Web service requester agent (client), using the following system procedures:

- g2-import-web-service-description — Creates a Web service description item from a URL on the Web.
- g2-import-web-service-description-from-xml-text — Creates a Web service description item from XML text.
- g2-invoke-web-service-operation — Invokes a remote web service operation.

These system procedures are located in `g2web.kb`, which is located in the `\g2\kbs\utils` or `/g2/kbs/utils` directory of your installation, depending on your platform.

Web Service Messages

A Web service message is a structure whose attributes correspond to WSDL message parts. The value of a message part attribute is either a text, an XML element value, or a sequence of XML element values.

An XML element value is a structure representing an XML element with this syntax:

structure
(tag-name: *text*,
attributes: *structure*,
children: *sequence*)

where:

- tag-name is the element tag name. This attribute is required.
- attributes is a structure containing named attribute values, which are texts. This attribute is optional.
- children is a sequence of XML elements and/or texts. This attribute is optional.

Attribute names use the same correspondence between XML names and G2 symbols used by G2GL, for example, `myAttribute` becomes my-attribute.

For example, this XML text:

```
<elt attrName="attrValue"><child>text1</child>text2</elt>
```

corresponds to this XML element value:

```
structure
(tag-name: "elt",
attributes: structure (attr-name: "AttrValue"),
children: sequence
    (structure (tag-name: "child", children: sequence("text1")), "text2"))
```

Importing Web Service Descriptions

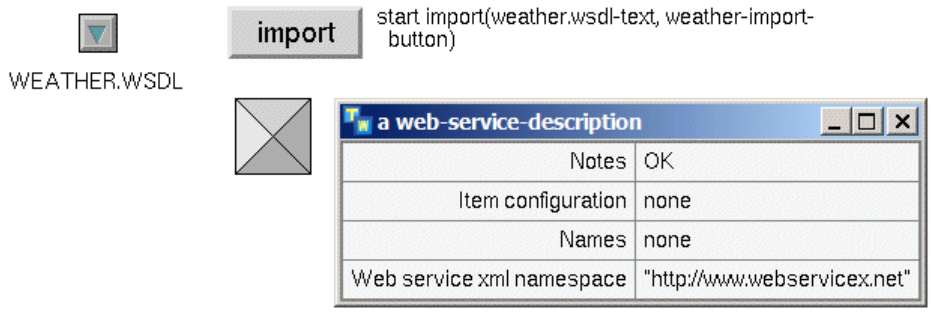
The first step in interfacing with a Web service is to create a Web service description object from the XML text that describes a Web service. G2 creates a web-service-description item based on the XML text.

For example, `http://www.webservicex.net/WeatherForecast.asmx?WSDL` provides the XML service description for a Web service that provides weather forecasts for a given location. To interface with the Web service, you create a Web service description object by calling the g2-import-web-service-description system procedure, providing the URL as the argument. This downloads the WSDL document from the Web. Alternatively, you can provide the WSDL document directly in a text, by calling the g2-import-web-service-description-from-xml-text system procedure.

The following procedure calls g2-import-web-service-description on a URL. The procedure transfers the resulting web-service-description object to the workspace of an item.

```
import (URL: text, item: class item)
description: class web-service-description;
begin
    description = call g2-import-web-service-description(URL);
    transfer description to the workspace of item at (the item-x-position of item,
    the item-y-position of item - 50);
end
```

This example shows the result of importing the XML text on the subworkspace of the weather.wsdl button. The resulting web-service-description object appears on the workspace of the import button. The Web service description object is defined in the `http://www.webservicex.net` target namespace.



Here is part of the free-text named weather.wsdl-text on the subworkspace of the import button:

Invoking Web Service Operations

To invoke a Web service operation, you call g2-invoke-web-service-operation with the following arguments:

- An endpoint-reference structure with this syntax:

  structure
  (service-namespace: *text*,
  service-name: *text*,
  endpoint-name: *text*)

- An operation name, as a symbol.
- An input message, which is a Web service message structure.

For example, the following procedure invokes the Web service operation named `GetWeatherByZipCode`. The input message is a structure with -ZIP-CODE as the attribute name, which corresponds to the `ZipCode` message part name. The endpoint name and output message part name are provided as arguments to the procedure.

```
weather-forecast-get-weather-by-zip-code(zip-code: text, endpoint-name: text,
    result-field: symbol)
endpoint-reference: structure =
    structure (service-namespace: "http://www.webservicex.net",
            service-name: "WeatherForecast",
            endpoint-name: endpoint-name);
operation-name: text = "GetWeatherByZipCode";
input-message: structure = structure(-ZIP-CODE: zip-code);
output-message: structure;
weather: sequence;
begin
    output-message = call g2-invoke-web-service-operation(endpoint-reference,
        operation-name, input-message);
    weather = call remove-whitespace(the value that is an attribute of
        output-message named by result-field);
    post "[weather]";
end
```

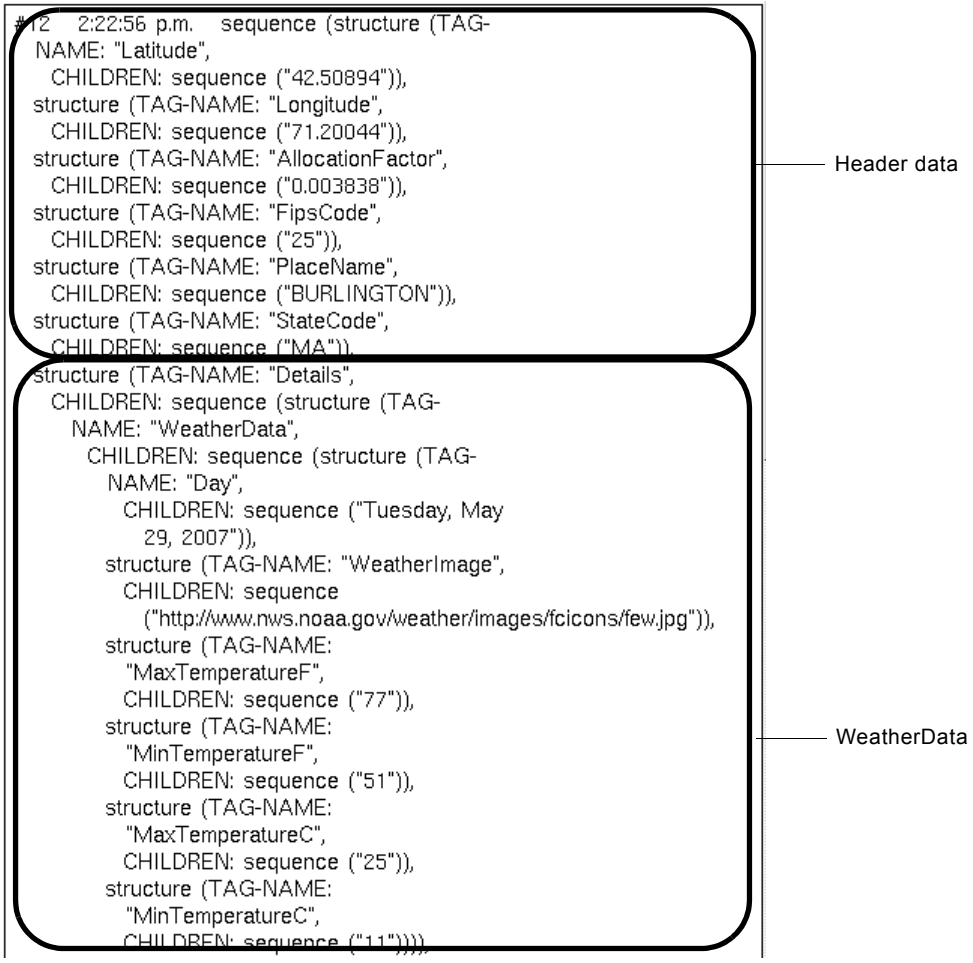
This action invokes the weather-forecast-get-weather-by-zip-code procedure for the given zip code, using a SOAP request. The -GET-WEATHER-BY-ZIP-CODE-RESULT result field corresponds to the WSDL element named `GetWeatherByZipCodeResult`.

```
start weather-forecast-get-weather-by-zip-code("04553", "WeatherForecastSoap",
    the symbol -GET-WEATHER-BY-ZIP-CODE-RESULT)
```

Similarly, this action invokes the weather-forecast-get-weather-by-zip-code procedure for the given zip code, using an HTTP request. The -WEATHER-FORECASTS result field corresponds to the WSDL element named `WeatherForecasts`.

```
start weather-forecast-get-weather-by-zip-code("04553", "WeatherForecastHttpGet",
    the symbol -WEATHER-FORECASTS)
```

Here is the resulting Message Board, which shows a sequence of XML element values with information about the zip code location and details about the weather for various days.



Invoking Web Service Operations from G2GL

You can invoke Web service operations from a G2GL Invoke activity by specifying the default-value-for-g2gl-variable attribute of a partner link variable to be an endpoint reference specification, to represent a link to a remote Web service, with the following syntax:

service-namespace: *service-namespace*, service-name: *service-name*, endpoint-name: *endpoint-reference*

The *endpoint-reference* is a structure with this syntax:

structure
(service-namespace: *text*,
service-name: *text*,
endpoint-name: *text*)

An Invoke activity that uses a partner link variable containing an endpoint reference invokes an operation on the remote Web service specified by the endpoint reference. The `g2\kbs\utils\g2web.kb` module must be loaded to enable G2GL to communicate with remote Web services.

Note that the Invoke activity waits for the operation to complete, even in the case of one-way communication where there is no reply; for example, invoking an HTTP operation waits for the HTTP response before continuing, because HTTP is a synchronous protocol.

For an example, see the Shakespeare demo in `g2web-demo.kb` located in the `g2\kbs\demos` directory.

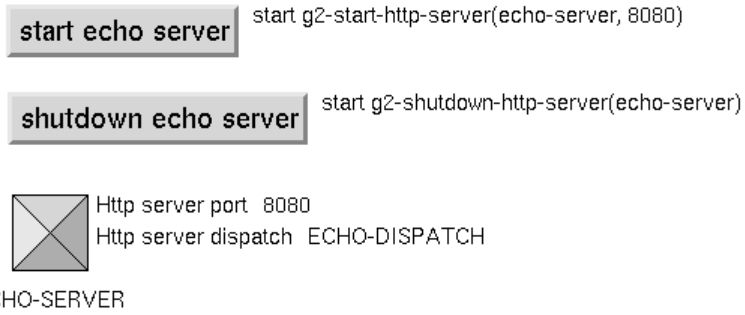
HTTP

G2 can act as a Web server and client, using the following system procedures:

- g2-start-http-server — Starts a task that listens on a given TCP/IP port for HTTP requests and passes them to the dispatch procedure of the specified server. The dispatch procedure processes HTTP requests on the server.
- g2-shutdown-http-server — Stops the listener task of the specified server.
- g2-send-web-request — Sends a request to a Web server at a given URL, returning the response when it arrives, where:
  - The request is a structure with these attributes: method, headers, and entity.
  - The response is a structure with these attributes: http-version, status-code, reason-phrase, headers, transfer-length, and connection.

Listening for HTTP Requests

The following example shows the result of starting an HTTP server that listens on port 8080 for HTTP requests. The echo-server object is an http-server, which specifies echo-dispatch as the http-server-dispatch procedure for handling HTTP requests. When the server is started, the http-server-port of the http-server is set to the specified port.



Here is the echo-dispatch procedure, which takes an http-server and a request structure as arguments. The procedure handles SOAP requests, HTTP requests, and HTTP file requests.

```
echo-dispatch(server: class http-server, request: structure) = (structure)
filestring: text;
response: structure;
begin
     post "Echo server received:
     [request]";
     if the path of request = sequence("soap") then
          response = call g2-handle-http-request-as-soap(server, request, soap-echo)
     else if the path of request = sequence("echo") then
          response = call http-echo(request)
     else begin
          filestring = call resolve-http-request-path(the path of request, server-root);
          response = call handle-http-request-from-file(server, request, filestring)
end;
  post "Echo server replied:
[response]";
  return response
end
```

For a description of handling HTTP requests as SOAP, see SOAP.

For the procedures used to handle HTTP requests and HTTP file requests, see the `g2web-demo.kb`.

Sending a Web Request

SOAP

G2 can send and receive SOAP 1.1 requests, using the following system procedures:

- g2-send-soap-request — Sends a SOAP request to a SOAP receiver at a given URL, returning the SOAP response when it arrives, where:

  - The request is a structure with these attributes: header-entries, body-entries, and action.
  - The response is a structure with these attributes, header-entries and body-entries.

- g2-handle-http-request-as-soap — Converts an HTTP request message into a SOAP request structure, passes it to a dispatch procedure, and converts the resulting SOAP response structure into an HTTP response. This system procedure is intended to be called by the http-server-dispatch procedure of an http-server, as shown in Listening for HTTP Requests.

For a description of XML elements for the request, see Web Service Messages.

Sending a SOAP Request

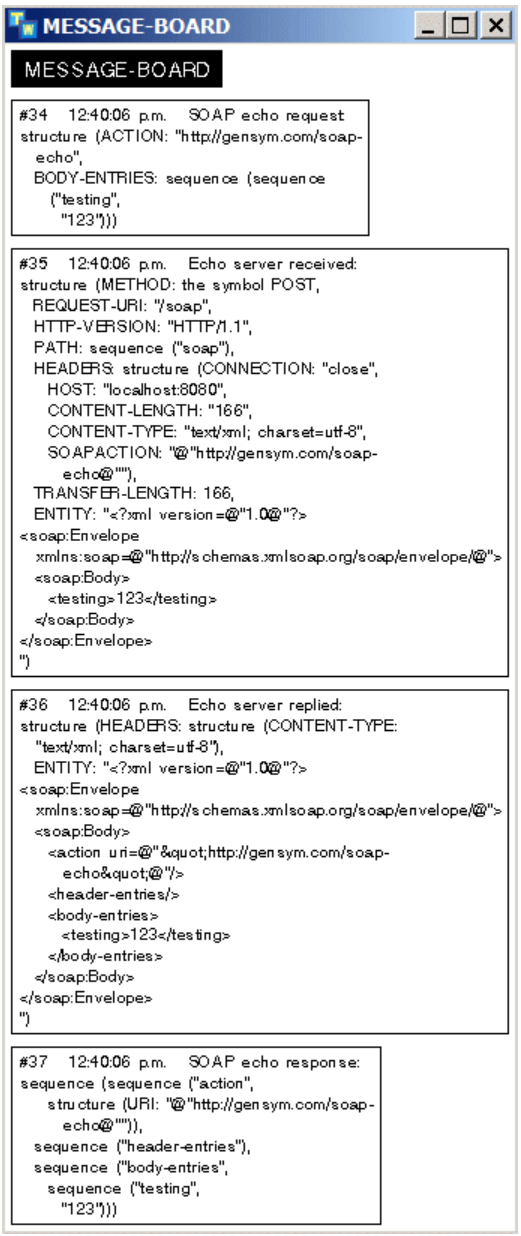
This procedure sends a SOAP request to an HTTP URL by calling g2-send-soap-request:

```
send-soap-echo-request()
URL: text = "http://localhost:[the http-server-port of echo-server]/soap";
request: structure = structure(
     action: "http://gensym.com/soap-echo",
     body-entries: sequence(sequence("testing", "123")));
response: structure;
body-entries: sequence;
begin
     post "SOAP echo request:
     [request]";
     response = call g2-send-soap-request(URL, request);
     body-entries = call remove-whitespace(the body-entries of response);
     post "SOAP echo response:
     [body-entries]";
end
```

Here is the soap-echo procedure that is the action of the SOAP request:

```
soap-echo(server: class http-server, request: structure) = (structure)
header-entries: sequence = sequence();
begin
     if the header-entries of request exists then header-entries = the header-entries of
          request;
     return structure (body-entries: sequence(
          sequence("action", structure(uri: the action of request)),
          insert-at-beginning(header-entries, "header-entries"),
          insert-at-beginning(the body-entries of request, "body-entries")));
end
```

This figure shows the result of calling send-soap-echo-request:



67

Interfacing with TCP/IP Sockets

Describes the system-defined items that allow communication with TCP/IP sockets.

Introduction

This chapter describes the classes and system procedures for communicating with TCP/IP sockets.

For details, see Network Reading and Writing in Network Operations in the *G2 System Procedures Reference Manual*.

TCP/IP Socket Communication

G2 provides the g2-socket class and various system procedures for managing network connections, using TCP/IP sockets, such as HTTP, and performing input/output operations to read and write data.

You use the following system procedures to manage connections to TCP/IP sockets, all of which allow other processing:

- g2-tcp-connect
- g2-tcp-listen

- g2-tcp-accept
- g2-tcp-close

Socket I/O

The following system procedures, implemented as methods on the g2-socket class, write data to a socket:

- g2-write-string
- g2-write-bytes

The following system procedures read data from a socket:

- g2-read-block
- g2-read-byte
- g2-read-bytes-as-text
- g2-read-bytes-as-sequence
- g2-read-line

In general, the system procedures that perform I/O through sockets uses the same procedure names as the system procedures that perform I/O using streams. However, note that the I/O system procedures for both sockets and streams are implemented as methods rather than as procedures. All system procedures allow other processing.

Foreign Functions

Describes how to call C or C++ foreign functions from within G2.

Introduction

In G2, the term **foreign function** refers to a function written in C or C++ code that a KB can access as if it were a local function. The foreign function interface is platform-independent and efficient, allowing you to isolate G2 from the effects of possible coding errors.

The term **foreign image** describes an executable file, external to G2, that contains the foreign functions you plan to call from your KB.

A foreign function call is synchronous. G2 does not perform other tasks until the function returns or times out. For asynchronous calls to C functions, see the

description of GSI remote procedure calls, described in the *G2 Gateway Bridge Developer's Guide*.

You can start foreign images two ways:

- As an external process, independent of G2 control.
- By spawning a process from within G2.

Several advantages to using external foreign image processes are:

- You can run a foreign image on a computer other than the one on which G2 is running.
- Using an external foreign image can prevent excessive memory usage on certain platforms during process spawning.
- A separately running foreign image is easier to debug.

Note On Windows platforms, you need G2 Gateway in order to build foreign functions. If you have installed the G2 Bundle without G2 Gateway, you must install this component in order to build foreign functions.

Foreign Functions Examples

The `ext` subdirectory of the G2 product directory contains files that help you build a sample foreign image.

The `samples` subdirectory of the `kbs` directory in the `g2` directory contains `fgntest.kb`, which contains examples of foreign function invocation.

Creating a Sample Foreign Image

The files for creating a sample foreign image are:

File	Description

`fgntest.tpl`	A template file that includes several functions, and their argument descriptions. The sample functions are:
	`cc_add_integers`
	`cc_add_reals`
	`cc_append_text`
	`cc_append_symbols`

`fgntest.c`	A sample C file that contains the functions listed in the template file, and which can be linked into a foreign image.
`foreign.h`	The header file to include for foreign functions.
`makefile`	A makefile that you complete to build an executable foreign image, and which invokes the `overlay.exe` program to build an executable foreign image: `fgntest.fgn`.
`overlay`	The Overlay utility, which you use to build an executable foreign image. Depending on the platform, this file may have a `.exe` extension.

To create a sample executable foreign image:

➔ Use the `makefile` to build an executable foreign image called `fgntest.fgn`.

Creating `fgntest.fgn` makes an executable from which the foreign functions in `fgntest.c` can be called from G2.

Calling the Sample Foreign Functions

Once the executable foreign image exists, you can use its functions from within G2 by loading the `fgntest.kb` file, which includes sample foreign function declarations that use the functions declared in the foreign image.

To call the sample foreign functions available in fgntest.fgn:

1 Start G2.

2 Load this KB:

`\g2\kbs\samples\fgntest.kb` (Windows)
`/g2/kbs/samples/fgntest.kb` (UNIX)

3 One way to connect to the foreign image you created is by choosing Main Menu > Miscellany > Connect To Foreign Image.

Starting and Connecting to the Foreign Image and Connecting to an External Process Foreign Image provide more information about starting a foreign image.

4 Enter the name of the foreign image you created.

After connecting to the foreign image, you can use the simple examples of invoking foreign functions that the `fgntest.kb` provides.

Using Foreign Functions

To use foreign functions, you collect existing C source files into an executable foreign image to which G2 connects. Gensym provides utilities to help you complete the steps for creating and using a foreign image.

To use foreign functions in a KB:

1 Create the template file based on the functions in one or more C source files.

2 Use the Gensym-provided makefile as a basis for using the **Overlay utility** to create an overlay C source file from the template file and for compiling and linking the appropriate files.

3 If the foreign image is a separate external process, start it. The foreign image must be running by the time the connect to external foreign image command is executed in G2, as described in Starting and Connecting to the Foreign Image.

4 If the foreign image is not presently running as an external process, connect to the foreign image from within G2. Note that connecting also starts the foreign image, unless you are connecting to an external image, as in Step 3. For a description of connecting to the image from within G2, see Connecting to an External Process Foreign Image.

5 Create a foreign function declaration in G2 that indicates the name of the functions.

6 Use the foreign function as you use any other function within G2.

7 Disconnect from the foreign image when you no longer need its functions. Disconnecting does not cause an external foreign image to exit, although it does terminate a foreign image spawned by G2.

Note Typically, you complete steps 1 – 3 at the command line of your system (independently from G2) and steps 4 – 7 from within G2. To automate one or more of the command line steps, you can use the system procedure g2-spawn-process-to-run-command-line along with other various file accessing system procedures, described in the *G2 System Procedures Reference Manual*.

The remaining sections describe each of these steps, which you should complete sequentially.

Creating a Foreign Function Template File

This is the first step to using a foreign function. The **template file** declares the names, arguments, and return value of each of the foreign functions in the foreign image. Each foreign function can return only a single value.

While there is no limit to the number of arguments for a foreign image, each symbol or text argument or result cannot exceed 64K.

Gensym provides the `fgntest.tpl` template file, located in the `\g2\ext` (Windows) or `/g2/ext` (UNIX) subdirectory of your G2 product directory. Create a new template file, or edit the sample template file, with any text editor. The template file describes the arguments and return values of each function in the existing C source file. The template file can have any name and extension. The example here uses the sample file name `fgntest.tpl`.

The template file lists each function on a separate line, using the syntax:

return-type function-name-in-c function-name-in-G2 argument-types

Template Syntax	Description
return-type	The G2 data type of the value the foreign function returns. Foreign functions return a single value. They do not return multiple values or non-values as do C structures or pointers. Possible data types are integer, float, symbol, and text.
function-name-in-c	The name of the foreign function as it appears in the original C source code. In keeping with C coding rules, do not use hyphens in names.

Template Syntax	Description
function-name-in-G2	The name of the function as it will appear in the foreign image, and thus what you will call in G2.
argument-types	The number and data types of the C function arguments. Possible argument types are integer, float, symbol, and text.

C and C++ Data Types and Character Conversion

The data types of C and C++ convert to the G2 data types as the next table describes:

C Or C++ Data Type	G2 Data Type
`double`	float G2 floats are transformed to and from C doubles in IEEE format.
`long`	integer G2 integers are transformed to and from C longs. It is not possible to get more than 30-bit integers in G2 by returning them from a foreign function; the values simply wrap around to fit inside 30 bits. If you need more precision, consider casting to type double and using G2 type float.

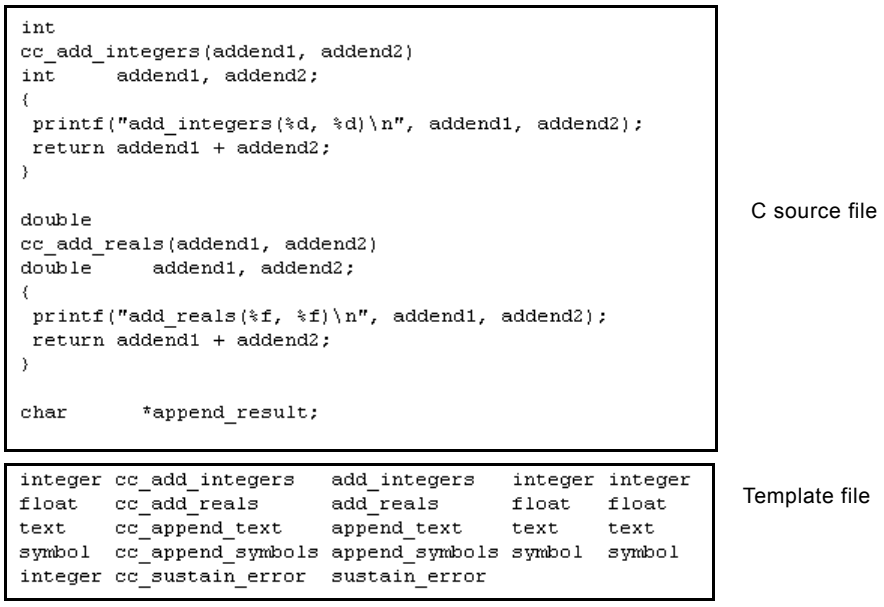
C Or C++ Data Type	G2 Data Type
`*char`	text G2 text is transformed between C character arrays as is, including G2's internal representations of some character sequences. Due to G2's internal representation of text characters, we recommend against returning anything other than alphanumeric characters from a foreign function. As a special case that is useful with special character sets, it is permissible to return a text string exactly as received from G2 as the argument of a foreign function.
`*char`	symbol G2 symbols are transformed to and from C character arrays, so the previous description is applicable.

A single template file can contain functions from multiple C source files, as long as each function appears on its own line.

For example, the next diagram shows part of the sample C source file, `fgntest.c`, containing two functions that you want to access from within G2:

```
cc_add_integers
cc_add_reals
```

The diagram also shows the template file you would create to hold information about the functions that the original C source file contains:



You should now have two files:

- The original C source file(s), `fgntest.c`.
- The new template file, `fgntest.tpl`, which you require for the next step.

Using the Overlay Utility through the Makefile

The next step to creating a foreign image is to use the Overlay utility and to compile and link the appropriate files.

Gensym provides the Overlay utility specifically for use with foreign functions. The utility takes the template file as an input file and outputs a C source file, called an **overlay file**, which marshals the arguments and return types for use by G2. The Overlay utility resides in the `ext` subdirectory of the G2 product directory.

Gensym also provides the `foreign.h` and the `icp.h` header files, and the `libforgn.lib` library, which are all used by the makefile as part of compiling and linking the files.

In conjunction with using the Overlay utility, Gensym also provides a platform-specific makefile, also located in the `ext` subdirectory. The makefile calls the Overlay utility, and compiles and links the files you need to create the foreign image.

Completing the Makefile Global Variables

The makefile that Gensym provides is an example. As such, you need to complete several global variables before using the makefile:

Use this variable name...	For the...
C_DIR	Location of the C source file if other than the makefile directory.
OBJ_DIR	Location of the obj files if other than the makefile directory.
EXE_DIR	Directory in which to place the resulting executable.
TEMPLATE_DIR	Location of the template file you created if other than the makefile directory. By default, the makefile assumes that the template file resides in the same directory as the C source file, so you do not have to complete this value.
OVERLAY_DIR	The location of the `overlay` executable.
H_DIR	The location of the required header files.
CC_FLAGS	
LIB_DIR	The location of the required libraries.

If you are building the example foreign image with the files Gensym provides, edit the makefile as the preceding table describes.

If you are building a foreign image with your own C source and template files, edit the makefile as the preceding table describes *and* include the appropriate source and template files.

Note The name that you provide for the executable foreign image must have a suffix of `.fgn` so that G2 can locate it in your directory. In the sample makefile, the file name is `fgntest.fgn`.

Running the Makefile

On UNIX platforms, run the Gensym makefile as you would any other makefile on your platform.

On Windows platforms, you must use the Microsoft Visual C++ compiler, version 6.0 or higher.

Starting and Connecting to the Foreign Image

To use a foreign image once it exists, the image must be started as a process, and G2 must connect to the foreign image. The two ways to start a foreign image are:

- From the command line by starting the foreign image as a process. If you start a foreign image a s separate process, you must then connect to the image from within G2 as described in Connecting to an External Process Foreign Image.
- From within G2 by connecting to the foreign image as described in Starting a Foreign Image from within G2.

Starting the Foreign Image as an External Process

To start the foreign image as a separate process:

➔ Enter the command for your operating system.

The sample commands use the example foreign image, *fgntest.fgn*.

Operating System	Example
UNIX	*fgntest.fgn* `1234` where *fgntest.fgn* is the name of the foreign image file you compiled and linked, and *1234* is the TCP/IP port number to which you want G2 to connect.
Windows	*fgntest.fgn* `1234` where *fgntest.fgn* is the name of the foreign image file you compiled and linked, and *1234* is the TCP/IP port number to which you want G2 to connect. The name you substitute for *fgntest.fgn* must have a maximum of eight characters to work in the Windows file system.

The sample commands associate the operating system UNIX and Windows with the network type TCP/IP. On some platforms, the opposite network type is also available. For example, entering a numeric-only value after *fgntest.fgn* directs the foreign image to use a TCP/IP network connection type.

Connecting to an External Process Foreign Image

If you start a foreign image as an external process, you must connect to it from within G2 before using the foreign image.

To connect to a foreign image that is an external process:

1. Pause G2 to be able to get to the Connect to Foreign Image menu choice.
2. Select Main Menu > Miscellany > Connect to Foreign Image.

   G2 displays the Text Editor with the statement Connect to Foreign Image, followed by a path name.
3. Enter Control + x to remove the current text in the Editor and to display a set of prompts.
4. Selecting the prompts, enter this command:

   connect to external foreign image at *connection-specification*

   and complete the *connection-specification*.

The syntax for *connection-specification* to connect to a foreign image over a TCP/IP connection is identical to the syntax for the icp-connection-specification attribute of a G2 data interface object. For more information about this syntax, see Defining the Connection Details.

The values you enter for the *connection-specification* arguments are determined by the values you used to start the foreign image process, as described in Starting the Foreign Image as an External Process.

Starting a Foreign Image from within G2

You can start a foreign image from within G2.

To start a foreign image:

1. Pause G2 to be able to get to the Connect to Foreign Image menu choice.
2. Select Main Menu > Miscellany > Connect to Foreign Image.

   G2 displays the Text Editor with the statement Connect to Foreign Image, followed by a path name.
3. Enter the path name for the foreign image file so that the command appears as:

   Connect to Foreign Image *foreign.fgn*

   where *foreign.fgn* is the name of the foreign image file, including a path if necessary. For example, to start the sample file on Windows, enter a command such as:

   ```
   Connect to Foreign Image "c:\gensym\myfiles\fgntest.fgn"
   ```

Connecting to a foreign image from within G2 in this way starts the image and connects to it.

Connecting to a Foreign Image with a G2-Init File

Further, you can add a `connect to external foreign image` command to an initialization file as long as you arrange for the image to be running by the time G2 issues the connect to external foreign image command. For information about using an initialization file, see Using an Initialization File.

Declaring a Foreign Function in a KB

Before using a foreign function within an expression, you must declare it to G2.

To declare a function as foreign:

➔ Select KB Workspace > New Definition > foreign-function-declaration.

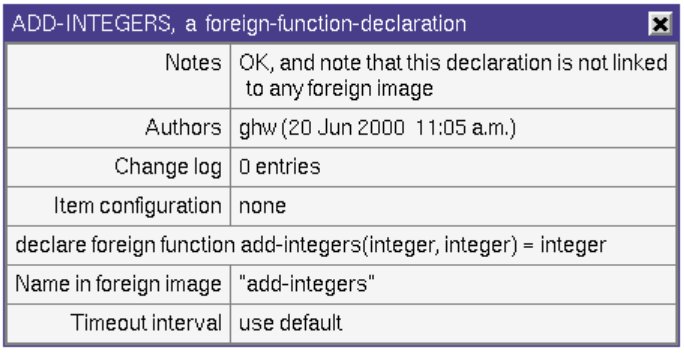
Selecting this declaration creates an instance of the foreign-function-declaration class, and invokes the Text Editor immediately so that you can complete the declaration.

The following figure shows an example declaration of one of the sample functions used throughout this chapter, `g2-demo-add`.

```
declare foreign function add_integers(integer, integer) = integer
```

The declaration text of this item is the declare foreign function statement (followed by the name of the function, its arguments and return type) that has been compiled and linked into the foreign image to which G2 is connected.

After you close the edit of the foreign function declaration, place the declaration on the workspace, and open its attribute table. The next figure shows such an attribute table with the declaration text:



After connecting to a foreign image, G2 maintains an internal list of the functions contained in the foreign image file. When you enter the foreign function name, G2

checks the arguments and return value against the information it has in the internal list. If a function of the name you enter does not exist, or if argument or return value discrepancies do exist, G2 displays an appropriate message in the notes attribute of the foreign function declaration.

The class-specific attributes of foreign-function-interface items are:

Attribute	Description
name-in-foreign-image	The name of the C function as it appears in the foreign image.
Allowable values:	Any foreign function name, text string.
Default value:	The name of the function in the declaration text, in lowercase.
timeout-interval	The amount of time before G2 times out while waiting for a foreign function to return.
Allowable values:	any positive value up to 536870 seconds (6 days, 5 hours, 7 minutes, and 50 seconds) use default
Default value:	use default (the value of the foreign-function-timeout-interval attribute of the timing parameters system table)

Providing the Name of the C Function

The name-in-foreign-image attribute indicates the exact name of the C function as it appears in the foreign image (the name you entered in the template file). By default, the value of this attribute is a lowercase version of the function name you enter in the declaration text.

If you edit the default name provided here, subsequently editing the declaration text will *not* update the name-in-foreign-image attribute.

For instance, if you enter a function name as my-foreign-function in the declaration text, that name will appear in this attribute. If you then change the value to:

```
"my-foreign-function"
```

(using underscore characters instead of dashes as word separators), G2 will *not* change this value if you edit the declaration text later. Instead, you must edit this attribute directly.

Setting the Timeout Interval

The timeout-interval attribute determines the amount of time allowed before G2 times out while waiting for a value from a foreign function. The value of this attribute (other than use default) overrides the foreign-function-timeout-interval value set in the Timing Parameters system table, which is 30 seconds by default.

Caution Foreign functions run synchronously. G2 will not perform any task until the foreign function returns or times out.

Typically, when G2 fails to receive a value from a foreign function, it means that the foreign image has exited due to a coding error.

Handling Possible Name Collisions

G2 maintains a name-to-image association between the name-in-foreign-image attribute and the function name as it appears within the foreign image.

Because G2 can connect to more than one foreign image, the possibility of a name collision exists. A name collision occurs when a foreign image transmits a foreign function name for which G2 already has an association. G2 may have obtained that name from a different foreign image or from a name previously transmitted by the foreign image to which G2 is connecting. When checking for name collisions, G2 distinguishes between upper-and lower-case names.

If G2 detects a name collision, it issues a warning in the logbook and replaces the old name-to-image association with the new one. Although G2 can detect the collision and issue a warning, we recommend that you make sure that each foreign function in your KB has a distinct name and that you do not depend on this behavior.

Using a Foreign Function

After you have connected to, and declared a foreign function in your KB, you can use that function in a statement. Using the add_integers example, the following diagram shows the text of a procedure. The procedure includes two integers as its arguments, passes those integer values to the external foreign function, add_integers, which adds the values and returns the total.

```
foreign_add_ints(X: integer, Y: integer)
result: integer;
begin
    result = add_integers(X, Y);
    post "The result is [result]."
end
```

Note G2 does not perform other processing while waiting for a foreign function to return. If you expect processing to take a long time, you may wish to adjust the value of the foreign-function-timeout-interval attribute in the Timing Parameters system table.

Disconnecting from the External Foreign Function

To disconnect from an external foreign image:

➔ Select Main Menu > Miscellany > Disconnect From Foreign Image.

which invokes the Text Editor for Disconnect From Foreign Image so that you can enter the name of the foreign image.

By default, the Text Editor displays the disconnect from foreign image statement, followed by a default path name.

To select the Disconnect from external foreign image at option, click after the from in the default statement to display the available disconnect prompts. Once the Disconnect from external foreign image at statement displays, the Text Editor prompts include the syntax for network connections.

If you select the per directory menu syntax when disconnecting G2 from a foreign image, the Text Editor prompts include both the names of any foreign image files (*filename.fgn*) to which G2 is already connected, and the network connections of foreign images to which G2 had been previously connected.

Choosing the appropriate item disconnects G2 from the associated foreign image.

Note Disconnecting from a spawned foreign image kills the process. Disconnecting from an external foreign image disconnects, but does not kill the process. The external image is then ready for future connection requests

2024

Windows Services

Describes how to run G2 and G2 bridges as a service under Windows.

Introduction

GService allows you to install and manage G2 and G2 bridges as services under Windows. You may use this utility to install any number of G2 or bridge processes as services as long as you provide a unique service name for each installed service. GService runs each service as a separate process. GService is installed in the `g2` directory of your G2 installation directory.

To use GService, you must have sufficient privileges to make modifications to the Service Control Manager and Windows Registry. We recommend that you log on with administrative privileges. You can also install and run GService by logging on with a local user account that supports the "Log on as a service" policy. To do this, choose Control Panel > Administrative Tools > Local Security Policy, then choose Local Policies > User Rights Assignment > Log on as a service, then choose Properties and add the user or group for that policy.

The following are benefits to running G2 processes as services:

- Allows G2 or a bridge to start and service requests when the operating system (or machine) starts, even when no user is logged on. This is very desirable if you wish G2 or a bridge to restart following a power failure or machine reboot, which is especially important when running mission-critical applications in an unattended mode.
- Allows G2 or a bridge to survive logoff/logon sequences, which saves the overhead and inconvenience of restarting them for each new user.
- Allows G2 or a bridge to run under a specific login account, different from the currently logged-in user. This is desirable when you have security concerns or account privilege issues.

The main features of GService are the ability to:

- Install, remove, start, stop, and obtain information about any service.
- Direct console output from the service and the G2 or bridge process to a log file.
- Direct service and G2 or bridge process output to a console window that is automatically created when the service starts.
- Specify command-line arguments for use by the G2 or bridge process when it is started, such as port number, network transport type, and so on.

Running GService

You run GService from the command-line, using the following syntax:

```
gservice command service-name [install-argument-spec] [...]
```

Note All references to files in command-line arguments must be to files on the local machine; you cannot install a service over the network or specify parameters that refer to files located on the network.

Argument	Description
command	Specifies the command to run. The options are: • `install` Installs a service in the Service Control Manager and allows you to specify various run-time options, that is, install arguments, for the service. • `remove` Removes or uninstalls a service from the Service Control Manager. • `start` Starts an installed service. When the service is started, the G2 or bridge process is also started. • `stop` Stops an installed service. When the service is stopped, the G2 or bridge process is also terminated. • `info` Displays information about the installed service such as log file name, bridge name, and so on. • `debug` Runs the service in a console window but does not start G2 or the bridge in the Service Control Manager. This is useful for diagnosing problems with the G2 or bridge process.
service-name	Any valid text string. The name must not contain spaces. The name must be unique, that is, no other service may have the same name.

Argument	Description
install-argument-spec	Used to specify certain run-time configuration options for the service and to indicate which G2 or bridge to run. You can specify one or more install arguments, separated by spaces, using the following syntax: *install-argument=value* Following are the install arguments that you can specify when you install a service.
	• `program`=*executable-path-name* (Required) Specifies the G2 or bridge process to run when the service starts. You must specify the full path to the G2 or bridge executable name. If the path includes spaces, you must surround it with double-quotes.
	• `console`=*truth-value* Indicates whether to create a console window when the G2 or bridge process starts. This option is `true` by default. If a log file is specified and this option is `true`, then you can only receive startup messages for the service. If a log file is not specified, all messages from both the service and the G2 or bridge process are directed to the console window. **Note:** If you start GService with the `colsole=false` argument, a G2 window automatically appears. To start G2 with no window, you must explicitly start G2 with the `-no-window` command-line option.

Argument	Description
	• `dependson=` *service* [, ...] Causes GService to ensure that another service is running before starting the current service, where *service* is the name of an existing service. For example, you might want to ensure that a bridge service starts before starting G2 as a service. With Windows 2000, you can check your service properties for dependences by choosing Start > Programs > Administrative Tools > Services to verify successful installation of the dependencies. Note that this install argument requires the actual service name as opposed to the application display name, which may be different. It is up to the user to determine the appropriate name for the service. If the name is misspelled, for example, it fails to enter the dependency.
	• `logfile=`*log-file-path-name* Indicates the log file name that will receive console messages from the service and G2 or bridge process. If you specify a log file, console output is automatically redirected to the log file. A log file is not required. If the path includes spaces, surround it with double-quotes.
	• `module-search-path="'`*search-path-2*`' '`*search-path-2*`' ..."` Provides an alternative to using the `parameters` specification with the `-module-search-path` keyword, which requires backslashes to allow nested quotes when specifying path names with spaces.

Argument	Description
	• `parameters="`*command-line-arg* [, ...]`"` Allows you to specify command-line arguments for the G2 or bridge process. This is useful if you wish to specify a port number or network transport. You must surround the list of parameters with double quotes. If a pathname within the list of parameters contains spaces, you must surround it with single quotes. **Note:** When starting G2 as a service, the default directory is no longer the directory in which the G2 executable resides. Therefore, unless you have defined an environment variable for the location of the `g2.ok` file, you should specify `parameters: "-ok` *ok-file*`"` to identify its location. **Note:** Similarly, we recommend that you use the `-log` command-line option when starting G2 to specify the location of the G2 log file; otherwise, it might be difficult to find the log file. **Note:** When running G2 as a service, you must start G2 with the `-no-tray` command-line option to suppress the G2 server icon in the notification area; otherwise, an error occurs when you start G2 as a service.
	• `restart=`*truth-value* Indicates whether to restart GService when the installed service terminates, either intentionally or due to an error. When `true`, Windows restarts the service after 60 seconds when it terminates. When `false`, the default, the service will not restart when it terminates, regardless of its error status.

Caution When starting a service with a console, you should avoid closing the console without first stopping the service; otherwise, the process remains running the background and you cannot stop it, using either the Service Control Manager or the Windows Task Manager. If this happens when running G2 as a service, Telewindow into the service and shut it down. If this happens when running a G2 bridge, you must reboot your computer to shut down the process.

Examples

The following examples are one-line commands specified in a command window.

Examples of Using GService with a Bridge Process

To install a service named `myservice` that starts the G2-ODBC Bridge on port number 22055:

```
gservice install myservice program="c:\Program Files\Gensym\
  g2-2011\odbc\g2-odbc.exe" parameters=22055
```

To install a service named `myservice` that starts the G2-ODBC bridge and disables console output:

```
gservice install myservice program="c:\Program Files\Gensym\
  g2-2011\odbc\g2-odbc.exe" console=false
```

To install a service named `myservice` that starts the G2-ODBC bridge and redirects output to a log file:

```
gservice install myservice program="c:\Program Files\Gensym\
  g2-2011\odbc\g2-odbc.exe" logfile=c:\temp\odbc.log
```

To install a service named `myservice` that starts the G2-ODBC Bridge on port number 22055 and to shut down the service if the port is not open:

```
gservice install myservice program="c:\Program Files\Gensym\
  g2-2011\odbc\g2-odbc.exe" parameters="-tcpport 22055
  -tcpipexact"
```

To start the installed service named `myservice`:

```
gservice start myservice
```

To remove the installed service named `myservice`:

```
gservice remove myservice
```

Examples of Using GService with a G2 Process

This example shows how to install a service named `g2-executable` that loads a KB named `myapp.kb` on port `1234`, uses the specified OK file, launches and starts the process with no window, uses the specified log file, and does not start a console. Note the use of double and single quotes around the path names with spaces.

```
gservice install g2-executable program="c:\Program Files\Gensym\
  g2-2011\g2\g2.exe" parameters="-kb 'c:\Program Files\Gensym\
  g2-2011\g2\kbs\demos\myapp.kb' -tcpport 1234
  -ok 'c:\Program Files\Gensym\g2-2011\g2\g2.ok'
  -no-window -no-tray" logfile="c:\Program Files\Gensym\g2-2011\
  g2\g2-service.log" console=false
```

Part X

Appendixes

Appendix A: Launching a G2 Process

Describes techniques, command-line options, and environment variables that can launch and configure the startup and execution of a G2 process.

Appendix B: Reserved Symbols

Explains and lists G2's reserved symbols.

Appendix C: Mouse Gestures, Key Bindings, and Shortcut Keys

Presents the default mouse gestures, key bindings, and shortcut keys.

Appendix D: Syntax Conventions

Describes the notation and user-specified terms used in G2 syntax.

Appendix E: G2 KBs and GIF Files

Describes the demonstration, sample, and utility KBs, and the GIF files that ship with G2.

Appendix F: Superseded Practices

Describes G2 capabilities that are obsolete and may not be supported indefinitely.

A

Launching a G2 Process

Describes techniques, command-line options, and environment variables that can launch and configure the startup and execution of a G2 process.

Introduction

This appendix contains reference information about starting G2 and specifying arguments that configure its startup and execution. This information is essentially the same for all platforms; minor variations between platforms are noted where appropriate.

Some techniques for starting G2 are platform dependent. For details on such techniques, see the `readme-g2` file and the *G2 Bundle Release Notes*. Be sure to consult these documents if you have any trouble starting.

You can use this information to:

- Start G2.
- Identify the operating system environment within which executes.
- Specify how utilizes certain of the computer's resources, such as the initial size of memory pools.
- Configure execution in various other ways.

Note In this appendix, the term Windows refers to any version of Microsoft Windows on which runs.

Starting the G2 Process

You can start a G2 process by entering an appropriate operating system command, using a command window or any other interface available on your platform. The default name of the executable file is:

- Windows: `g2.exe`
- UNIX: `g2`

On Windows platforms, you can start G2 from the Start menu, as well as from a batch file.

See the `readme-g2.html` file for information about platform-dependent techniques for starting G2.

On Windows platforms, you can run G2 as a service. For details, see Chapter 8, "GService" on page 53.

Writing Standard Output Messages to a Log File

As a new process starts up, it creates standard output messages, which describe:

- The initial allocations for memory regions.
- Network port numbers.
- Errors encountered while processing an authorization (OK) file.

A process also displays standard output messages when it must attempt to allocate additional memory and when it detects an internal error. The location of the messages depends on the platform:

- Under UNIX, standard output messages are displayed in the command window from which you launched G2. You can launch G2 with the `-log` command-line option to write standard output messages to the log file you specify. See `log`.
- Under Windows platforms, standard output messages are written to a log file. You have three options:
  - Launch G2 with the `-log` command-line option to write standard output messages to the log file you specify. See `log`.
  - Launch G2 without the `-log` command-line option to create a uniquely named log file in the directory specified by your `TEMP` environment variable.
  - Launch G2 with the `-no-log` command-line option to write standard standard output messages to the command window and not in a log file. See `no-log`.

Writing Network I/O Tracing Messages to a File

You can direct G2 to write network-event trace messages to a specified file. The trace messages concern read, write, connect, accept, and close events to and from G2.

Use the `G2_NETWORK_TRACE_FILE` environment variable to specify the pathname of the file. For example, on a UNIX platform:

```
G2_NETWORK_TRACE_FILE /usr/gensym/g2-2011/g2/io-trace.txt
```

There is no equivalent command-line option.

Using an Initialization File

When you launch a new G2 process, you can cause it to execute a series of commands automatically. You do this by creating an **initialization file** that the new G2 process executes. When you use the `-init` command-line option, the initialization file can have any filename; without the command-line option it must have the name `g2.init`.

Each time you launch a new G2 process, it automatically looks for a file named `g2.init`. If the process finds such a file, it executes the commands in the file sequentially in the order in which they appear.

To use an initialization file, do one of the following:

- Place the file in the default G2 directory, that is, the directory from which the G2 process loads the `g2.ok` file.
- Place the file in a location that you specify to G2 using the `-init` command-line option. See `init`.

If you do not provide an initialization file, the G2 process continues executing without initialization.

Coding an Initialization File

A `g2.init` file is an ASCII file that contains one or more commands. Each command must begin on a new line. Blank lines can appear between the command lines. All texts are case-insensitive, unless they appear inside double-quoted strings.

The most commonly used commands are for loading, which have the same functionality as the corresponding menu choices in the G2 Main Menu: `load kb` and `merge kb`.

In addition, you can start G2 automatically, using `start g2`. If you include the `start g2` command, it should be the final command in the initialization file. The G2 process does not execute any commands that it reads after executing a `start g2` command.

In addition, you can include any command that appears in the Save KB dialog in the classic G2 user interface. These commands include variations on the commands that appear in the Miscellany menu for connecting to and disconnecting from foreign images, writing G2 statistics files, and loading attribute files. They also include commands for saving the KB and saving a module.

If G2 detects an error in any initialization command, it stops executing the rest of the initialization file, and therefore might load only part of the knowledge that your G2 process requires. To avoid this situation, you should thoroughly test the command sequences found in your initialization file.

Note Starting from September 2016 release (G2 2015.9), comments (texts started with a semicolon ";") are supported in G2 initialization files, so that notes and temporary disabled commands can be kept in the same file.

The following table lists the initialization commands and their purposes. In the syntax, *filename* is a path to any file, including a UNC network path on Windows platforms.

Initialization Command	Purpose

`load KB "`*filename*`"`
`  [,{starting afterwards |`
`  not starting afterwards |`
`  warmbooting afterwards |`
`  automatically resolving conflicts |`
`  manually resolving conflicts |`
`  bringing formats up-to-date} ]`
	Load the specified knowledge base.
	For more information, see "Loading a KB" on page 79.

`merge KB "`*filename*`"`	Merge the specified knowledge base into the current KB. For more information, see "Merging a KB File" on page 92.
`start G2`	Start the current KB. This command must appear last in the initialization file.
`shut down G2`	Shut down G2 Server.

`save current KB as`
`  (`*quoted-file-path* `by default)`
`  [{`*overriding-file-name-symbol* `|`
overriding-quoted-file-path`}]`
	Save the current KB to the specified file, by default, or to the specified file name that overrides the default. This command pops up a dialog confirming the save.
	For more information, see "Saving an Unmodularized KB" on page 75.

Initialization Command	Purpose

`save module` *module-name* `as`
`(`*default-quoted-file-path* `by default)`
`[{`*overriding-file-name-symbol* `|`
overriding-quoted-file-path`}]`
`[, including all required modules]`
`[, using clear text]`
	Save the specified module to the specified file, by default, or to the specified file name that overrides the default. This command automatically saves the module without popping up a dialog.
	For more information, see "Saving a Modularized KB" on page 74.

`save module` *module-name* `into`
`(`*default-quoted-directory-path* `by default)`
`[`*overriding-quoted-directory-path*`]`
`[, including all required modules]`
`[, using clear text]`
	Save the specified module to the specified directory, by default, or to the specified directory that overrides the default. The saved file name is the same as the previous loaded one.
	This command automatically saves the module without popping up a dialog.

`save top-level module as`
`(`*default-quoted-file-path* `by default)`
`[{`*overriding-file-name-symbol* `|`
overriding-quoted-file-path`}]`
`[, including all required modules]`
`[, using clear text]`
	Save the current top-level module to the specified file, by default, or to the specified file name that overrides the default.
	This command automatically saves the module without popping up a dialog.

`save top-level module into`
`(`*default-quoted-directory-path* `by default)`
`[`*overriding-quoted-directory-path*`}]`
`[, including all required modules]`
`[, using clear text]`
	Save the current top-level module to the specified directory, by default, or to the specified directory that overrides the default. The saved file name is the same as the previous loaded one.
	This command automatically saves the module without popping up a dialog.

`connect to foreign image "`*filename*`"` `connect to external foreign image at`	Connect to the specified foreign image or external foreign image. For more information, see "Starting and Connecting to the Foreign Image" on page 1626.

Initialization Command	Purpose

`disconnect from foreign image`
`disconnect from all foreign images`
`disconnect from external foreign image at`
	Disconnect from the specified foreign image, all foreign images, or external foreign image.
	For more information, see "Disconnecting from the External Foreign Function" on page 1632.

`write g2 stats as`	Create a statistics file related to memory usage. For more information, see "Correcting Unbounded Memory Requirements" on page 1498.
`load attribute file "`*filename*`"`	Load attribute values for a set of items from the specified attribute file. This is a superseded practice. For more information, see "Attribute Files" on page 1764.

Using Command-Line Options

You can include one or more **command-line options** in the command that starts G2. The syntax is:

```
g2 [option]...
```

In command-line syntax summaries, brackets `[]` indicate optional elements, braces `{}` group elements, a vertical bar `|` separates alternatives, and an ellipsis (. . .) indicates zero or more repetitions of the preceding element.

Command-line options pass various kinds of information to a new process. Regardless of your computer's operating system, each command-line option must begin with the - (hyphen) character, as in this sample command line:

```
g2 -rgn1lmt 7500000
```

The command-line options can appear in any order. Some options require or allow an *argument*: a keyword, pathname, expression, or other value that specifies the exact meaning of the option.

Note When referring to pathnames with spaces on Windows platforms, you must surround the pathname with double quotes, for example, `"c:\Program Files\Gensym\g2-2011\g2\"`.

Supported Command-Line Characters

When you launch from the command line, interprets the command line as a string of 8-bit bytes representing Latin-1 characters. Any two-byte Unicode characters, such as Korean or Japanese characters, therefore become pairs of Latin-1 characters, and the command line is processed accordingly.

Using Environment Variables

Many command-line options have as counterparts a similarly named environment variable. For example, the `-v8ok` command-line option has a counterpart `G2V8_OK` environment variable.

If you start processes many times for similar purposes, setting environment variables once can be more convenient than specifying the same options in each command line. Using G2 command-line options, rather than setting environment variables, might be more appropriate when your G2 process is running a delivered application.

Note If you specify a command-line option when its counterpart environment variable is already set, the command-line option setting takes precedence.

Dictionary of Command-Line Options

The rest of this chapter describes all command-line options in alphabetical order.

The description of each command provides the following sections.

Summary

A brief description of the option, to allow you to determine quickly whether it relates to your needs.

Platforms

The platforms on which the option is available. Most are available on all platforms.

Syntax

The syntax to use when specifying the option on the command line. In command line syntax summaries, brackets `[]` indicate optional elements, braces `{}` group elements, a vertical bar `|` separates alternatives, and an ellipsis (`...`) indicates zero or more repetitions of the preceding element.

Equivalent Environment Variable

You can also specify many options using environment variables. An environment variable automatically applies each time you start, obviating the need to specify a command-line option.

Description

A detailed description of the option and its effects.

Special Considerations

When needed, additional information to keep in mind when using the option.

Example

One or more applications of the option in a typical command line.

background

Changes the gray background of your G2 window to a solid color or to a gray-and-white tiling pattern derived from a GIF or XBM image file you specify.

Platforms

All platforms

Syntax

```
-background { color | file-path | gensym }
```

color: A symbol that represents a supported G2 color.

file-path: A full pathname to a `.gif` or `.xbm` file to use as the background. If the pathname contains spaces, surround it with double quotes.

`gensym`: The G2 "bricks" background, which G2 used as background until Version 8.1.

Equivalent Environment Variable

None

Description

For a list of supported G2 colors, see Identifying the G2 Color Palette.

For information on customizing the background pattern, see Customizing the Gensym Background Pattern"Customizing the Gensym Background Pattern" in Chapter 2 "The Developer's Environment" in the .

The image file must contain no more than 128x128 pixels.

On Windows platforms, you can specify a UNC network path as the *file-path*, such as `\\my-server\my-dir\`.

Examples

```
g2 -background red
```

```
g2 -background
"c:\Program Files\Gensym\g2-2011\tile.gif"
```

cert

Specifies the name of the SSL server certificate to use.

Platforms

All platforms

Syntax

Windows:

`-cert` *name*

name: The Common Name (CN) of the certificate in the local machine's `my` certificate store.

UNIX:

`-cert` *file*

file: The name of the OpenSSL server certificate to use, where *file* is a file containing a private key and a certificate in PEM format, which consists of the DER format base64 encoded with additional header and footer lines.

Equivalent Environment Variable

`G2_CERT`

Description

You specify the certificate when encrypting communication, using the `-secure` command-line option. G2 uses SSPI on Windows and OpenSSL on UNIX.

Example

(Windows) The following command creates a self-signed certificate, suitable for testing, named `test`. `Makecert` is included in the Platform SDK.

```
makecert -r -pe -n "CN=test" -e 01/01/2036 -len 2048
-eku 1.3.6.1.5.5.7.3.1 \

-ss my -sr localMachine -sky exchange \

-sp "Microsoft RSA SChannel Cryptographic Provider" -sy 12
```

cjk-language

Directs the new G2 process to use your specified Han character-style when displaying and printing ideographic, historically Chinese, Han characters.

Platforms

All platforms

Syntax

```
-cjk-language { chinese | japanese | korean }
```

Equivalent Environment Variable

None

Description

For documentation on the G2 Chinese-Japanese-Korean (CJK) language preference, see Specifying a Han Character-Style Preferencee *G2 Reference Manual*.

Example

```
g2 -cjk-language chinese
```

default-language

Specifies the default language of the G2 process.

Platforms

All platforms

Syntax

```
-default-language language-name
```

language-name: Name of a system-defined G2 language facility—`english`, `japanese`, or `korean`—or name of a language—`francais`, `italiano`, or any other symbol—for which a language-translation item in the current KB provides translations of G2 system-defined menu choices.

Equivalent Environment Variable

```
G2_DEFAULT_LANGUAGE
```

Same as the *language-name* argument, described above.

Description

G2 uses its default language when determining the current KB's current language. For a description of how G2 determines the current language for the current KB, see Natural Language Facilities.

Special Considerations

By default, each g2-window item in the current KB, including those that result from a Telewindows connection, can be associated with a distinct language-translation item, independent from the G2 process's default language and from the current KB's current language. For more information, see G2-Windows.

Example

On a Windows platform, this command directs a new G2 process to load the KB file `factory.kb` as its new current KB and to use `francais`, a language-translation item, presumably in `factory.kb`, as the default language:

```
g2 -default-language francais
   -kb "c:\Program Files\g2-2011\g2\kbs\factory.kb"
```

On a UNIX platform, this command directs a new G2 process to load the KB file `factory.kb` as its new current KB and to use `francais`, a language-translation item, presumably in `factory.kb`, as the default language:

```
g2 -default-language francais
   -kb /usr/gensym/g2-2011/g2/kbs/factory.kb
```

display

Directs the new process to display its window on a platform running an X Windows server.

Platforms

UNIX

Syntax

To route the display to a platform via a TCP/IP network connection:

`-display` *machine-name*`:`*server-number*`.`*screen-number*

machine-name: Identifier of a machine running an X Windows server that communicates via a TCP/IP network connection.

server-number: Identifier of an X Windows server process on *machine-name*.

screen-number: Identifier of an X Windows virtual screen managed by X Windows server process *server-number* on *machine-name*.

Equivalent Environment Variable

`DISPLAY`

The `DISPLAY` environment variable for most window managers is usually configured, by default, to be "local host", server `0`, screen `0`. See your X Windows documentation for more information.

Description

This option allows you to display a process's window on a specific machine that runs an X Windows server, specified by its *machine-name*. Note that *machine-name* must already be assigned for each machine connecting to a network using TCP/IP transport protocol.

If you omit *machine-name*, the process assumes the local node is the destination. If you omit *screen-number*, it assumes screen `0`.

For more information see your platform's X Windows reference documentation.

Example

This command directs the new process to display its window on the X Windows server `1`, screen `1` running on machine `mynode`, reached via a TCP/IP network connection:

```
g2 -display mynode:1.1
```

do-not-catch-aborts

When fetal errors happen, do not catch aborts and let Operating System take over.

Platforms

All platforms

Syntax

To stop catching aborts on G2 startup:

```
-do-not-catch-aborts
```

Description

This option will disable all internal error trappings and let Operating System take over when fetal errors happen. This option could be useful if user need to generate core dumps for sending to G2 support team.

exit-on-abort

Casues G2 to exit if a G2 abort occurs.

Platforms

All platforms

Syntax

```
-exit-on-abort
```

Equivalent Environment Variable

None

Description

Exits G2 if an abort occurs.

fonts

Identifies the directory that contains standard font files.

Platforms

All platforms

Syntax

```
-fonts fonts-directory-path
```

fonts-directory-path: Path, with trailing delimiter character, of a directory that contains standard font files.

Equivalent Environment Variable

```
FONTS
```

A directory path, with trailing delimiter character.

Description

G2 includes a set of standard font files. When a new process starts up, it expects by default to find the standard font files in a subdirectory named `fonts` under the home directory, that is, the directory where the executable is installed.

Use the `-fonts` option to direct to look for its standard font files in a custom location.

Special Considerations

The directory delimiter character to use as a trailing character depends on the platform:

- On UNIX platforms, include a trailing / (slash) character.
- On Windows platforms, include a trailing \ (backslash) character.

On Windows platforms, you can specify a UNC network path as the *fonts-directory-path*, such as `\\my-server\my-dir\`.

Example

On a Windows platform, this command starts a process and directs it to finds its standard font files in the directory `c:\fonts\custom-g2\fonts\`:

```
g2 -fonts c:\fonts\custom-g2\fonts\
```

On a UNIX platform, this command starts a process and directs it to finds its standard font files in the directory `/usr/kmm/custom-g2/fonts/`:

```
g2 -fonts /usr/kmm/custom-g2/fonts/
```

fullscreen

Directs the new process to display its window so that it is the same size as the screen where it displays.

Platforms

All platforms

Syntax

```
-fullscreen
```

Equivalent Environment Variable

None

Description

If this option is omitted, the new process opens a window that is, by default, 90% of the size of the screen.

In contrast, on a Windows platform, the `-screenlock` command-line option displays window so that it is the same size as the screen *and* so that it cannot appear behind any other open window. See `screenlock`.

Special Considerations

Use of the `-fullscreen` option prevents viewing the window border and its selectable components that are managed by your platform's window manager that is, X Windows, DECwindows, or Windows.

Example

This command starts a process and displays its window so that its extent is the same size as the screen where it displays:

```
g2 -fullscreen
```

g2passwdexe

Allows you to specify an alternative location for the `g2passwd` program, which you can rename.

Platforms

All platforms

Syntax

`-g2passwdexe` *file-path*

file-path: The directory location for the possibly renamed `g2passwd` file.

Equivalent Environment Variable

None

Description

The `g2passwd` file is used for changing user passwords at secure G2 sites, either from within G2 or from a command console. This command-line option allows you to specify a pathname to that file, which you might have renamed.

On Windows platforms, you can specify a UNC network path as the *file-path*, such as `\\my-server\my-dir\`.

Example

```
g2 -g2passwdexe c:\mydir\mypasswd.exe
g2 -g2passwdexe /mydir/mypasswd.exe
```

geometry

For the window of a new process, specifies the dimensions in pixels and position as an offset in pixels from the upper-left corner of the screen.

Platforms

All platforms

Syntax

`-geometry` *width*x*height*`{+|-}`*x-offset*`{+|-}`*y-offset*

width: Width of the window in pixels.

height: Height of the window in pixels.

+*x-offset*: Pixels from the left of the screen.

−*x-offset*: Pixels from the right of the screen.

+*y-offset*: Pixels from the top of the screen.

−*y-offset*: Pixels from the bottom of the screen.

Equivalent Environment Variable

None

Description

This option specifies the height and width in pixels of your window, which is stored in the g2-window-height and g2-window-width attribute of the g2-window.

On platforms that runs the X Windows window manager, if you omit any values in the geometry string, the process takes the missing values from defaults used by the X Windows resource manager.

On all platforms, you must provide the *width*x*height* argument and the *x-offset* and *y-offset* if you want to specify an offset; otherwise, the offset arguments are ignored.

You can also use negative offsets.

For more information see the X Windows reference documentation for your platform.

Example

This command launches the G2 process with a window that is 800 pixels wide, 600 pixels high, and is offset from the upper-left corner by 20 pixels in each dimension.

```
g2 -geometry 800x600+20+20
```

height

Specifies the height in pixels of the new window.

Platforms

All platforms

Syntax

```
-height number-of-pixels
```

number-of-pixels: A positive integer from 1 to 32,767

Equivalent Environment Variable

None

Description

This option specifies the height in pixels of your window, which is stored in the g2-window-height attribute of the g2-window.

By default, G2 displays a window whose height is 90% of the height of the screen.

On Windows platforms, the height refers to the entire window, including the title bar and the black frame around the window. On UNIX platforms, the height refers to the client window area only; it does not include the height of the title bar and window frame.

Example

This command starts a process with a window whose height in pixels is 1000 and whose width in pixels is equivalent to 90% of the width of the screen, the default:

```
g2 -height 1000
```

help

Directs the new process to output text that shows the syntax of all command-line options.

Platforms

All platforms

Syntax

```
-help
```

Equivalent Environment Variable

None

Description

After issuing this command-line option, the process exits.

G2 writes the help text as follows:

- On UNIX platforms, to the launch window.
- Under Windows, to the console window that appears when G2 is launched, unless the `-log` command-line option is given, in which case the help text is logged to the log file.

Since the console window disappears when G2 exits, you can specify a log file to preserve the help text for future use.

Example

This command directs G2 to output text that shows the syntax of all command-line options, then exit:

```
g2 -help
```

icon

Specifies the text that appears below the icon that appears when you iconize the new window.

Platforms

All platforms

Syntax

`-icon` *icon-text*

icon-text: Text of the name of the icon for the process.

Equivalent Environment Variable

None

Description

When you iconize a window, your platform's window manager hides the window and displays a named icon instead. The `-icon` option specifies the name that appears below that icon.

The text must conform to the requirements for icon names, in terms of the length in characters, the case of alphabetic characters, and so on, established by your platform's window manager.

Special Considerations

To embed a blank in *icon-text* or to specify a mixed case *icon-text* on a platform that does not support commands in mixed-case characters, surround *icon-text* with double quotes, such as:

```
g2 -icon "OpAsst"
```

Example

This command starts a new process and causes the text "OPA" to appear below the icon whenever you minimize the window:

```
g2 -icon OPA -name "Operator's Assistant"
```

This command also causes the text "Operator's Assistant" to appear in the title bar of the local G2 window. See `name`.

init

Directs the new G2 process to use a particular G2 initialization file.

Platforms

All platforms

Syntax

`-init` *init-file-path*

init-file-path: Location and name of a G2 initialization file.

Equivalent Environment Variable

`G2_INIT`

Description

A G2 initialization file is a text file that contains one or more commands. A G2 initialization file can have any name and extension when you use the command-line `-init` option or the environment variable.

If you do not specify the `-init` option, G2 searches for a file named `g2.init` in the directory from which G2 is launched. If G2 finds this file, it loads it as the G2 initialization file.

For more information about the initialization file, see Coding an Initialization File.

When both `-kb` and `-init` options are present on the command line, the KB specified by `-kb` option is loaded first, then rest commands in the `g2.init` file are executed normally. See `kb`.

When both `-start` and `-init` options are present on the command line, the new G2 process also starts the new current KB, after all commands in the `g2.init` file are executed, unless there is a command to shut down G2. See `start`.

On Windows platforms, you can specify a UNC network path as the *init-file-path*, such as `\\my-server\my-dir\`.

Special Considerations

If the file `NOCMD.INIT` is stored in the root directory, that is, in the directory / on UNIX platforms or the directory \ on Window platforms), G2 checks whether the file name specified by *init-file-path* is listed in `NOCMD.INIT`.

If NOCMD.INIT resides in the root directory, G2 loads an initialization file only if its full pathname is listed there.

Example

This command starts a new G2 process on a Windows platform and, after starting up, executes the initialization commands in the file c:\Program Files\Gensym\g2-2011\g2\my-g2.init:

```
g2 -init "c:\Program Files\Gensym\g2-2011\g2\my-g2.init"
```

This command starts a new G2 process on a UNIX platform and, after starting up, executes the initialization commands in the file /usr/gensym/g2-2011/g2/myg2.init:

```
g2 -init /usr/gensym/g2-2011/g2/myg2.init
```

init-string

Passes a text value that assigns into the g2-window-initial-window-configuration-string attribute of the g2-window item that is associated with this window.

Platforms

All platforms

Syntax

```
-init-string init-string-text
```

init-string-text: An unquoted, blank-delimited string of characters.

Equivalent Environment Variable

None

Description

By default, after G2 starts, it creates a new g2-window item. It then assigns the text value specified as the argument to the `-init-string` option to the g2-window-initial-window-configuration-string attribute of the new window.

Before assigning the specified text into the g2-window-initial-window-configuration-string attribute, G2 first normalizes the string as a symbol. That is, it converts all lowercase characters to uppercase, except for any character that follows an @ (at sign) quoting character.

You can provide user-defined command-line arguments when starting G2, using the `-init-string` command-line option, then use the g2-get-command-line-arguments-from-launch system procedure to access those user-defined arguments in G2. The information is stored in the window object inside G2. For example, you can use this argument when displaying Telewindows as an ActiveX control inside of a COM-compliant container. For an example, see the *Telewindows User's Guide*.

Example

This command starts a G2 process and passes the value "Manager" to the g2-window-initial-window-configuration-string attribute of the G2 window:

```
g2 -init-string Manager
```

kb

Directs the new G2 process to load the specified KB file as the new current KB.

Platforms

All platforms

Syntax

`-kb` *kb-file-path*

kb-file-path: Location and name of a KB file.

Equivalent Environment Variable

None

Description

If you include the `-kb` option in a command line that starts G2, after the new G2 process starts up, G2 loads the file specified by *kb-file-path* as its new current KB.

If you do not specify the directory in which the file is stored, G2 looks for the file in the directory from which G2 was launched. If you do not supply a filename extension in *kb-file-path*, G2 assumes that the extension is `.kb`.

If the `-start` option is present on the command line, the new G2 process also starts the new current KB. If the -never-start is the KB will not be started automatically, even if the KB specifies otherwise.

On Windows platforms, you can specify a UNC network path as the *kb-file-path*, such as `\\my-server\my-dir\`.

Special Considerations

If the `-init` option is also specified in the command line, the new G2 process loads the KB specified in the `-kb` option first, and then execute all commands in the `g2.init` file.

If the file `NOCMD.KB` is stored in the root directory (that is, in the directory / on UNIX platforms or the directory \ on Window platforms, G2 checks whether the file name specified by *kb-file-path* is listed in `NOCMD.KB`.

If `NOCMD.KB` resides in the root directory, G2 loads a KB file only if its full pathname is listed there.

Example

This command starts a new G2 process on a Windows platform, which also loads the file `c:\Program Files\Gensym\g2-2011\g2\kbs\factory.kb` as its new current KB:

```
g2 -kb "c:\Program Files\Gensym\g2-2011\g2\kbs\factory.kb"
```

This command starts a new G2 process on a UNIX platform, which also loads the file `/usr/gensym/g2-2011/g2/factory.kb` as its new current KB:

```
g2 -kb /usr/gensym/g2-2011/g2/factory.kb
```

kfepindex, kfepkojin, and kfepmain

Specify custom locations of dictionary files that the new G2 process uses when translating between the Kanji, Hiragana, and Katakana alphabets of the Japanese language.

Platforms

All platforms

Syntax

```
-kfepindex kfepindex-file-path
-kfepkojin kfepkojin-file-path
-kfepmain kfepmain-file-path
```

kfepindex-file-path: Custom location of the `index.dic` file.

kfepkojin-file-path: Custom location of the `kojin.dic` file.

kfepmain-file-path: Custom location of the `main.dic` file.

Equivalent Environment Variable

None

Description

When the new G2 process loads a KB into which the `japanese.kl` file has been merged, and if the files `index.dic`, `kojin.dic`, and `main.dic`, do not reside in their default location, then you must include the `-kfepindex`, `-kfepkojin`, and `-kfepmain` options, respectively. By default, the files `index.dic`, `kojin.dic`, and `main.dic` reside in a subdirectory of the G2 installation directory.

See Natural Language Facilities for information about using the `language.kl` file.

On Windows platforms, you can specify a UNC network path as the *file-path*, such as `\\my-server\my-dir\`.

Special Considerations

If you move the `index.dic`, `kojin.dic`, and `main.dic` files from their default location after installing G2, you must specify the appropriate option to identify the file's new location.

Example

This command starts a new G2 process on a Windows platform and specifies a custom location for the `index.dic` file:

```
g2 -kfepindex c:\lang\index.dic
```

This command starts a new G2 process on a UNIX platform and specifies a custom location for the `index.dic` file:

```
g2 -kfepindex /usr/kmm/lang/index.dic
```

language

Specifies a window-specific G2 natural language facility or language-translation item for the g2-window item that is associated with the window.

Platforms

All platforms

Syntax

`-language` *language-name*

language-name: Name of a standard G2 language facility—english, japanese, korean, or russian—or name of an existing language-translation item—francais, italiano, or any other symbol—in the G2 KB.

Equivalent Environment Variable

None

Description

G2 can specify a window-specific language. Use the `-language` option to specify a standard G2 natural language facility (english, korean, japanese, or russian) or a language-translation item in the G2 KB. This option determines how system-defined menu choices, Text Editor buttons, and so on appear in your G2 window.

Using the `-language` option sets the g2-window-specific-language attribute in the g2-window item of the local window. Specifying this command-line option is equivalent to specifying the G2 Window Specific Language in the login dialog. If you don't specify the `-language` option, the default value of the g2-window-specific-language attribute of this g2-window is none, and the language-translation item used to translate text in your window is determined by the current language of the G2 KB.

When you specify `-language`, the option takes effect only if you also specify a KB with the `-kb` option. Otherwise, the default language of the subsequently loaded KB will override the value of `-language`.

Example

This command starts a secure G2 process, specifying a language and other information necessary to start the process. The command assigns the symbol korean into the g2-window-specific-language attribute of the g2-window:

```
g2 -window viper -user-name howard -password fearnoevil
   -kb secure.kb -user-mode manager -language korean
```

The result is that your window displays the text of G2 system-defined menu choices, and so on, based on the language-translation item named korean found in the G2 KB.

local-window

Explicitly starts G2 with a local window.

Platforms

All platforms

Syntax

```
-local-window
```

Equivalent Environment Variable

None

Description

Currently, this command-line option is redundant, because G2 starts with a local window, by default. However, in a future release, G2 will start without a window, by default. Therefore, we recommend that you explicitly add this option to your startup scripts now if you want G2 to start with a local window, so that in the future, you will not have to change your scripts.

log

Specifies a particular pathname to which standard output messages should be written.

Platforms

All platforms

Syntax

`-log` *log-file-path*

log-file-path: Location and name of a file to which the new process writes standard output messages.

Equivalent Environment Variable

None

Description

If the specified *log-file-path* names an existing file, G2 overwrites the existing file. G2 does not append version information to user-defined log file names.

On Windows platforms, in the absence of the `-log` command-line option, G2 creates a uniquely named log file in the directory specified by your `TEMP` environment variable. The log file name consists of these components:

productname-date-unique-id`.log`

For example, for a log file created on January 1, 2002, which is the fifth log file generated during the G2 session, G2 would use this file name:

```
%PATH%\G2-000101-5.log
```

To avoid writing a log file on Windows platforms, use the `-no-log` command-line option. See no-log. This is the default on UNIX platforms.

On Windows platforms, you can specify a UNC network path as the *log-file-path*, such as `\\my-server\my-dir\`.

Example

This command starts a new process and routes its standard output messages to the file `c:\g2\logs\G2-000410-4.log`:

```
g2 -log c:\g2\logs\G2-000410-4.log
```

magnification

Specifies the new process's default ratio of magnification, that is, pixels per G2 workspace unit, for workspaces displayed at full scale.

Platforms

All platforms

Syntax

`-mag[nification]` *magnification-ratio*

magnification-ratio: A decimal value from 0.50 to 2.66.

Equivalent Environment Variable

None

Description

If you specify neither the `-magnification` option nor the `-x-magnification` or `-y-magnification` options, the new process uses a `-magnification` setting of 1.0. For more information about using the `-magnification`, `-x-magnification`, and `-y-magnification` options, see Specifying the Resolution and Magnification.

Special Considerations

Alternatively, for a display device that supports distinct settings for horizontal and vertical resolutions, you can use the `-x-magnification` and `-y-magnification` command-line options to specify distinct horizontal and vertical magnifications. However, don't combine the `-magnification` argument with either the `-x-magnification` or `-y-magnification` options. See `x-magnification and y-magnification`.

Example

This command starts a process that displays workspaces at full scale at the magnification of two pixels per G2 workspace unit:

```
g2 -magnification 2.0
```

manually-resolving-conflicts

Loads the specified KB without automatically resolving conflicts.

Platforms

All platforms

Syntax

```
-manually-resolving-conflicts
```

Equivalent Environment Variable

None

Description

Beginning with G2 Version 6.0, when you load a KB from the command line with the `-kb` option, G2 automatically resolves conflicts, as if you had enabled the automatically resolve conflicts option in the Load KB dialog. Prior to G2 Version 6.0, loading a KB with the `-kb` option did not automatically resolve conflicts.

If there are conflicts in your KB, you might want a chance resolve those conflicts manually, just as you can when you load a KB from within G2. G2 provides this command-line option, as well as a command for the `g2.init` file, to allow you to load a KB without automatically resolving conflicts. If this option is not specified, any conflicts will be resolved automatically.

You must use this command-line option with the `-kb` command-line option to specify the KB file to load.

For general information on resolving conflicting class-definitions, see:

- "Detecting Conflicting Class-Definitions" on page 96.
- "Automatically Resolving Conflicting Class-Definitions" on page 96.
- "Manually Resolving Conflicting Class-Definitions" on page 98.

For information on the equivalent command to use in the `g2.init` file, see Using an Initialization File.

Example

This command starts a new G2 process on a Windows platform and loads the specified KB, without resolving conflicts:

```
g2 -kb "c:\Program Files\Gensym\g2-2011\g2\kbs\factory.kb"
-manually-resolving-conflicts
```

This command starts a new G2 process on a UNIX platform and loads the specified KB, without resolving conflicts:

```
g2 -kb /usr/gensym/g2-2011/g2/factory.kb
-manually-resolving-conflicts
```

module-map

Specifies the location of the optional module-map file.

Platforms

All platforms

Syntax

`-module-map` *module-map-file-path*

module-map-file-path: Location and name of the module-map file.

Equivalent Environment Variable

`G2_MODULE_MAP`

Same syntax as *module-map-file-path*.

Description

G2 uses the optional module-map file when loading and saving modular KBs that contain directly required modules. Each record in a module-map file associates a module name with either a fully qualified directory pathname or a fully qualified file pathname.

If no `-module-map` option is specified, and if the `G2_MODULE_MAP` environment variable is not set, G2 looks for a file named `g2.mm` in the directory from which you launched G2.

For more information about using a module-map file and the `-module-map` option, see Using a Module Map File to Load and Save a KB.

On Windows platforms, you can specify a UNC network path as the *module-map-file-path*, such as `\\my-server\my-dir\`.

Example

This command starts a new G2 process on Windows and directs G2 to use the module-map file named `c:\Program Files\Gensym\g2-2011\g2\modmap.txt`:

```
g2 -module-map "c:\Program Files\Gensym\g2-2011\g2\modmap.txt"
```

This command starts a new G2 process on UNIX and directs G2 to use the module-map file named `/usr/gensym/g2-2011/g2/modmap.txt`:

```
g2 -module-map /usr/gensym/g2-2011/g2/modmap.txt
```

module-search-path

Specifies the search path for locating modular KB files.

Platforms

All platforms

Syntax

`-module-search-path` *search-dir-path*

`-module_search_path` *search-dir-path*

or

`-module-search-path "`*search-dir-path* `[`*search-dir-path*`] ... "`

`-module_search_path "`*search-dir-path* `[`*search-dir-path*`] ... "`

search-dir-path: A fully qualified directory path that names a location where G2 searches for a modular KB file that contains a directly required module.

Equivalent Environment Variable

`G2_MODULE_SEARCH_PATH`

The value of the environment variable is limited to 1023 characters. Under UNIX and VMS, the syntax is the same as for the command-line option.

Description

This option specifies a list of one or more directories that G2 searches to locate KB files that contain directly required modules. G2 searches this list of directories in the order in which they appear if it cannot find a directly required module's KB file in the directory that contains the top-level module's KB file.

Previous versions of G2 required that the KB file for a top-level module and the KB files for its directly required modules reside in the same directory. By default, G2 still searches in this manner for the KB files of directly required modules.

If you specify more than one directory, enclose the list of directories in double quotes. Use a space character to separate the directory paths in the list.

On Windows platforms, if the pathname includes spaces, you must surround it with single quotes (`' '`).

On Windows platforms, you can specify a UNC network path as the *search-dir-path*, such as `\\my-server\my-dir\`.

For more information about using a module search path and the `-module-search-path` option, as well as examples, see Using a Module Search Path to Load KB Files.

Example

This command starts a new G2 process on Windows and declares two directory paths in the module search path, where one path includes a space and is, therefore surrounded by single quotes:

```
g2 -module-search-path "\kbs\current-release\
   'c:\Program Files\Gensym\g2-2011\g2\kbs\'"
```

This command starts a new G2 process on UNIX and declares two directory paths in the module search path:

```
g2 -module-search-path "/kbs/current-releases/
   /usr/gensym/g2-2011/g2/kbs/"
```

name

Specifies the text that appears in the title bar of the new G2 window.

Platforms

All platforms

Syntax

`-name` *window-title-text*

window-title-text: A string of characters; must conform to the requirements of your platform's window manager for title bar text.

Equivalent Environment Variable

None

Description

The location, appearance, allowable characters, and allowable length of the title text are determined by the requirements of your platform's window manager.

Special Considerations

To use this option with Japanese, use 8-bit characters; 16-bit characters do not display correctly.

To embed a blank in *window-title-text* or to specify a mixed case *window-title-text* on a platform that does not support commands in mixed case characters, surround *window-title-text* with double quotes, such as:

```
g2 -name "Operator's Assistant"
```

Example

This command starts a new process and specifies the text "Operator's Assistant" to appear in the title bar of its window:

```
g2 -name "Operator's Assistant" -icon OPA
```

This command also causes the text "OPA" to appear below the process's icon, when it is minimized.

netinfo

Specifies that the host name and port number should appear in the title bar of the G2 window.

Platforms

All platforms

Syntax

```
-netinfo
```

Equivalent Environment Variable

None

Description

This feature allows you to immediately see the host and port to which to connect via Telewindows. The format of the title bar text is:

G2 - [*host:port*]

If you start G2 with the `-name` command-line option, the specified name replaces G2 in the title bar.

When launching the G2 server from the Start menu on Windows platforms, the network information appears in the title bar, by default. As a result, the network information also appears in the G2 icon in the Windows task bar.

Example

This command starts a new process with the network information in the title bar:

```
g2 -netinfo
```

network

Specifies the network transport protocol on which G2 should listen for incoming network connections.

Platforms

All platforms

Syntax

```
-network { tcpip }
```

`tcpip`: Listen for network connections that use the TCP/IP transport protocol.

Equivalent Environment Variable

None

Description

When a new G2 process is running on a machine that connects to a network via multiple transport protocols, use the `-network` option to direct G2 to listen for incoming network connections using a particular transport protocol. Currently, the only supported network protocolo is TPC/IP. Thus, this command-line option is reserved for future use.

By default, G2 listens on all the transports that it can.

If you supply more than one `-network` option on the command line, G2 listens on all the specified network transports.

However, G2 will *not* exit, warn, or otherwise signal an error if there is a transport on which it cannot listen. G2 *will* exit if it is unable to use the network to the minimal extent necessary to begin its per-process license census.

Example

This command starts a new G2 process and directs it to listen for incoming network connections only via the TCP/IP transport protocol:

```
g2 -network tcpip
```

never-start

Directs the new G2 process to not start the KB identified in the accompanying `-kb` option.

Platforms

All platforms

Syntax

```
-never-start
```

Equivalent Environment Variable

None

Description

When used with the `-kb` option, setting of the start-kb-after-load? attribute of the Miscellaneous Parameters system table in the KB is overridden. If the `-kb` option is not included with the `-never-start` option in the command line, G2 ignores the `-never-start` option.

Example

This command starts a new G2 process on a Windows platform, which loads the KB stored in the file `c:\Program Files\Gensym\g2-2011\g2\kbs\demos\axldemo.kb` and inhibits auto-start, specified in its system-tables:

```
g2 -never-start -kb "c:\Program Files\gensym\g2-
2011\g2\kbs\demos\axldemo.kb"
```

This command starts a new G2 process on a UNIX platform, which loads the KB stored in the file `/usr/gensym/g2-2011/g2/kbs/demos/axldemo.kb` and and inhibits auto-start, specified in its system-tables:

```
g2 -never-start -kb /usr/gensym/g2-2011/g2/kbs/demos/axldemo.kb
```

no-backing-store

Disables the use of the default backing-store facility.

Platforms

UNIX

Syntax

```
-no-backing-store
```

Equivalent Environment Variable

None

Description

Using an X-Server Backing-Store

By default, with backing-store in effect, the X-server caches a window image in its own memory each time a window is obscured or iconized. Then, whenever a window, or part thereof, is redrawn, the X-server simply redraws the window from memory, rather than requesting an update from the G2 server.

The advantage of this facility is for Telewindows users, especially those connecting to G2 across a slow network. Using this default option precludes the need for a Telewindows client to make G2 redraw requests each time a window, or any portion of a window, must be redrawn after being iconized or obscured in any way. Caching the window on the X-server in such an environment can then significantly improve window redraw performance.

Two disadvantages exist when using the backing-store option:

- The X-server may require more memory.
- If the server stops responding, redrawn windows may not be current.

X-Server Memory Requirements

The X-server can potentially require more memory when using the backing-store feature. The amount of memory required depends on:

- The size of the window being cached
- The number of colors in use

If the X-server has sufficient memory when you start G2 or Telewindows, it uses what is available to cache the window. If more memory is required for backing-

store, the X-server allocates whatever it needs. Additionally allocated memory for backing-store then remains in use for the duration of the G2 or Telewindows process.

Updating Cached Windows

Using backing-store does not affect regular window updates from the G2 server. For example, G2 continually updates display items, such as readout-tables and trend-charts, even if a window is obscured. Thus, when the X-server redraws a window containing display items, its data is always current.

An obscured or iconized window can potentially become out of date if the G2 server stops responding to the client. In this case, with backing-store in effect, the X-server redraws the window from its previous state, even if that state is no longer current. Conversely, if backing-store is *not* in use (G2 was launched with the `-no-backing-store` command-line option), attempting to redraw an obscured window results in an entirely blank window that remains until the G2 server responds once more.

Example

This command starts a new process and disables the use of the backing-store facility:

```
g2 -no-backing-store
```

no-log

Specifies that Windows should not write a log file for the G2 process.

Platforms

Windows

Syntax

```
-no-log
```

Equivalent Environment Variable

`G2_NO_LOG`

Description

By default, if you launch G2 without the `-log` command-line option, Windows creates a uniquely named log file in the `TEMP` directory, as described in Writing Standard Output Messages to a Log File.

The only way to avoid generating a log file is to use the `-no-log` option.

G2 accepts the `G2_NO_LOG` environment variable, which causes G2 not to generate a log file, as if you had started G2 with the `-no-log` command-line option. To use the variable, set it to a value of `1`.

Example

This command starts a new G2 process with no log file:

```
g2 -no-log
```

no-tray

Starts a G2 process with no icon in the system tray.

Platforms

Windows

Syntax

```
-no-tray
```

Equivalent Environment Variable

None

Description

When running G2 as a service, you must start G2 with the `-no-tray` command-line option to suppress the icon; otherwise, an error occurs when you start G2 as a service.

Example

This command starts a new G2 process with no icon in the system tray.

```
g2 -no-tray
```

no-window

Starts a G2 process with no visible window.

Platforms

All platforms

Syntax

```
-no-window
```

Equivalent Environment Variable

None

Description

Use the `-no-window` option to run G2 where no visible window is needed. Typically, you use Telewindows to connect to G2 on such a machine.

Note When you launch G2 from a Microsoft Windows icon, G2 needs somewhere to put standard output messages, and will expose a console window to hold them unless `-log` was also specified in the command line. Specifying `-log` allows G2 to redirect the standard output messages to a log file, avoiding the need for a console window. No windows then appear when G2 starts.

Example

This command starts a new G2 process that has no visible window:

```
g2 -no-window
```

ok

Specifies the location of the G2 authorization file.

Platforms

All platforms

Syntax

`-ok` *ok-file-path*

ok-file-path: Location and name of the authorization file; you can specify a file with any name and location.

Equivalent Environment Variable

`G2_OK`

Same as *ok-file-path*.

Description

The default name of the authorization file is `g2.ok`.

If the file `NOCMD.OK` is stored in the root directory, G2 checks whether the authorization file name specified by *ok-file-path* is listed in `NOCMD.OK`. G2 loads the file only if it is listed there. The root directory is the directory / on UNIX platforms or the directory \ on Windows platforms.

If you omit the `-ok` option, the new process looks for the authorization file in the directory from which you launched G2.

On Windows platforms, you can specify a UNC network path as the *ok-file-path*, such as `\\my-server\my-dir\`.

Special Considerations

You can also use the `-v11ok` command-line option, the `G2V11_OK` environment variable, and the `g2v11.ok` file to specify the location of the authorization file. The order of precedence for identifying the authorization file to use is:

1. The `-v11ok` command-line option.
2. The `-ok` command line arg.
3. The `G2V11_OK` environment variable.
4. The `G2_OK` environment variable.

5 A file named `g2v11.ok` in the G2 home directory.

6 A file named `g2.ok` in the G2 home directory.

For more information, see v11ok.

Example

On a Windows platform, this command starts a new process and identifies `c:\Program Files\Gensym\g2-2011\g2\my.ok` as the authorization file:

```
g2 -ok "c:\Program Files\Gensym\g2-2011\g2\my.ok"
```

On a UNIX platform, this command starts a new process and identifies `/usr/gensym/g2-2011/g2/my.ok` as the authorization file:

```
g2 -ok /usr/gensym/g2-2011/g2/my.ok
```

password

Specifies the password for starting a secure G2 process.

Platforms

All platforms

Syntax

`-password` *password-string*

password-string: A series of characters that constitute a password.

Equivalent Environment Variable

None

Description

Specifying the `-password` option corresponds to filling in the Password field in the login dialog.

Example

This command starts a secure G2 process, specifying a password and other information necessary to start the process.

```
g2 -window viper -user-name howard -password fearnoevil
   -kb secure.kb -user-mode manager -language korean
```

private-colormap

Causes G2 to request a dedicated colormap from the X server. The default is to use the standard colormap, which is shared by all applications.

Platforms

UNIX

Syntax

```
-private-colormap
```

Equivalent Environment Variable

None

Description

Although most color X workstations are capable of displaying many more than 256 colors, most X servers can represent only 256 colors at a time. What those colors are depends on the applications that access the X server and sometimes on the order in which they are run.

When an application wants to display a pixel in a certain color, it asks the X server to map the color to an index in the colormap. If the color is not already present in the colormap, the X server adds it, using up one more slot in the colormap. If this continues, the colormap becomes full. When an application asks for a new mapping and the colormap is full, the X server responds by using the closest available color to the one requested. The result may not be an acceptable color.

Applications, such as Web browsers, which display images (GIFs, JPEGs, and so on) typically need many colors, increasing the chance that the colormap will become full. For example, if you run Netscape on your UNIX machine before running G2, there is a chance that G2 will not be able to find space in the colormap for even its 64 basic colors, and the display will look wrong.

The alternative is to use a private colormap. When a private colormap is in use, no other applications can fill up the slots. In the case of G2, this guarantees that the 64 basic colors will always be available.

An additional benefit to using a private colormap is that more color cells will be available should G2 need them. In fact, G2 does need them to display GIFs well. If you change the image-palette attribute in the Color Parameters system table to Custom colors from *my-image-def*, G2 displays the GIF rendered better than it has been able to before under X.

Special Considerations

The disadvantage of using a private colormap is that the X server has to actively switch between color maps. It does this based on focus. If G2 is using a private colormap and some other application has the focus, the colors in G2 will appear obviously wrong. When G2 is given focus, the X server installs its colormap, and the G2 window looks fine. However, the colors of all other applications, as well as icons and other items drawn by the X server and the window manager, will be wrong. They are still using the same indices into the colormap as they were before, but those indices no longer make sense because a different colormap is in use.

Since users may find this color change problem disconcerting, the `-private-colormap` flag is off by default.

Note This option will be ignored if you are using 24-bit display because the colormap would be too large.

Example

This command starts a new process and requests a private colormap from the X server:

```
g2 -private-colormap
```

regserver

Registers Telewindows.

Platforms

Windows

Syntax

```
-regserver
```

Equivalent Environment Variable

None

Description

When you install the G2 Bundle, Telewindows automatically calls `-regserver` to register both Telewindows Next Generation (`twng.exe`) and Telewindows (`tw.exe`).

Telewindows looks in the registry when connecting to G2, as follows:

When connecting...	Telewindows connects...
Using the Connect Telewindows menu choice on the G2 server icon	Telewindows Next Generation, if it can find it in the registry; otherwise, Telewindows
Telewindows ActiveX control	Telewindows
WorkspaceView ActiveX control	Telewindows Next Generation

To use Telewindows instead of Telewindows Next Generation when using the Connect Telewindows menu choice, unregister Telewindows Next Generation. See `unregserver`.

The command-line option supports an optional argument, `-s`, to suppress the dialog that appears when you successfully register Telewindows.

Note You can specify the command-line option by using `-regserver` and `-s`, or you can use the Windows style command-line options, `/regserver` and `/s`.

Telewindows writes its location is these Windows registry keys, depending on the executable:

Executable	Registry Key
`twng.exe`	`installDir`
`tw.exe`	`installDirClassic`

The location in the registry for both keys is:

`HKEY_LOCAL_MACHINE\SOFTWARE\Gensym\Telewindows\`*version*

where *version* is the Telewindows version and revision, including `Beta`, if appropriate. For example, the location for Telewindows Version 2011 Rev. 0 is:

`HKEY_LOCAL_MACHINE\SOFTWARE\Gensym\Telewindows\8.3 Rev. 0`

resolution

Specifies the resolution of the monitor on which the G2 window appears.

Platforms

All platforms

Syntax

```
-resolution dots-per-inch
```

dots-per-inch: An integer in the range 36 to 200.

Equivalent Environment Variable

None

Description

This option specifies the resolution, in pixels per inch, at which the new process displays its local window and any Telewindows to which it is connected. Use this option to adjust the absolute size of window to the resolution characteristics of your display device. For more information see Specifying the Resolution and Magnification.

By default, displays a window at 75 dots per inch (dpi).

Special Considerations

For a display device that supports distinct settings for its vertical (y-axis) resolution and horizontal (x-axis) resolution, you can specify separate default vertical and horizontal resolutions, as follows:

```
g2 -x-resolution 75 -y-resolution 100
```

Do not combine the `-resolution` option with either the `-x-resolution` or `-y-resolution` options. If you specify the `-x-resolution` option, you should also specify the `-y-resolution` option, and vice versa. See `x-resolution and y-resolution`.

Example

This command starts a new G2 process that displays its window at a resolution of 150 dots per inch:

```
g2 -resolution 150
```

rgn1lmt

Specifies the initial supply of available memory for data other than symbols and graphics images.

Platforms

All platforms

Syntax

`-rgn1lmt` *number-of-bytes*

number-of-bytes: The integer 4750000 or higher, up to the maximum per-process allocation determined by your platform's operating system settings.

Note Do not include commas when specifying *number-of-bytes*.

Equivalent Environment Variable

`G2RGN1LMT`

Same syntax as *number-of-bytes*.

Description

G2 maintains its supply of available memory in three regions. This option controls the initial supply of available memory in its Region 1 memory pool. G2 uses its Region 1 memory pool to store all data other than symbols and graphics images.

The new process allocates Region 1 memory when it is launched. G2 standard output messages at startup indicate the memory allocation.

The default amount of Region 1 memory is 10,000,000 bytes.

Special Considerations

If your `-rgn1lmt` option specifies less than the minimum number of bytes, G2 displays a warning standard output message and supplies the minimum amount.

For information about G2 memory management, see Memory Management. For details about Region 1 memory allocation, see in that chapter:

- G2 Memory Regions.
- Determining Region 1 and Region 2 Memory Requirements.
- Specifying G2 Memory Allocation.

Example

This command starts a new process and directs it to allocate 8,500,000 bytes as its initial supply of Region 1 memory:

```
g2 -rgn1lmt 8500000
```

G2 attempts to allocate more Region 1 memory as is required by the current KB's processing.

rgn2lmt

Specifies the initial supply of available memory for symbol data.

Platforms

All platforms

Syntax

`-rgn2lmt` *number-of-bytes*

number-of-bytes: The integer 3000000 or higher, up to the maximum per-process allocation determined by your platform's operating system settings.

Note Do not include commas when specifying *number-of-bytes*.

Equivalent Environment Variable

`G2RGN2LMT`

Same syntax as *number-of-bytes*.

Description

G2 maintains its supply of available memory in three regions. This option controls the initial supply of available memory in its Region 2 memory pool. G2 uses its Region 2 memory pool to store symbols and related internal data.

The new process allocates Region 2 memory when it is launched. G2 standard output messages at startup indicate the memory allocation.

The default amount of Region 2 memory is 3,000,000 bytes.

Special Considerations

If your `-rgn2lmt` option specifies less than the minimum number of bytes, G2 displays a warning standard output message and G2 supplies the minimum amount.

For information about G2 memory management, see Memory Management. For details about Region 2 memory allocation, see in that chapter:

- G2 Memory Regions.
- Determining Region 1 and Region 2 Memory Requirements.
- Specifying G2 Memory Allocation.

Example

This command starts a new process and directs it to allocate 4,500,000 bytes as its initial supply of Region 2 memory:

```
g2 -rgn2lmt 4500000
```

G2 attempts to allocate more Region 2 memory as is required by the current KB'sprocessing.

rgn3lmt

Specifies the *maximum* memory used by to render the background images of workspaces.

Platforms

All platforms

Syntax

`-rgn3lmt` *number-of-bytes*

number-of-bytes: The integer 400000 or higher, up to the number of bytes of memory actually available to the target platform's window manager.

Note Do not include commas when specifying *number-of-bytes*.

Equivalent Environment Variable

`G2RGN3LMT`

Same syntax as *number-of-bytes*.

Description

G2 maintains its supply of available memory in three regions. This option controls the *maximum* memory that a new process can allocate for its Region 3 memory pool. G2 uses its Region 3 memory pool to manage the graphics image data that displays as the background images of workspaces and icons.

The new process does *not* allocate Region 3 memory when it is launched. Instead, G2 allocates Region 3 memory at the time that the KB must display a background image.

The default maximum amount of Region 3 memory is 2,500,000 bytes. If you specify a *number-of-bytes* larger than 2,500,000 bytes, the new process can manage more precomputed image renderings, which, in turn, means that background images can display more quickly.

Special Considerations

If your `-rgn3lmt` option specifies less than the allowable minimum (400,000), G2 displays a warning standard output message and substitutes the minimum specification.

For information about G2 memory management, see Memory Management. For details about Region 3 memory allocation, see in that chapter:

- G2 Memory Regions.
- Restricting Region 3 Memory.
- Specifying G2 Memory Allocation.

Example

This command starts a new process and restricts it to allocating at most 500,000 bytes of Region 3 memory:

```
g2 -rgn3lmt 500000
```

G2 attempts to allocate Region 3 memory as is required by the current KB's processing.

screenlock

Displays the new window so that it occupies the entire screen.

Platforms

Windows

Syntax

```
-screenlock
```

Equivalent Environment Variable

None

Description

On Windows platforms, this option displays the new window at the top of the window hierarchy and prevents any other application window from being on top.

In contrast, on any supported platform, the `-fullscreen` command-line option displays the window so that it is the same size as the screen *and* so that the window can appear behind any other open window.

Example

This command starts a new process and directs it to display its window so that its extent occupies the entire screen:

```
g2 -screenlock
```

After executing this command, the user also cannot cause any other application's window to appear on top of the window that displays.

secure

Encrypts communication.

Platforms

All platforms

Syntax

```
-secure
```

Equivalent Environment Variable

None

Description

This command-line option causes G2 and Telewindows to use SSL on all TCP/ICP connections. G2 uses SSPI on Windows and OpenSSL on UNIX.

When the connection is encrypted, the padlock icon appears in the status bar.

start

Directs the new G2 process to start the KB identified in the accompanying `-kb` option.

Platforms

All platforms

Syntax

```
-start
```

Equivalent Environment Variable

G2_START

Description

When used with the `-kb` option, G2 starts the specified KB. If the `-kb` option is not included with the `-start` option in the command line, G2 ignores the `-start` option.

Specifying the `-start` option overrides the setting of the start-kb-after-load? attribute of the Miscellaneous Parameters system table in the KB specified as the argument to the accompanying `-kb` option.

Special Considerations

If the `-init` option is also specified in the command line, the new G2 process executes all commands in the `g2.init` file first, then starts the newly loaded KB, if it's not started yet by any command in the `g2.init` file, unless there is a command to shut down G2.

Example

This command starts a new G2 process on a Windows platform, which loads and begins running the KB stored in the file `c:\Program Files\Gensym\g2-2011\g2\kbs\my-kb.kb`:

```
g2 -start -kb "c:\Program Files\gensym\g2-2011\kbs\my-kb.kb"
```

This command starts a new G2 process on a UNIX platform, which loads and begins running the KB stored in the file `/usr/gensym/g2-2011/g2/kbs/my-kb.kb`:

```
g2 -start -kb /usr/gensym/g2-2011/g2/kbs/my-kb.kb
```

tcpipexact

Prohibits the new G2 process from attempting to open a network connection to any TCP/IP port other than that specified in the accompanying `-tcpport` command-line option.

Platforms

All platforms

Syntax

```
-tcpipexact
```

Equivalent Environment Variable

None

Description

This option directs the new G2 process to exit before completing its startup, if it cannot open a network connection to the TCP/IP port specified in the accompanying `-tcpport` command-line option.

This option requires that you also include the `-tcpport` option in the command line that launches a G2 process. G2 ignores this option unless the command line also includes the `-tcpport` option.

Example

This command starts a G2 process that attempts to open a network connection to the TCP/IP port `1711`:

```
g2 -tcpport 1711 -tcpipexact
```

If this attempt is not successful, the G2 process does *not* attempt to open a network connection to another TCP/IP port, and automatically exits.

tcpport

Directs the new G2 process to open a network connection to the specified TCP/IP port, with additional attempts to connect to other TCP/IP ports as necessary.

Platforms

All platforms

Syntax

`-tcpport` *tcpip-port-number*

tcpip-port-number: A positive integer; however, TCP/IP port numbers under 1000 are often reserved by your platform and should be avoided for use with G2.

Equivalent Environment Variable

None

Description

This option directs the new G2 process to open a network connection to a TCP/IP port. Specify the port's name as an argument, following the option.

If the new G2 process cannot open a connection to the specified port, this option also directs G2 to attempt to open a connection to the G2 default TCP/IP port `1111`. If this is not successful, G2 increments the port's last digit (to `1112`), attempts to connect to that port, and so on. G2 stops after trying to connect to TCP/IP port `1210`, that is, after incrementing the G2 default TCP/IP port number 100 times.

Special Considerations

Including the `-tcpipexact` option in the command line modifies how the new G2 process attempts to open a TCP/IP network connection. See the section `tcpipexact`.

The number of sockets available to G2 at run time is determined by various operating system specified limits. These limits can vary between operating system releases and as such it is difficult to predict exactly how many network connections a given G2 can service. Further complicating this situation is that in many operating systems, descriptors for files and sockets are derived from a shared resource. That is to say, the maximum number of sockets can be decreased for every file opened.

The following table lists the theoretical maximum number of outstanding sockets available to G2. In practice, you will never be able to manage exactly this many sockets. Before deploying an application which may approach these upper limits of socket usage, you should test your operating system's ability to safely accommodate G2's resource needs.

Sockets Available to G2

Operating System	Maximum Number of Sockets
Microsoft Windows	64
RedHat Linux	1024
Sun Solaris	1024
IBM AIX	2048
HP HP-UX	1024
Compaq Tru64 Unix	4096

Example

This command starts a G2 process that attempts to open a network connection to TCP/IP port 1711:

```
g2 -tcpport 1711
```

ui

Specifies the user interface style when starting G2.

Platforms

All platforms

Syntax

```
-ui standard | classic
```

Equivalent Environment Variable

None

Description

Both options provide a standard user interface, where “standard” implies the Windows standard, which includes standard selection-style, mouse gestures, keystrokes, and key bindings. On UNIX platforms, you get as many native features as the platform supports.

For backward compatibility, you can also run G2with its classic user interface, where “classic” implies G2 6.x behavior. The classic user interface uses a single document interface (SDI), and classic G2 selection-style, menus, and mouse gestures, keystrokes, and key bindings.

Examples

To start G2 with its default user interface, use this command:

```
g2
```

Starting G2 with no command line options is equivalent to:

```
g2 -ui standard
```

To start G2 with its classic user interface:

```
g2 -ui classic
```

unregserver

Unregisters Telewindows.

Platforms

Windows

Syntax

```
-unregserver
```

Equivalent Environment Variable

None

Description

When you install the G2 Bundle, Telewindows automatically calls `-regserver` to register both Telewindows Next Generation (`twng.exe`) and Telewindows (`tw.exe`). See regserver.

Telewindows looks in the registry when connecting to G2, as follows:

When connecting...	Telewindows connects...
Using the Connect Telewindows menu choice on the G2 server icon	Telewindows Next Generation, if it can find it in the registry; otherwise, Telewindows
Telewindows ActiveX control	Telewindows
WorkspaceView ActiveX control	Telewindows Next Generation

To use Telewindows instead of Telewindows Next Generation when using the Connect Telewindows menu choice, unregister Telewindows Next Generation.

The command-line option supports an optional argument, `-s`, to suppress the dialog that appears when you successfully unregister Telewindows.

Note You can specify the command-line option by using `-unregserver` and `-s`, or you can use the Windows style command-line options, `/unregserver` and `/s`.

Telewindows writes its location is these Windows registry keys, depending on the executable:

Executable	Registry Key
`twng.exe`	`installDir`
`tw.exe`	`installDirClassic`

The location in the registry for both keys is:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Gensym\Telewindows\version
```

where *version* is the Telewindows version and revision, including `Beta`, if appropriate. For example, the location for Telewindows Version 2011 Rev. 0 is:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Gensym\Telewindows\8.3 Rev. 0
```

user-mode

Specifies the user mode for starting a secure G2 process.

Platforms

All platforms

Syntax

`-user-mode` *user-mode-string*

user-mode-string: Series of characters that names a user mode.

Equivalent Environment Variable

None

Description

Specifying the `-user-mode` option corresponds to filling in the User Mode field in the login dialog and sets the g2-user-mode attribute of the current g2-window.

When you specify `-user-mode`, the option takes effect only if you also specify a KB with the `-kb` option. Otherwise, the default user mode of the subsequently loaded KB will override the value of `-user-mode`.

Example

This command starts a secure G2 process, specifying a user mode and other information necessary to start the process.

```
g2 -window viper -user-name howard -password fearnoevil
   -kb secure.kb -user-mode manager -language korean
```

user-name

Specifies the user name under which to start a secure G2 process.

Platforms

All platforms

Syntax

`-user-name` *user-name-string*

user-name-string: A user name defined in the authorization file for a secure G2.

Equivalent Environment Variable

None

Description

Specifying the `-user-name` option corresponds to filling in the User Name field in the login dialog and sets the g2-user-name attribute of the current g2-window.

Example

This command starts a secure G2 process, specifying a user name and other information necessary to start the process.

```
g2 -window viper -user-name howard -password fearnoevil
   -kb secure.kb -user-mode manager -language korean
```

v11ok

Specifies a custom location for an authorization file that is specific to G2 Version 2011.

Platforms

All platforms

Syntax

`-v11ok` *v11-ok-file-path*

v11-ok-file-path: Location and name of the authorization file for Version 2011; you can specify a file with any name and location.

Equivalent Environment Variable

`G2V11_OK`

Same syntax as *v11-ok-file-path*.

Description

Like the `-ok` option, the `-v11ok` option specifies the path of an authorization file for this process.

Specifying the `-v11ok` option allows the site manager to configure a system environment so that processes launched using a previous version authorize using an authorization file for that previous version, while processes launched using Version 2011 authorize using a separate authorization file.

If the file `NOCMD.OK` is stored in the root directory, G2 checks whether the authorization file name specified as *v11-ok-file-path* is listed in `NOCMD.OK`. G2 loads the file only if it is listed there. The root directory is the directory / on UNIX platforms or the directory \ on Windows platforms.)

On Windows platforms, you can specify a UNC network path as the *v11-ok-file-path*, such as `\\my-server\my-dir\`.

Special Considerations

You can also use the `-ok` command-line option, the `G2_OK` environment variable, and the `g2.ok` file to specify the location of the authorization file. The order of precedence for identifying the authorization file to use is:

1. The `-v11ok` command-line option.
2. The `-ok` command line arg.
3. The `G2V11_OK` environment variable.
4. The `G2_OK` environment variable.
5. A file named `g2v11.ok` in the G2 home directory.
6. A file named `g2.ok` in the G2 home directory.

For more information, see ok.

Example

On a Windows platform, this command starts a new process and identifies `c:\Program Files\Gensym\g2-2011\g2\my-g2.ok` as the Version 2011 authorization file:

```
g2 -v11ok "c:\Program Files\Gensym\g2-2011\g2\my-g2.ok"
```

On a UNIX platform, this command starts a new process and identifies `/usr/gensym/g2-2011/g2/my.ok` as the Version 2011 authorization file:

```
g2 -v11ok /usr/gensym/g2-2011/g2/my-g2.ok
```

verbose

Prints information about the current G2 to the console or log file, such as the location of the G2 OK file.

Platforms

All platforms

Syntax

```
-verbose
```

Equivalent Environment Variable

None

Description

Example

```
g2 -verbose
```

For example:

```
2004-09-30 10:00:28
2004-09-30 10:00:28   [Using G2 OK file: "C:\Program Files\Gensym\
g2-2011\g2\g2.ok"]
2004-09-30 10:00:28
2004-09-30 10:00:28   It is now OK to run G2!
```

width

Specifies the width in pixels of the window.

Platforms

All platforms

Syntax

```
-width number-of-pixels
```

number-of-pixels: A positive integer from 1 to 32,767.

Equivalent Environment Variable

None

Description

This option specifies the width in pixels of your window, which is stored in the g2-window-width attribute of the g2-window.

By default, G2 displays a window whose width is 90% of the height of the screen.

On Windows platforms, the width refers to the entire window, including the title bar and the black frame around the window. On UNIX platforms, the width refers to the client window area only; it does not include the width of the title bar and window frame.

Example

This command starts a process with a window whose width in pixels is 1000 and whose height in pixels is equivalent to 90% of the height of the screen, the default:

```
g2 -width 1000
```

window

Specifies the window name or class of g2-window that a secure G2 uses as its local window.

Platforms

All platforms

Syntax

`-window` *window-name-or-class*

window-name-or-class: Name or class of a g2-window item or of an item whose class is a subclass of g2-window.

Equivalent Environment Variable

None

Description

Specifying the `-window` option corresponds to the G2 Window Name or Class field in the login dialog.

Examples

This command starts a secure G2 process, specifying a window and other information necessary to start the process.

```
g2 -window viper -user-name howard -password fearnoevil
   -kb secure.kb -user-mode manager -language korean
```

window-style

Specifies the window style that G2 uses as its local window.

Platforms

All platforms

Syntax

```
-window-style {default | standard-large | g2-5.x | standard}
```

Equivalent Environment Variable

None

Description

This command-line option allows you to choose a larger version of the standard window style or the traditional G2 window style. By default, uses the `standard` window style in which workspaces have editable title bars and close buttons. Specifying `default` as the window style is the same as specifying `standard`.

For examples of each of these window styles, see "G2 Window Styles" in Chapter 2 "The Developers Environment" in the *G2 Reference Manual*.

Example

This command starts a G2 process, using a larger version of the standard window style:

```
g2 -window-style standard-large
```

x-magnification and y-magnification

Specifies the window's default ratio of horizontal (x-axis) or vertical (y-axis) magnification, in pixels per G2 workspace unit, for workspaces displayed at full scale.

Platforms

All platforms

Syntax

`-x-mag[nification]` *magnification-ratio*
`-y-mag[nification]` *magnification-ratio*

magnification-ratio: A decimal value from 0.50 to 2.66.

Equivalent Environment Variable

None

Description

For a display device that supports distinct settings for vertical and horizontal resolutions, you can use the `-x-magnification` and `-y-magnification` options to specify distinct horizontal and vertical magnifications.

These options specify the ratio of pixels per workspace unit at which the new window displays workspaces. For either option, specify a decimal value in the range 0.50 to 2.66.

If you do not specify the `-x-magnification` or `-y-magnification` options or the `-magnification` option, the new window uses a `-magnification` setting of 1.0; this is equivalent to a `-x-magnification` setting of 1.0 and a `-y-magnification` setting of 1.0.

By specifying different combinations of `-x-magnification` and `-y-magnification` settings, you can display workspaces in the process's window at an effectively equivalent absolute size on different display devices.

For more information about using the `-magnification`, `-x-magnification`, and `-y-magnification` options, see Specifying the Resolution and Magnification.

Special Considerations

Alternatively, you can use the `-magnification` option to specify the same setting for both the horizontal and vertical axes. However, do not combine the `-magnification` argument with either the `-x-magnification` or `-y-magnification` options. See magnification.

Example

This command starts a process whose window displays full-scale workspaces with a horizontal magnification of two pixels per G2 workspace unit and with a vertical magnification of one and a half pixels per G2 workspace unit:

```
g2 -x-magnification 2.0 -y-magnification 1.5
```

x-resolution and y-resolution

Specifies the horizontal (x-axis) and vertical (y-axis) resolution of the monitor on which a window appears.

Platforms

All platforms

Syntax

`-x-res[olution]` *dots-per-inch*
`-y-res[olution]` *dots-per-inch*

dots-per-inch: An integer in the range 36 to 200.

Equivalent Environment Variable

None

Description

For a display device that supports distinct settings for its horizontal and vertical resolution, you can use the `-x-resolution` and `-y-resolution` options to specify separate default horizontal and vertical resolutions.

These options specify the resolution as the number of dots per inch (dpi). By default, displays a window at a default resolution of 75 dpi.

These values are stored in the g2-window-x-resolution and g2-window-y-resolution attributes of the g2-window.

By specifying different combinations of `-x-resolution` and `-y-resolution` settings, you can display workspaces on a window at an effectively equivalent absolute size on different display devices.

For more on the `-resolution`, `-x-resolution`, and `-y-resolution` options, see Specifying the Resolution and Magnification.

Special Considerations

Alternatively, you can use the `-resolution` option to specify the same setting for both the horizontal and vertical axes. However, do not combine the `-resolution` option with either the `-x-resolution` or `-y-resolution` options. See `resolution`.

Example

This command line starts a process that displays its window at a horizontal resolution of 150 dpi, and a vertical resolution of 175 dpi:

```
g2 -x-resolution 150 -y-resolution 175
```

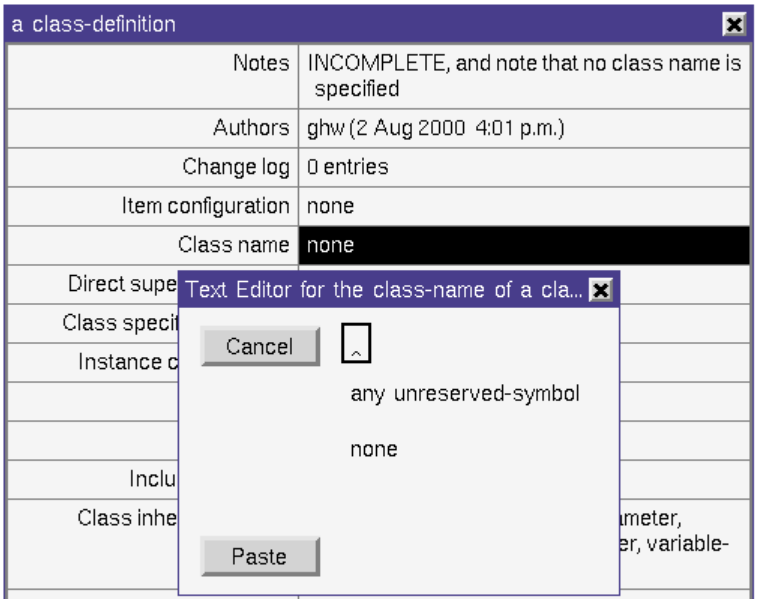
B

Reserved Symbols

Explains and lists G2's reserved symbols.

Introduction

G2 **reserved symbols** are symbols that cannot serve as a user-defined name in G2. In the Text Editor, the prompt any unreserved-symbol indicates where you must enter a user-defined name. In the example below, the text editor is open for specifying the name of a class. The G2 compiler will not accept a reserved word or the name of an existing class.



If you enter a reserved word for a class name, the G2 compiler will prevent you from exiting the editor and display this error message: This cannot be parsed. You cannot use a reserved symbol as the name of an item, the name of a class, the name of a class-specific attribute in a class definition, a region name in an icon description, or the name left of the colon in a language-translation.

List of Reserved Words

The three categories of reserved words are:

- Reserved words in the G2 language.
- Reserved ordinary system-defined attributes.
- Reserved hidden system-defined attributes.

Reserved Words in the G2 Language

Here is a list of reserved words that are keywords in the G2 language:

a above across ago all an and any at available-frame-vector average	be becomes begin below between by	call case change checking collection color-pattern conclude connected could count current
day days deviation do down during	else end every everything exists exit expiration	false first focus for format
giving go	has hour hours	if in infer input integral interpolated invoke is
last	maximum minimum minute minutes moved	name named nearest no none not nothing
of on one or output over	past per product	rate repeat return rules

same second seconds send set should simulated standard start subworkspace sum symbol	that the then there this to true	undefined unspecified until upon
value values via	wait was week weeks were when will with workspace	yes

Reserved Ordinary System-Defined Attributes

Here is the list of reserved ordinary system-defined attributes:

action action-priority allow-duplicate-elements annotations applicable-class array-is-permanent array-length attribute-displays attribute-initializations authors	background-color background-images blank-for-type-in?	categories change change-log characters-procedure chart-style class class-inheritance-path class-name class-of-object-computed-by-this class-of-procedure-invocation class-specific-attributes comment-procedure condition connection-arrows connector-formats cross-section-pattern

data-series data-server data-server-for-messages data-type default-cell-format default-evaluation-setting default-message-properties default-procedure-priority default-simulation-time-increment default-update-interval depth-first-backward-chaining-precedence depth-of-image description-of-frame dialog-height dialog-title dialog-type dialog-width dialog-x-position dialog-y-position direct-superior-classes disable-interleaving-of-large-messages dismissed-callback display-format display-update-interval display-update-priority display-wait-interval	element-type end-document-procedure end-element-procedure error-description error-procedure error-source-column error-source-item error-source-line expression-to-display external-simulator-configuration external-system-has-a-scheduler	fatal-error-procedure file-name-of-image file-status file-system first-class focal-classes focal-objects foreground-color format-for-type-in-box format-of-image formula frame-style

g2-connection-status g2-meter-name g2-routing-information g2-to-g2-interface-name g2-user-mode g2-user-name g2-window-height g2-window-initial-window-configuration-string g2-window-management-type g2-window-mode-is-valid g2-window-operating-system-type g2-window-remote-host-name g2-window-reroute-problem-report g2-window-specific-language g2-window-style g2-window-time-of-last-connection g2-window-user-is-valid g2-window-user-name-in-operating-system g2-window-width g2-window-x g2-window-x-resolution g2-window-y g2-window-y-resolution gfi-input-file-format gfi-input-file-pathname gfi-input-interface-object gfi-input-time-stamp-format gfi-input-variables-update-mode gfi-input-when-active gfi-output-file-format gfi-output-file-pathname gfi-output-file-update-frequency gfi-output-time-stamp-format gfi-output-values gfi-output-when-active grouping-specification gsi-application-name gsi-connection-configuration gsi-interface-name gsi-interface-status gsi-variable-status	have-edit-option-buttons-for-type-in? height-of-image history-keeping-spec	icon-description icp-connection-specification identifying-attributes ignore-gfi-input-base-time include-in-menus increment-per-dial-ruling increment-per-meter-ruling inherited-attributes initial-value initial-value-for-simulation initial-values initializable-system-attributes initialization-vector-for-external-simulator input-vector-to-external-simulator instance-configuration instantiate integration-algorithm interface-initialization-timeout-period interface-status interface-timeout-period interface-warning-message-level interpolate interval-to-poll-external-system inverse-of-relation item-configuration items-belonging-to-this-model

junction-block	keep-sorted	label label-to-display last-recorded-value line-pattern list-is-permanent low-value-for-dial-ruling low-value-for-meter-ruling
maximum-number-of-output-lines maximum-value message-keywords minimum-value model-simulation-status module-assignment	name-in-foreign-image name-in-remote-system names native-window-height native-window-left native-window-state native-window-top native-window-width notes number-of-pending-callbacks	off-value on-value options output-vector-from-external-simulator
patterns-definition plots point-formats poll-external-system-for-data prefer-buffered-drawing proprietary-package	qualified-name	readout-table-display-value relation-is-permanent relation-is-symmetric relation-name relation-summary remote-g2-expression remote-process-initialization-string renamed-gfi-output-file-pathname requires-call-next-method? rule-priority

save-image-data-with-kb scan-interval second-class send-all-values-at-beginning-of-cycle set-value-while-sliding? show-operator-logbook-in-this-window? show-prompts-for-type-in show-simulated-values simulation-control-specifications simulation-details simulation-formula simulation-procedure start-document-procedure start-element-procedure stub-length stubs superior-connection synchronized	table-size text text-conversion-style time-axis time-increment-for-update timeout-for-rule-completion timeout-interval title title-bar-text tokens-definition tracing-and-breakpoints trend-chart-format type-of-relation	uninterrupted-procedure-execution-limit update-callback uuid
validity-interval value-axes value-on-activation variable-or-parameter view-preferences	warning-procedure when-to-show-value width-of-image workspace-margin	

Reserved Hidden System-Defined Attributes

Here is the list of hidden system-defined attributes:

active-stubs attribute-display-items	background-color border-color button-status	cached-media-bin chart-axis-computed-details chart-data-series class connection-input connection-is-directed connection-output connection-position-sequence connection-style connection-vertices containing-module current-attribute-displays
default-window-position-and-scale do-not-strip-text-mark dynamic-breakpoints	edges-of-workspace effective-data-type evaluation-attributes	following-item-in-workspace-layering format-type foundation-class
g2-array-sequence g2-hash-table-number-of-entries g2-hash-table-sequence g2-list-sequence g2-priority-queue-number-of-entries g2-priority-queue-sequence g2-window-client-version g2-window-is-embedded g2-window-of-view g2-window-ui-style	history history-using-unix-time	icon-color icon-heading icon-reflection icon-variables image-data inlined-calls internal-media-bin item-active item-color-pattern item-height item-notes item-status item-width item-x-position item-y-position items-in-this-relation
last-recorded-value-text latent-listeners layer-position	manually-disabled? minimum-size-in-workspace mouse-cursor	name-box name-box-item

parent-of-subworkspace permanent position-in-workspace	relationships representation-type	selected-items selected-window-handle selected-workspace size-in-workspace slider-value strip-text-mark stripe-color
table-cells table-header table-rows text-alignment text-color text-font text-x-magnification text-y-magnification transient type-in-box-value type-in-box-variable-or-parameter	ui-client-connection-status ui-client-mode-is-valid ui-client-operating-system-type ui-client-remote-host-name ui-client-specific-language ui-client-time-of-last-connection ui-client-user-is-valid ui-client-user-mode ui-client-user-name ui-client-user-name-in-operating-system uses-floating-license	value-structure value-structure-using-unix-time value-to-display values-for-table-of-values
window-handles		

Generating a List of System-Defined Attributes

If *g2-attribute* is the name of a reserved attribute of a system-defined class, then you cannot use it as the name of an attribute of a user-defined class.

Some system-defined classes such as object or connection are user-extensible; other system-defined classes such as logbook-parameters are not user-extensible. To avoid possible inheritance problems, you cannot use system-defined attributes of user-extensible system-defined classes as user-defined attributes; thus, these attributes are considered reserved words in G2. However, you can use system-defined attributes of non user-extensible system-defined classes as user-defined attributes; these attributes are considered unreserved.

If you attempt to use a reserved word as a user-defined attribute, G2 takes the following actions:

- When entering a reserved word in the G2 Text Editor, an error such as the following appears in the text editor:

  This is uncompilable. HEIGHT-OF-IMAGE is the name of a G2 system attribute and cannot be a user-defined attribute."

- When loading a KB from an older version of G2 in which the reserved word was not a system-defined attribute, an error such as the following appears in the Operator Logbook:

  HEIGHT-OF-IMAGE is the name of a G2 system attribute and cannot be a user-defined attribute.

  Also, the notes of the user-defined class-definition contains an error such as the following:

  OK, and note that the class-specific-attribute height-of-image is now a reserved G2 attribute. You must rename it before starting G2.

To obtain the name of the user-defined class that uses the reserved word, use the following Inspect command:

highlight the symbol height-of-image in every class-definition

You can use the following system procedure to get a list of all reserved words:

g2-get-all-reserved-system-attribute-names
(*type*: symbol)
-> *reserved-words*: sequence

Returns a sequence of all reserved system-defined class attribute names, in alphabetical order, where *type* is one of these symbols:

- ordinary — Returns all non-hidden attributes.
- hidden — Returns all hidden attributes of user-extensible classes.
- all — Returns both ordinary and hidden attributes.

If *g2-hidden-attribute* is the name of a hidden attribute of a system-defined class, you may use it as the name of an attribute of a user-defined class. However, we recommend that you avoid this practice. For example, using history as the name of an attribute of a user-defined class would shadow its use as a hidden attribute of a float-parameter. Similarly, using containing-module as the name of an attribute of a user-defined class would shadow its use in GFR and GMS.

Note that you cannot use reserved symbols as the name of a user-defined attribute.

Mouse Gestures, Key Bindings, and Shortcut Keys

Presents the default mouse gestures, key bindings, and shortcut keys.

Introduction

Both G2 and Telewindows support standard mouse gestures for selection, where "standard" implies the Windows standard. They also support a number of other mouse gestures, key bindings, and shortcut keys for interacting with selection, workspaces, and items. G2 uses a selection style user interface where commands apply to the current selection.

The mouse gestures, key bindings, and shortcut keys are available on all platforms, and in both G2 and Telewindows, unless otherwise noted.

You can use configurations to change G2's default key bindings and shortcut keys and to make new assignments. For more information, see Configuring Keystrokes.

Note G2 configurations that restrict capabilities such as cloning an item do not, by default, restrict those capabilities in Telewindows. For example, if cloning is restricted for an item in G2, you can still copy the item by holding down the CTRL key and dragging the item. To restrict these types of user interface interactions, you can create configurations that restrict non-menu choices for selecting an object and selecting an area. For more information, see Configurations.

Mouse Gestures for Selection

Mouse Gesture	Action
Left-click an item	Select the item
Hold down the SHIFT key and click an item that is not selected	Add the selected item to an existing selection
Hold down the SHIFT key and click a selected item	Toggle the membership of the selected item in the selection
Left-click a workspace	Cancel the current selection and select the workspace
Drag in the open area of a workspace	Select all items in the rectangular area
Hold down the SHIFT key and drag an item	Expand the selection to include the item, then move the selection
Hold down the SHIFT key and drag in the open area of a workspace	Expand the selection to include the items in the rectangular area
Hold down the ALT key and click a connected network of items	Select the connected network of items
Hold down the ALT and SHIFT keys and left-click a connected network of items	Toggles the membership of the selected network of items in the selection

Mouse Gestures for Interacting with Selections

Mouse Gesture	Action
Right-click an item	Select the item and display its popup menu
Right-click an item that is part of a selection	Display a popup for interacting with the selection, which includes: Move, Clone, Transfer, Align, and Delete
Right-click in the open area of a workspace	Select the workspace and display its popup menu
Double-click an item	Select the item and display its attribute table
Double-click an attribute value in a table	Open the classic G2 text editor for editing the attribute value
Drag an item or selection	• For an item, select and move the item • For a selection, move the entire selection • For a connection between two items, move the connection and the items on both ends of the connection
Hold down the CTRL key and drag an item	Copy the item and attach it to the mouse, then lift the mouse button to place the copied item on the workspace

Mouse Gestures for Interacting with Workspaces

Key Binding	Action
Drag the title bar of a workspace	Move the workspace
With the right mouse button, drag the open area of a workspace or workspace view Hold down the CTRL or ALT key and drag the open area of a workspace or workspace view With the middle mouse button, drag the open area of a workspace or workspace view	Move the workspace or workspace view (Windows only)
Move the mouse wheel forward in a workspace	Move the workspace down by ten percent
Move the mouse wheel back in a workspace	Move the workspace up by ten percent
Hold down the CTRL key and move the mouse wheel forward in a workspace	Display the selected workspace at 80% of its current scale with the selected item in the center
Hold down the CTRL key and move the mouse wheel back in a workspace	Display the selected workspace at 120% of its current scale with the selected item in the center

Key Bindings for Scrolling Workspace Views

The following key bindings are only available for scrolling workspace views; therefore, they are only available when running Telewindows on Windows platforms. For more information on workspace views, see Chapter 3, "Using the Standard Telewindows Interface" in the *Telewindows User's Guide*.

Key Binding	Action
HOME	Scroll the workspace view to the top-left corner
END	Scroll the workspace view to the bottom-left corner
CTRL+HOME	Scroll the workspace view to the top-right corner
CTRL+END	Scroll the workspace view to the bottom-right corner
PAGE UP	Scroll the workspace view up
PAGE DOWN	Scroll the workspace view down
CTRL+PAGE UP	Scroll the workspace view left
CTRL+PAGE DOWN	Scroll the workspace view right
Arrow keys	Scroll the workspace view in the direction of the arrow
Hold down the CTRL key and press an arrow key	Scroll the workspace view one pixel at a time in the direction of the arrow

General Key Bindings

.

Key Binding	Action
DELETE on a selection	Delete all items in the selection with confirmation.
SPACEBAR on a selection	Perform the default action on the selected item, for example, displaying the item's table, editing an attribute, or pressing an action button.
RETURN or ALT+RETURN on a selection	Display the table(s) for the selected item(s). **Note:** Pressing Return when a workspace is selected does not display its table so as to avoid displaying the table in undesirable situations, such as when a G2 XL Spreadsheet (GXL) cell is selected.
ESC on a selection, table, or dialog box	Cancel the current selection, hide the table, or cancel the dialog box
Menu key or SHIFT+F10 on a selected item	Display the popup menu for the selected item.
TAB on a workspace	Select the next item in the layering from top to bottom.
SHIFT+TAB on a workspace	Select the previous item in the layering from top to bottom.
TAB in a table	Move the selected cell to the next cell in the table.
SHIFT+TAB in a table	Move the selected cell to the previous cell in the table.
DOWN ARROW or UP ARROW in a table	Move the selected cell to the next or previous cell, respectively.
Arrow key on a pulldown menu or menu choice	Move the selected pulldown menu or menu choice to the next or previous pulldown menu or menu choice.

Key Binding	Action
RETURN on a selected menu choice	Execute the menu choice.
RETURN on a selected pulldown menu	Dismiss the pulldown menu.
Arrow key on a selection	Move all items in the selection by 10% in the direction of the arrow.
Arrow key on a workspace	Move the selected workspace by 10% in the direction of the arrow.
Hold down the CTRL key and press an arrow key on a selection	Move all items in the selection by 1% in the direction of the arrow.
Hold down the CTRL key and press an arrow key on a workspace	Move the selected workspace by 1% in the direction of the arrow.
F1	Display a list of help topics, using Windows HTML Help (Windows only).
F5	Refresh.

General Shortcut Keys

Keystroke Command	Action
CTRL+A	In a workspace, select all items on the workspace; in the G2 Text Editor, dismiss the editor with confirmation.
CTRL+C	Refresh

Keystroke Command	Action
CTRL+O	Display the Load KB dialog
CTRL+Y	Display the Login dialog.
CTRL+Z Pause/Break key or Fn+Pause key	Pause G2.
CTRL + / CTRL + ?	Display the help screen for G2's default key bindings. The help screen can be also displayed programmatically by calling the g2-system-command system procedure, described in the *G2 System Procedures Reference Manual*.

Shortcut Keys for Workspaces

Keystroke Command	Action
Workspace Scaling	
CTRL+B CTRL+PLUS SIGN (+) CTRL+EQUAL SIGN (=)	Display the selected workspace at 120 percent of its current scale.
CTRL+F CTRL+0 (zero)	Display the selected workspace at full scale.
ALT+F	Display the selected workspace at normalized full scale, where one window unit = one workspace unit = one pixel.
CTRL+N	Display the selected workspace at 80 percent of its current horizontal scale.
CTRL+P	Circulate the selected workspace up in the stack of workspaces.

Keystroke Command	Action
CTRL+Q	Display the selected workspace at one-sixteenth of its current scale, that is, at one-fourth of the workspace's current *x*-axis scale and one-fourth of its current *y*-axis scale.
CTRL+S CTRL+MINUS SIGN (-) CTRL+UNDERSCORE (_)	Display the selected workspace at 80 percent of its current scale.
CTRL+W	Display the selected workspace at 120 percent of its current horizontal scale.
CTRL + .	Display the selected workspace centered within its window, at full scale, or the largest scale that makes the workspace completely visible, whichever is smaller.
ALT + .	Display the selected workspace at its maximum scale to fit within its window.
CTRL+4	Display the selected workspace at four times its current scale.
Workspace Movement	
CTRL+D ↓ (DOWN-ARROW)	Move the selected workspace down by ten percent of its height.
ALT+D ALT + ↓	Move the selected workspace down by one percent of its height.
CTRL+U ↑ (UP-ARROW)	Move the selected workspace up by ten percent of the height of its window.
ALT+U ALT + ↑	Move the selected workspace up by one percent of the height of its window.
CTRL+L ← (LEFT-ARROW)	Move the selected workspace left by ten percent of the width of its window.

Keystroke Command	Action
ALT+L ALT + ←	Move the selected workspace left by one percent of the width of its window.
CTRL+R → (RIGHT-ARROW)	Move the selected workspace right by ten percent of the width of its window.
ALT+R ALT + →	Move the selected workspace right by one percent of the width of its window.
CTRL+T	Lift the selected workspace to the top.
CTRL+V	Drop the selected workspace to the bottom.
CTRL+I	Lift the bottom workspace to the top.
CTRL+TAB CTRL+F6	Select the next workspace in the stack of workspaces.
CTRL+SHIFT+TAB CTRL+SHIFT+F6	Select the previous workspace in the stack of workspaces.
CTRL+F4	Close the selected workspace.

Changes from Earlier G2 Versions

To implement standard selection and other standard interactions, G2 reimplemented the behavior of the following mouse gestures and shortcut keys available in earlier versions of G2.

Mouse Gesture/ Keystroke Command	G2 6.x Behavior	G2 7.0 Behavior
Left-click an item	Display the popup menu for the item	Select the item and give focus to the workspace of the selected item.
Left-click a workspace	Display the KB Workspace menu	Cancel the current selection and select the workspace.
Drag in the open area of a workspace	Move the workspace	Select all items in the rectangular area.
CTRL+A	In a text editor, dismiss the text editor; no effect in a workspace	Select all items on the selected workspace.
CTRL+O	Move the selected workspace so that its origin is at the center of its window	Display the Load KB dialog.

Syntax Conventions

Describes the notation and user-specified terms used in G2 syntax.

Introduction

This book uses these types of conventions for describing G2 syntax:

- **Descriptive notation** describes how you specify the syntax.
- **User-specified terms** describe the type of information you can specify.

Syntax Notation

The following notation describes G2 syntax. In the descriptions of the syntax notations, a **syntax element** is a self-contained piece of syntax, for example, a stand-alone user-specified term, or a set of self-contained syntax enclosed by { } or [ ]. Except for the last notation, do not enter these notation characters in G2 code.

This notation...	Has this meaning...
{ }	Groups syntax elements.
{*choice* | *choice*}	Separates alternative syntax elements enclosed by { }.

This notation...	Has this meaning...
[ ]	Encloses optional syntax elements.
... [, ...] [; ...]	Indicates that the preceding syntax element can be repeated any number of times, with or without a separator, depending on the notation.
[ ]	Required characters used to signify list and array elements and text expressions.
:=	Defines a user-specified term.
->	Indicates the value returned by an expression.

All other characters that appear in syntax descriptions are literal characters that you must enter.

User-Specified Terms

The syntax descriptions use a number of standard terms to represent user-specified syntax. User-specified terms are shown like this: *term*. When entering a statement in the G2 editor, substitute your own term for *term*.

There are two general types of user-specified terms:

- **Expressions**, which G2 evaluates to return an item or value.
- **Literal terms**, which are a sequence of characters that G2 does not evaluate.

There are several categories of user specified terms:

- **Values terms**, which are expressions and literal terms that represent values of a particular type.
- **Item terms**, which are expressions and literal terms that represent items of a particular class.
- **Class terms**, which are literal symbols that name a particular class or one of its subclasses.
- **Attribute terms**, which are literal symbols that name an attribute.
- **Other terms**, which are expressions and literal terms that represent various types of G2 entities other than those mentioned above.

Value Expression Terms

These are terms that represent expressions that evaluate to values of a particular type. They do not evaluate to items.

This term...	Is an expression that evaluates to a value of type...
value-expression	float, integer, symbol, text, or truth-value
quantity-expression	float or integer
float-expression	float
integer-expression	integer
symbolic-expression	symbol
text-expression	text
truth-value-expression	truth-value

For expressions that evaluate to a value of type item-or-value, see Item Expression Terms.

For more information, see Expressions.

Literal Value Terms

These are terms that represent literal values of a particular type. They do not represent value expressions or items.

This term...	Is a series of characters that signifies a value of type...
value	float, integer, symbol, text, or truth-value
quantity	float or integer
float	float
integer	integer
symbol	symbol
text	text
truth-value	truth-value

For more information on literal values, see Evaluating Expressions.

Also, when the syntax requires a literal symbol, it rarely uses the term *symbol*. Instead, it uses a more specific term that indicates the kind of symbol. For example, the following user-specified terms are literal symbols: *class-name*, *attribute-name*, *connection-class-name*, and *local-name*.

Item Expression Terms

These are terms that represent expressions that evaluate to items. They do not represent values. Here are a few common examples.

This term...	Is an expression that evaluates to an item of the class or a subclass of...
item	item
variable	variable
parameter	parameter
g2-array	g2-array
g2-list	g2-list
connection	connection
g2-window	g2-window
kb-workspace	kb-workspace

In addition, the following expression evaluates to an item or value:

This term...	Is an expression that evaluates to...
item-or-value	• A value of type float, integer, symbol, text, or truth-value, or • An item of any class.

Note Anywhere that the syntax uses the expression *item* or *item-or-value*, you can substitute a *generic-reference-expression*, preceded by one of the generic-reference prefixes (the, any, every, each, a, or an), depending on the context. For more information, see Other Expression Terms.

Attribute Reference Terms

This term represents an expression that refers to an attribute.

This term...	Is an expression that evaluates to...
attribute	Any direct or indirect reference to any system-defined or user-defined attribute.

Item Name Terms

These are terms that represent literal symbols that name an item. Here are some examples that appear most commonly in syntax.

This term...	Is a literal symbol that names an item of the class or a subclass of...
item-name	item
procedure-name	procedure
relation-name	relation
kb-workspace-name	kb-workspace

Class Name Terms

These are terms that represent literal symbols that name a class.

This term...	Is a literal symbol that names an item of the class or a subclass of...
class-name	Any class.
connection-class-name	The connection class.

For convenience, the value of the class-name attribute of a class-definition refers to the class-definition item. For example, the action move tank by (20, 20) moves the class-definition item that defines the tank class.

Attribute Name Terms

These terms represent literal symbols that name an attribute.

This term...	Is a literal symbol that names...
attribute-name	Any system-defined or user-defined attribute.
quantity-attribute-name	Any user-defined attribute declared with type quantity.
item-attribute-name	An attribute declared to be an instance of some class or given by a variable or parameter.
simple-attribute-name	Any system-defined or user-defined attribute that contains any value, except an item.
typed-attribute-name	A user-defined attribute declared with any type.
untyped-attribute	A user-defined attribute not declared with a type.

Other Expression Terms

These are general-purpose and special-purpose expressions that the syntax uses.

This term...	Is an expression that...
time-expression	Evaluates to a value of type integer or float, which is interpreted as a number of seconds. The syntax is: *integer-expression* {seconds | minutes | hours | days | weeks} For example, 5 seconds, 5 minutes, 5 hours, the current time +15, and 2000 return integer values, whereas 1.5 seconds and the current subsecond time + 12.5 return float values.
ddd.dddd-format	Formats the display of a floating point number, by indicating the number of decimal digits to display to the left and right of the decimal point. For example, the expression ddd.dd formats a floating point number to the hundredths decimal place. See Formatting Numeric Values.
x, y	Evaluates to a value of type integer or float, which is interpreted as x and y coordinates measured in workspace units.
generic-reference-expression	Refers generically to a set of classes or types. It has the following syntax: {*class-name* | *type*} [*local-name*] [*generic-reference-qualifier*] A generic reference expression is always preceded by a generic reference prefix, which depends on the context. The available prefixes are: the, any, every, each, a, or an. The syntax only uses this reference when it *requires* a generic reference. In addition, you can use a generic reference expression preceded by the prefix the, any, or every whenever the syntax refers to *item*.

This term...	Is an expression that...
generic-reference-qualifier	Is used in the composition of a generic reference expression. It has the following syntax: *generic-reference-qualifier* := {upon *kb-workspace*} | {connected *connected-expression*} | {at *at-expression*} | {nearest to *item*} | {superior to *item*} | {that is *relation-name item*} | {named by *symbolic-expression*} | {in {*g2-list* | *g2-array*} } | {name of *item*}
event-expression	Refers to an event that G2 detects when processing whenever rules. It has the following syntax: {*variable* | *parameter*} receives a value | *variable* fails to receive a value | *object* is moved {by the user | by g2} | *item* {becomes | ceases to be} *relation-name item*
statement	Defines a clause in a procedure, which can be any *action* or another procedure *statement*.
action	Defines an action in a rule or procedure statement.

Other Literal Terms

These are terms that represent literal terms other than items or values.

This term...	Is a literal term that...
argument	Names an argument in a procedure, whose type you declare in the procedure.
color-name	A symbol naming one of G2's supported colors. You can also provide a symbol of the form RGB*rrggbb* as a valid color name, where *rr*, *gg*, *bb*, are the 8-bit hex values for red, green, and blue. The full 24-bit color is used for drawing if the window is capable of it; otherwise, the closest Gensym standard color is used. You use the rgb-symbol function to convert the RGB values to a symbol. See Rgb-Symbol Function.
color-attribute-name	Any attribute of a class that defines a color region, which vary depending on the item: foreground-color, background-color, text-color, border-color, icon-color, stripe-color.
item-location	Is any of: center, left center, right center, top center, top left corner, top right corner, bottom center, bottom left corner, bottom right corner.
local-name	Names a local name in a procedure, whose type you declare in the procedure.
module-name	Names a module.
portname	Names a port, used in stub definitions.
region-name	Names an icon region.
rule-category-name	Names a category specified in the Categories attribute of a rule.
time-unit	Represents a unit of time: second, seconds, minute, minutes, hour, hours, day, days, week, weeks.

This term...	Is a literal term that...
type	Refers to any G2 value type (item-or-value, value, quantity, float, integer, symbol, truth-value, or text).
window-location	Is any of: center, left, right, top, or bottom.
workspace-location	Is any of: top left, top right, top center, right center, bottom left, bottom center, bottom right, or left center.

E

G2 KBs and GIF Files

Describes the demonstration, sample, and utility KBs, and the GIF files that ship

Introduction

The `g2` directory of your G2 product installation directory includes a subdirectory named `kbs`, which contains a variety of KBs and other resources that can be useful to the G2 developer. The `kbs` directory contains four subdirectories:

- `demos`
- `samples`
- `tutors`
- `utils`

This appendix describes the resources that are available in these subdirectories.

Demonstration KBs

The following KBs are available in the `demos` directory:

KB Name	Description
`axldemo.kb`	Provides a demonstration for G2 ActiveXLink.
`business.kb`	Simulates a number of different business processes, including orders, inventory, stocking, and purchasing.
`dialogs-demo.kb`	Provides examples of how to implement custom and built-in Windows dialogs for viewing in Telewindows.
`explnfac.kb`	Provides an example of how to use the Explanation facility for tracing rule execution through variables.
`g2-80r0-doc-examples.kb`	Provides examples used in the documentation for G2 Version 8.0 Rev. 0 features.
`g2-80r0-doc-examples-remote.kb`	Provides examples used in the documentation for the G2 publish-subscribe facility.
`g2gl-credit-rating-example.kb`	Provides a demonstration of the G2 Graphical Language (G2GL).
`gms-native-language-demo.bk`	Shows how localized G2 Menu System (GMS) menus render as native Windows menus when viewed through Telewindows.
`gms-native-large-menu-demo.bk`	Shows various features of the G2 Menu System (GMS) menu bar template, which renders as a native Windows menu bar in Telewindows.
`gms-native-multiple-menu-bar-demo.bk`	Shows various features of the G2 Menu System (GMS) menus rendered as native Windows menus in Telewindows.
`gms-native-popup-menu-demo.kb`	Shows G2 Menu System (GMS) popup menus rendered as native Windows popups when viewed through Telewindows.
`kbtools.kb`	The top-level module of a library of buttons subclasses from uil-button.
`mill.kb`	Illustrates how to develop a simulation of discrete events. Also demonstrates icon animation techniques.

KB Name	Description
nms-demo.kb	Provides examples of how to use the Native Menu System (NMS) to create and manipulate native Windows menus when viewed through Telewindows.
publish-subscribe-doc-ex.kb	Provides examples used in the documentation for the G2 publish-subscribe facility.
publish-subscribe-remote-doc-ex.kb	Provides examples used in the documentation for the G2 publish-subscribe facility.
space.kb	Simulates and diagnoses various failures on board an orbiting satellite.

Note This directory also contains a number of KBs found in the utils directory to support some of the demos. For a description, see Utility KBs.

Sample KBs

The following KBs are available in the samples directory:

KB Name	Description
chaos.kb	Demonstrates the ability of a trend-chart to plot what is known in chaos theory as a bifurcation diagram.
charts.kb	Provides examples and demonstrations of all G2 charts, including Windows chart views as well as classic G2 charts.
fgntest.kb	A sample KB that works in conjunction with sample foreign function test files to demonstrate how one might use foreign functions.
g2tog2.kb	Demonstrates a technique for moving objects between different G2s.
gsi_exam.kb	Contains seven examples that exercise various aspects of GSI. Three examples demonstrate item passing features.
image.kb	Demonstrates background images in workspaces, workspace borders, and image definitions.
itempass.kb	Demonstrates item passing between G2 processes and between G2 and GSI.

KB Name	Description
japanese.kb	Provides local translations of all G2 system-defined menu choices.
language.kb	Provides local translations of all G2 system-defined menu choices for several European languages.
objpass.kb	Demonstrates the object-passing capabilities between two G2 processes.
profile-demo.kb	Provides a simple example of how to obtain profile data about executable items in a KB.
sptools.kb	The top level module of a set of several system procedure examples.
statfun.kb	Provides a library of statistical functions.
twgame.kb	Presents a simple game to play between two or more concurrent Telewindow's users.
twtour.kb	A documented demonstration KB that illustrates methodology for switching a Telewindows session from one G2 process to another.

Tutorial KBs

The following G2 utilities are available in the tutors directory.

KB Name	Description
ch2.kb ch2soln.kb	The starting KB and solution KB for Chapter 2 in the *Getting Started with G2 Tutorials*.
ch3.kb ch3soln.kb	The starting KB and solution KB for Chapter 3 in the *Getting Started with G2 Tutorials*.
ch4.kb ch4soln.kb	The starting KB and solution KB for Chapter 4 in the *Getting Started with G2 Tutorials*.
ch5.kb	The starting KB for Chapter 5 in the *Getting Started with G2 Tutorials*.
solution.kb	The solution KB for Chapter 5 in the *Getting Started with G2 Tutorials*.

Utility KBs

The following G2 utilities are available in the `utils` directory.

KB Name	Description
`g2com.kb`	The KB used to run G2 ActiveXLink.
`g2cuidev.kb`	The top-level module of GDI, which requires `g2uimenu`, `g2uifile`, `g2uitree`, and `g2uiprnt`.
`g2uifile.kb`	The dialogs used by GDI, triggered from menu selections in g2uimenu.
`g2uimenu.kb`	The GMS menus for the G2 menus of GDI.
`g2uiprnt.kb`	The print dialog.
`g2uitree.kb`	The tree-view control and object manager.
`gdddemo.kb`	Demo KB for GDD. Contains examples of dynamic displays.
`gdddev.kb`	Tools for developing GDD displays. Once the dynamic display has been created, the gdddev module can be removed from your KB.
`gddlib.kb`	Samples of the three basic kinds of dynamic displays. Each sample can be easily customized by using the icon editor and updating the attributes.
`gddroot.kb`	The tools procedures to maintain a dynamic display once it has been installed in an application. This is the only GDD KB necessary once the displays are developed.
`gfr.kb`	The G2 Foundation Resources utility.
`gms.kb`	The G2 Menu System utility for adding a pulldown menu system to a KB.
`gmsdemo.kb`	A demo of the features of GMS.
`gold.kb`	The G2 OnLine Documentation system KB.
`goldui.kb`	The GOLD user interface KB.
`guicolor.kb`	A `guitools.kb` module that provides color selection dialog.
`guidata.kb`	A `guitools.kb` module that provides support for editing attributes configured as lists or arrays.

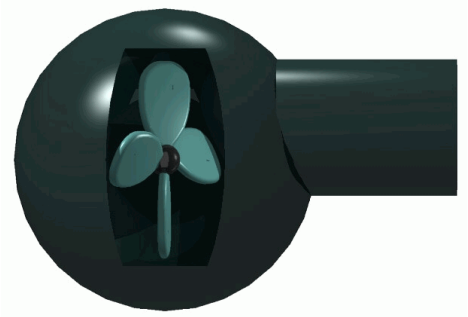
KB Name	Description
guide.kb	The top-level GUIDE/UIL module. Provides graphical editors and front-end to all UIL objects. Load this module to build GUIDE applications.
guidelib.kb	The GUIDE library.
guidemo.kb	An online tutorial for using GUIDE.
guidesa.kb	GUIDE objects for creating and editing scroll areas.
guigfr.kb	A guitools.kb module that provides dialogs for editing GFR resources.
guimove.kb	A guitools.kb module that provides a move dialog with which you can adjust the x and y coordinates of any item.
guislide.kb	A module that provides an example of subclassing a UIL object; in particular, a slider control. Can load as a stand-alone module.
guitools.kb	The top-level module of a set of GUI tools.
gxl.kb	The G2 XL Spreadsheet (GXL) utility.
gxldemo.kb	A demonstration of the GXL features.
japanese.kl	The Japanese language facilities.
jiscodes.kl	The Japanese Industrial Standard (JIS) codes.
korean.kl	The Korean language facilities.
kscodes.kl	The KSC-5601 codes on a series of workspaces.
language.kl	All of the language facilities (except for Japanese and Korean) appear in this KB.
profiler.kb	A complete profiling utility for obtaining information about executable items in a KB.
profroot.kb	The top-level KB for the profiler.kb.
starter.kb	A module that contains the GOLD book objects for documents in the G2 Starter Kit.
sys-mod.kb	A modularized KB containing all of the G2 system procedures.
uil.kb	A UIL module that provides a general API to all UIL objects.
uilcombo.kb	A UIL module that supports combo boxes.

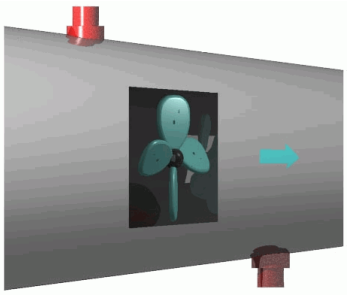
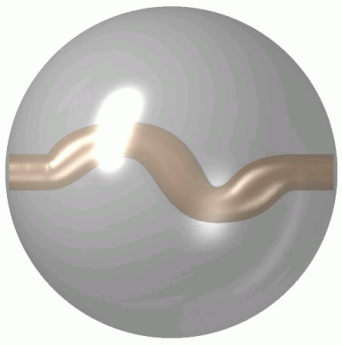
KB Name	Description
uildefs.kb	A UIL module that provides definitions for UIL objects.
uillib.kb	The UIL main library.
uilroot.kb	A UIL module that supports the creation, configuration, activation and deletion of navigation buttons.
uilsa.kb	UIL API and support for scroll areas.
uilslide.kb	UIL API and support for sliders.
uiltdlg.kb	UIL API and support for tabbed dialogs.

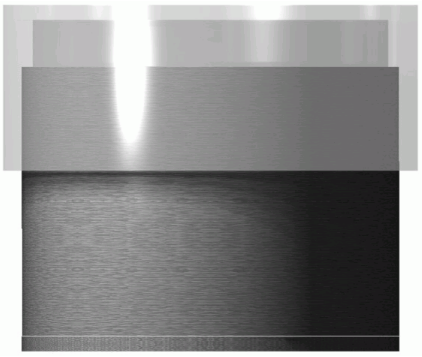
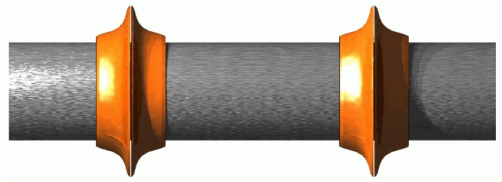
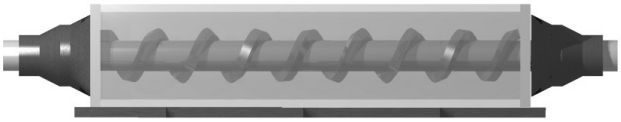
GIF Files

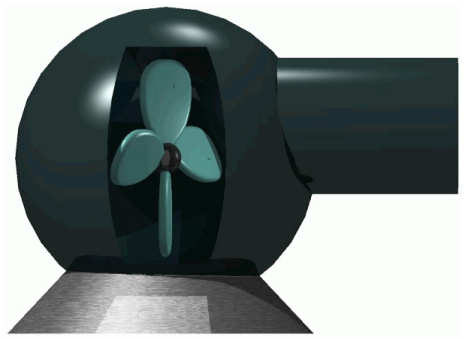
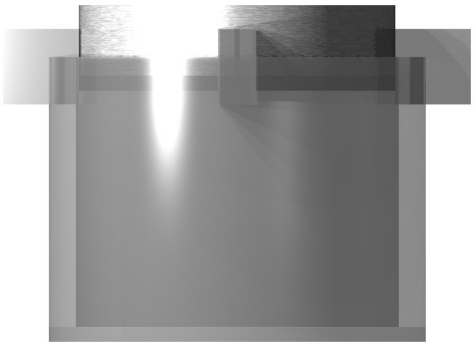
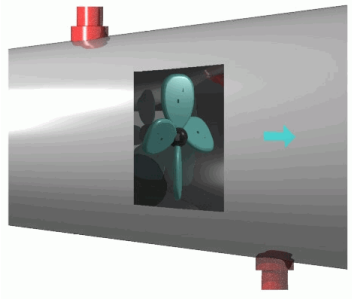
Several GIF files are also included in the demos directory. They can be used in images and definitions, and as the backgrounds of workspaces. These icons are for use *only* within G2 knowledge bases. Use with any other application is prohibited.

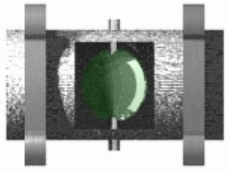
GIF File	Example or Description
agitatr1.gif	A large agitator.
agitatr2.gif	A second large agitator.
attank.gif	A large tank.
bin.gif	A large bin.
blower.gif	



GIF File	Example or Description
car.gif	
comprssr.gif	
disttwr.gif	A large distiller.
exchgr.gif	

GIF File	Example or Description
filter1.gif	A large filter.
filter2.gif	A large filter.
furn1.gif	A large furnace.
furn2.gif	A large furnace.
gashldr.gif	
hopper1.gif	A large hopper.
kiln1.gif	
mixer.gif	

GIF File	Example or Description
pump.gif	
reactor.gif	A large reactor.
tankroof.gif	
turbine.gif	
vessel1.gif	A large vessel.
vessel2.gif	A large vessel.
vessel3.gif	A large vessel.

GIF File	Example or Description
vessel4.gif	A large vessel.
vlv3way.gif	
vlvcklg.gif	A large version of vlvcksm.gif.
vlvcksm.gif	
vlvmanlg.gif	A larger version of vlvmansm.gif.
vlvmansm.gif	
world.gif	A large world map.

F

Superseded Practices

Describes G2 capabilities that are obsolete and may not be supported indefinitely.

Introduction

The following G2 practices and capabilities are superseded as of G2 5.0 and might or might not be supported in future releases of G2. These capabilities should not be used in new code and should be replaced in existing code:

- Attribute files
- Unscheduled drawing mode
- XOR drawing mode
- G2 File Interface (GFI)
- G2 Simulator

- Icon position and size attributes
- OLE drag and drop

This chapter contains a brief description of each of the capabilities listed. For information about them, see the *G2 Superseded Practices* document, which is available from Gensym upon request.

Some other G2 practices are also superseded, but Gensym currently intends to support them indefinitely. Such practices and their replacements are described where appropriate in the *G2 Reference Manual*.

Attribute Files

G2 can set item attributes by loading values from a text file, called an attribute file, via the Main Menu > Miscellany > Load Attribute File menu choice.

Attribute files are a superseded capability as of G2 5.0. In place of attribute file I/O, G2 provides a number of system procedures that do file I/O in a much more convenient and efficient way. These are described in the *G2 System Procedures Reference Manual* in the File Operations section.

For information about attribute file operations, see the *G2 Superseded Practices* document, available from Gensym on request.

Drawing Modes

Two drawing modes have been superseded as of G2 5.0:

- Unscheduled drawing mode
- XOR drawing mode

For information about drawing modes, see the *G2 Superseded Practices* document, available from Gensym on request.

Unscheduled Drawing

The allow-scheduled-drawing? attribute in the Drawing Parameters system table determines whether drawing is performed immediately (attribute is no) or by a scheduled task (attribute is yes).

As of G2 5.0, immediate drawing is a superseded practice. In place of immediate drawing, always use scheduled drawing by specifying allow-scheduled-drawing? as yes.

XOR Drawing Mode

The paint-mode? attribute in the Drawing Parameters system table specifies whether drawing is done in XOR mode (attribute is no) or Paint mode (attribute is yes).

As of G2 5.0, XOR mode is a superseded practice. In place of XOR mode, always use Paint mode by specifying paint-mode? as yes. Paint mode is the default mode when you start G2.

G2 File Interface (GFI)

The G2 File Interface (GFI) utility can read and write external data files. Such files can capture a log of KB execution, initialize a KB when execution begins, and provide synchronous or asynchronous events during KB execution.

GFI is a superseded capability. In place of GFI, G2 provides a number of system procedures that do file I/O in a more convenient and efficient way. These are described in the *G2 System Procedures Reference Manual*.

For information about GFI, see the *G2 Superseded Practices* document, available from Gensym on request.

G2 Simulator

The G2 simulator is a special kind of data server that provides simulated values for variables and parameters while G2 runs. The Simulation Parameters system table has a simulator-on? attribute, which controls the simulator and is no by default. The simulator does nothing unless you explicitly turn it on.

The G2 simulator is a superseded capability. In place of the simulator, use rules and procedures to simulate data, or obtain one of the many third-party software products that do simulation.

For information about the G2 simulator, see the *G2 Superseded Practices* document, available from Gensym on request.

Icon Position and Size Attributes

The icon-x-position, icon-y-position, icon-height, and icon-width attributes have been superseded in favor of item-x-position, item-y-position, item-height, and item-width. The superseded attributes still appear as prompts in the editor, because they exist for backward compatibility.

OLE Drag and Drop

The OLE drag and drop facility is a superseded practice. Use the following facilities as alternatives to OLE drag and drop:

Use this facility...	Instead of this superseded facility...
Cutting and pasting	Dragging text between applications using copy to OLE and cut to OLE.
Item passing	Dragging items between G2 processes using ole-clone and ole-transfer.

For information on these alternative facilities, see:

- Cutting/Pasting between G2 and Other Applications.
- Passing User- and System-Defined Classes.

A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z

A

abstract class: User- or system-defined class that cannot be instantiated. An abstract class can be a direct superior class, and can provide definitions common to its subclasses. *Contrast with* concrete class.

abstract type: A name declared with an abstract type contains a value whose specific type is any of that abstract type's subtypes. G2 offers these abstract types: item-or-value, value, and quantity. *Contrast with* specific type *and* composite type.

action: A command that G2 can execute within a rule, procedure, action button, or user menu choice.

activatable subworkspace: The subworkspace of an item that has been configured using this configuration statement:

```
declare properties as follows : activatable-subworkspace
```

allocation message: A message that G2 prints on the console when it allocates more memory during KB loading or execution.

allocation report: An allocation table or message.

allocation table: A table that G2 prints on the console when G2 begins execution. The table describes memory requirements, defaults, requests, and initial allocations.

annotation: The textual representation of compound attributes. For charts, the annotation describes the chart type and format. For trend charts, an annotation is a detailed syntactical description of the non-default values of component attributes and component defaults.

antecedent: In a rule, the phrase that specifies the conditions for executing the actions in the rule's consequent. A rule's antecedent specifies a truth-value expression. *Contrast with* consequent.

area box: A heavy-bordered rectangle within which you select an area of a workspace, as part of the Operate on Area workspace menu choice.

argument: A value or item that is passed to an invoked procedure or function.

asynchronous mode: A GFI input mode that typically provides a KB with asynchronous events that occur over the course of the KB's execution.

attribute: A characteristic or property of an item (any object, connection, workspace, and so on) in a G2 knowledge base. An item's set of attributes is defined by the item's class.

attribute descriptions: The detailed internal specifications of an attribute. All attribute descriptions appear in the *G2 Class Reference Manual*.

attribute display: A display that shows the values, and optionally the names, of one or more attributes of an item. You can move an attribute display by dragging with the mouse, and you can edit an attribute's value after clicking the mouse on an attribute display. You can also apply configurations to an item's attribute displays. *Contrast with* name box.

attribute file: A text file, external to G2, that specifies values for one or more attributes of one or more existing items in a KB. Use an attribute file to populate programmatically the attribute values of existing items.

attribute table: A two-column table that shows the name and value of each attribute of an item.

authorization file: Also known as the `g2.ok` file. This file identifies each authorized user and optionally associates each user name with a user mode.

authorization level: A method for managing G2 license options with Gensym's Telewindows license options.

B

background: In G2's window, the visible pattern upon which G2 displays the current KB's workspaces. (Clicking the mouse on this background displays the Main Menu.) In a workspace, the visible color or image upon which the workspace's items appear to reside. (Clicking the mouse on this background displays the KB Workspace menu.)

background-color color attribute: A color attribute for a workspace, rule, generic formula, or other item with a text box representation style.

backward chaining: An inferencing technique related to data seeking, in which G2 seeks the value of a variable by invoking rules that can conclude the variable's value. *Contrast with* forward chaining.

base time: The time on which the time-stamps in a GFI file are based. For a file that uses explicit time-stamps, the base time is the earliest time in the file. For a file that uses relative time-stamps, the base time is kept separately in the file's header line.

begin-end block: A compound statement in a procedure or method. The compound statement begins with the reserved symbol begin, includes two or more statements, and ends with the reserved symbol end.

border-color color attribute: The color attribute for the border of a workspace, rule, generic formula, or other item with a text box representation style.

breadth-first backward chaining: The activity where G2 simultaneously invokes every rule that is able to conclude a new value for the target variable. G2 schedules each invoked rule according to its declared priority. *Contrast with* depth-first backward chaining.

breakpoint: A certain point within an executing statement at which the KB stops processing. Breakpoints are a debugging tool. G2 provides four levels of breakpoints, ranging from no breakpoints to breakpoints at every step, which you can set for global scope in the Debugging Parameters system table and for local scope in the tracing-and-breakpoints attribute of individual executable items.

bridge: An application program that passes items and values to and from G2. A bridge application exchanges data between G2 and another application, device (such as a programmable logic controller), or system. A bridge application typically calls functions provided in Gensym's G2 Gateway product.

buttons: Items that are among the components of a G2 application's user interface. Buttons perform actions or provide values for variables and parameters.

C

cardinality: A property of a relation that indicates how many instances of a relation's first class can be related to how many instances of the relation's second class. A relation's cardinality can be one-to-one, one-to-many, many-to-one, or many-to-many.

category: A named association of rules. To associate a rule with a rule category, enter a non-reserved symbol in the rule's categories attribute; this names a rule category and associates the rule with that category. Use the invoke action to invoke all rules for a category.

causal knowledge: Knowledge about cause and effect relationships.

class: A group of items that have the same icon, attributes, and behavior. Classes are organized into a hierarchy, in which each class inherits the attributes of its superior class, but may have additional attributes of its own.

class hierarchy: A hierarchical structure of class definitions that is built into G2, but can be extended by the user.

class inheritance path: A linear list of all the classes from which a class inherits.

class list workspace: A particular kind of workspace that presents lists of classes, items, or other entities for entering in the Text Editor.

class-specific attribute: An attribute that is specified by a class's definition, rather than inherited from the definitions of its superior classes.

clock tick: The fundamental unit of time within G2. The time interval of each clock tick is determined by the user-settable minimum-scheduling-interval attribute of the Timing Parameters system table.

collection time: The instant in time when a variable or parameter receives a value. For a variable, G2 uses this value and the variable's validity interval to determine an expiration time for each collected value.

color attribute: A named portion of an item's knowledge that can have a color value, such as the background-color color attribute of workspaces. Color attributes do not appear in the attribute tables of items.

Color Indicator: In the Icon Editor, a display that names the color of the current layer.

command-line option: Included in the operating system command that launches a new G2 process, this keyword affects how the new G2 starts and runs.

compilation configuration: A configuration that affects how G2 compiles attributes that refer to classes, items, and attributes.

compilation dependency: A relationship between an item with compiled attributes and other items, where the item's compiled attributes refer to the knowledge of those items.

compiled attribute: An attribute that can contain an expression, an action, or a statement.

component: For a trend chart, a building block that customizes a charting feature or a representation of data.

component reference: In a trend chart, the number or name by which you specify a particular component of a compound attribute.

component subtable: One method of displaying compound attribute components in trend charts. A component subtable lists all of the component attributes, and provides a means of interactively customizing a component.

composite attributes: Attributes that appear in an attribute table as one attribute, but which are composed of more than one subattribute.

composite type: A type that is composed of one or more values of any general, specific, or composite type. The G2 composite types are structure and sequence. *Contrast with* abstract type *and* specific type.

compound attribute: One or more components in a trend chart.

concrete class: User- or system-defined class that can be instantiated. *Contrast with* abstract class.

concurrent execution: The process of executing statements, including actions, in parallel. When G2 executes a set of statements concurrently, the result is as if all statements had executed at the same time. *Contrast with* sequential execution.

configuration: A declaration that changes the default behavior of items.

conflict workspace: A permanent, named workspace that G2 creates and displays after loading two or more modularized KB files, or after merging a KB file into the current KB, where definitions of classes differ among the loaded KB files.

connection post: An object that you use to draw a connection from one workspace to another, from one object to another object on its subworkspace, or across a single workspace. G2 considers that items connected to two connection posts with the same name are also connected to each other.

consequent: In a rule, the phrase that specifies one or more actions for G2 to perform, either sequentially or simultaneously. G2 performs these actions only if evaluating the expression in the rule's antecedent produces a sufficient truth-value. *Contrast with* antecedent.

consistently modularized: A KB file or the current KB is consistently modularized when: every module is named in its own Module Information system table; there is only one module that is not directly required by any other module; every top-level workspace is associated with a module; for each attribute table that is transferred to a workspace, that workspace and the table's superior item are associated with the same module; modules containing definitions are directly required by all modules that contain instances of those definitions's classes; there are no cyclic dependencies in the module hierarchy. *Contrast with* unmodularized.

current KB: The items contained in the memory of a running G2 process.

current knowledge: The parts of an item's knowledge that have been updated since the current KB started. G2 discards the current knowledge of items when you reset or restart the current KB, or when you save the current KB to a KB file. *Contrast with* permanent knowledge.

current language: An existing language translation item (or the symbol english) that G2 uses to display the names of menus and menu choice text (and for some languages, a custom Text Editor interface) in the G2 developer's environment. *Contrast with* default language.

current layer: In the Icon Editor, the layer that is being edited. The current layer is indicated by a heavy border.

current scale: For a workspace, the scale at which G2 displays that workspace, which is a factor of its default scale. By specifying a new current scale, you can interactively or programmatically change the displayed size of a workspace. *Contrast with* default scale.

current task queue: The internal queue that the scheduler maintains for tasks currently eligible for execution. After G2 performs a task, the scheduler removes it from the current task queue.

Cursor Indicator: In the Icon Editor, a display that shows the current cursor position as x, y coordinates.

cyclic dependency: In a module hierarchy, the condition in which a module M is directly required by another module that module M directly or indirectly requires.

D

data interface object: The common term for a g2-to-g2-data-interface object.

data point: For variables keeping history, the value of the variable and its corresponding collection times.

data seeking: The attempt to obtain a new current value for a variable.

data server: The source of values for a variable, or the destination where G2 can send data with the set action.

datatype: *See* type.

dedicated license: One type of G2 license for Telewindows. In a dedicated Telewindows environment, each license is authorized for a specific level of access to the server.

default attribute: For a user-defined class U and its superior classes that define more than one attribute with the same name, the version of the attribute defined by the class closest to class U on U's class inheritance path.

default language: The language translation item (or the symbol english) that G2 uses to display the names of menus and menu choice text (and for some languages, a custom Text Editor interface) in the G2 developer's environment. You set the current KB's default language only with the `-default-language` command-line option. *Contrast with* current language.

default scale: By default, a new workspace's current scale is the *normalized* scale for the G2 process, which G2 determines by calculating the ratio of workspace units per pixel of resolution on your computer's display device.

default value: The value that an attribute other than a variable or parameter value has on instantiation. This value persists until something changes it. Resetting G2 does not change it. *See also* initial value.

dependent variable: A simulated variable whose value is not based on the variable's previous value. A dependent variable does not require an initial value.

depth-first backward chaining: For a set of rules that can conclude a new value for a particular variable, the activity where G2 invokes those rules in order of precedence, according to the depth-first-backward-chaining-precedence attribute of each rule in the set. *Contrast with* breadth-first backward chaining.

direct superior class: The class or classes from which a class inherits directly. A class also inherits indirectly from all the superior classes of its direct superior class or classes.

directly-requires: The relationship between two modules, where the knowledge in one module depends upon the knowledge in the other. Before loading or merging the modularized KB file F that contains module M, G2 first loads each modularized KB file that contains a module that module M directly requires, then loads the modularized KB file F.

display: System-defined classes whose instances are items that show the value of a parameter, variable, or expression. Readout tables, dials, meters, graphs, trend charts, and freeform tables are displays.

duplicate definitions: Duplicate items that are class definitions. *Contrast with* identical definitions.

duplicate methods: Methods that have the same name, and are defined on the same class, but differ in the number of arguments each method takes.

duplicate items: Items that have the same name, whether or not the items have the same type or are functionally equivalent. *Contrast with* identical items.

E

element: A member of a list or array. Elements can be numeric values, truth values, symbols, text strings, or items.

element index: A numeric value specified within square brackets, used as a positional reference to a list or array element.

encapsulation: In object-oriented programming, the technique of including in an object and its context the knowledge of how to perform an operation on that object, rather than in the code that invokes the operation.

error condition: An unexpected discrepancy that occurs while G2 is handling information.

error handler: The portion of an application that responds to error conditions. G2 provides a default error handler, which responds to an error condition by placing a message on the Operator Logbook. You can write a custom error handler in a G2 procedure by coding an on error statement whose block of statements respond to error conditions that arise within that procedure.

evaluation attributes: A special type of attribute that controls the way G2 evaluates expressions. Evaluation attributes are currently used only in freeform tables.

event: A system-defined occurrence. The antecedent of a whenever rule can refer to events.

event detection: G2's activity of invoking certain rules after detecting an event.

event log format: A GFI file format in which data is stored in rows. Each row contains a time-stamp, a variable or parameter name, and a value.

event-driven: Acting in response to events. Forward chaining is one kind of event-driven processing, where G2 invokes certain rules after detecting new knowledge.

exceptional float values: Values of type float that represent the values negative infinity (-Inf), positive infinity (+Inf), and not a number (NaN). Using these values in an arithmetic expression produces another exceptional float value.

expiration time: The time when the value of a variable will expire. This is calculated for all values of variables and displayed in the attribute table.

explicit time-stamp: A GFI time-stamp format in which a date and time are explicitly listed.

expression: A phrase that G2 evaluates to produce a value or a reference to an item.

extensible class: A class that can be the direct superior class of a user-defined class.

extent: The visible, rectangular portion of a workspace. Also, the visible, rectangular region within which G2 displays an item's representation.

external interface: An interface between G2 and external systems.

external variable: A kind of object in the external system to which data is sent or from which a GSI (G2 Gateway) variable receives data.

F

filter expression: Identifies a set of items to which an Inspect command applies.

floating license: One type of G2 license for Gensym's Telewindows product. Floating Telewindows licenses are G2-server based; that is, the G2 license includes a finite number of floating Telewindows connections.

focal class: The class of items to which one or more generic rules apply.

focusing: The technique of invoking multiple generic rules to apply to a particular item or set of items.

foreground-color color attribute: A color attribute for a workspace. For the items on a workspace, the workspace's foreground-color can determine the color value of color attributes for items with system-defined default settings.

foreign function: A function written in C or C++ code that a KB can access as if it were a local function.

foreign function interface: An external interface that allows you to call C and C++ functions from within G2, just as you would any G2 function.

foreign image: An executable file, external to G2, that contains the foreign functions to call from a KB.

formatting attributes: Special attributes that determine the visual aspects of certain items, such as charts and freeform tables.

formula: An equation that provides values for variables or parameters.

forward chaining: An inferencing technique related to event detection, in which G2 invokes certain rules whose antecedent refers to a variable or parameter that has received a new value. *Contrast with* backward chaining.

foundation class: Any extensible system-defined class except a mixin.

free text: A type of text item called to label various items in your KB. Free text lets you label your KB informatively and attractively.

future task queue: The internal queue that G2 maintains for tasks that are eligible for execution at a future point in time.

fuzzy truth value: A truth value, in the range of -1.0 true to +1.0 true, that indicates a degree of certainty in the truth of a condition, assertion, or comparison. For example, +0.9 true is a fuzzy truth value that indicates a high degree of certainty in the truth of a particular comparison.

G

G2 clock: The internal clock by which G2 tracks time.

G2 Gateway standard interface (GSI): A Gensym product that supports building applications that interface with G2 in various ways. *See also* GSI message service; GSI variable.

G2 linearization: The algorithm that G2 uses to linearize multiple inheritance.

G2 GUIDE: *See* GUIDE.

G2 Standard Interface: *See* G2 Gateway.

G2-meter class: A class that inherits from quantitative-variable and the mixin g2-meter-data-service. You can use such a class to instantiate G2-meters, and use these to measure how G2 uses time and memory.

G2-to-G2 variable: For passing values between two G2s, a variable subclass that includes the g2-to-g2-data-service mixin class as one of its direct superior classes. Instances of a g2-to-g2-variable include two additional attributes, g2-to-g2-interface-name and remote-g2-expression.

garbage collection: A capability that identifies abandoned storage and reclaims it for reuse.

generic formula: An algebraic formula that G2 can evaluate to calculate a value for a class of variables. In contrast with simulation formulas, generic formulas are evaluated only as a result of data seeking. *Contrast with* specific formula.

generic reference expression: In certain contexts and depending on the accompanying quantifier, refers either to one or to a set of items, attributes, variable or parameter values, or list or array elements. The expression:

... any custom-object connected to my-valve ...

is a generic reference expression.

generic reference qualifier expression: In a generic reference expression, an expression that qualifies the set of items with respect to their system-defined relationships with other items. In this expression:

... any custom-object connected to my-valve ...

the phrase connected to my-valve is a generic reference qualifier expression.

generic rule: A rule that can apply to more than one item. When G2 invokes a generic rule, it creates one rule invocation for each item or value that meets the conditions specified in the antecedent's generic reference expression.

generic simulation formula: An expression that provides simulated values for an attribute of any class of variable. It exists in its own statement box, the same way that a generic formula does.

Gensym character set: The characters that are valid to specify in a symbol or text value in G2. G2 provides facilities for its Text Editor that allow you to enter any character in the Gensym character set. When inputting or outputting symbol and text values, G2 also observes rules for translating those values using the character codes of standard character sets.

gfi-variable: A variable that specifies GFI as its data server. The direct superiors of a gfi-variable class include any G2 variable class, and the gfi-data-service mixin class.

graphic elements: Components that make up any single-color layer in an icon. Examples of graphic elements are: circles, lines, and points.

group: In the Icon Editor, collection of two or more graphical elements on one layer.

GSI: *See* G2 Gateway standard interface.

GSI message server: A user-defined G2 class with gsi-message-service as one of its direct superior classes. A GSI message server permits messages to be sent via the item to a bridge process. *See Also* G2 Gateway standard interface.

GSI variable: A user-defined G2 variable subclass with gsi-interface as one of its direct superior classes. A GSI variable must specify a valid GSI interface object. *See Also* G2 Gateway standard interface.

GUIDE: A Gensym product, also known as G2 GUIDE. A knowledge base that allows you interactively to create the user-interface components for a G2-based application. Objects created using GUIDE are a permanent part of your knowledge base.

H

hash table: An internal data structure that allows G2 to quickly locate, among many instances of a user-defined class, one or more items with an indexed attribute that contains a particular value. G2 automatically creates and maintains one hash table for each indexed attribute declared in any user-defined class in your KB.

heuristic knowledge: Knowledge that is based on experience or observation, but not necessarily verifiable.

hidden attributes: Attributes inherent within an item, but which are not visible in its attribute table. Hidden attributes include:

- attribute-displays
- name-box

hierarchy of classes: An organization of classes into superior and subclasses to allow for inheritance of attributes and other knowledge. Each class inherits the attributes of its superior classes.

history: The past values of a variable or parameter. Each value is stored with the date and collection time.

I

icon: The graphic representation of objects and items of other system-classes with an iconic representation style. In G2, items of many system-defined classes appear as icons. Use the Icon Editor to define the icon for user-defined subclasses of the object class.

icon-color: For items of system-defined classes that have an iconic representation style, this is both the name of a color attribute and an icon region.

Icon Size Indicator: In the Icon Editor, a display that shows the size of the icon in workspace units.

ICP (Intelligent Communications Protocol): Gensym's proprietary communications protocol, which allows G2s, G2 Gateways, and Telewindows to share information and distribute control among one or more G2 processes. ICP is a layer built on top of the TCP/IP networking protocol.

identical definitions: Identical items that are class definitions. *Contrast with* duplicate definitions.

identical items: Items that are the same in every respect whatsoever. The tables of two identical items are indistinguishable. *Contrast with* duplicate items.

image: A bitmap or other graphical image created outside G2. You can use an external image as part of an icon, or as the background of a workspace.

Image Indicator: In the Icon Editor, a display that shows the name of an externally defined image that is included in a layer.

immediate class: In a discussion of a class and its properties, the class that is the subject of the discussion.

immediate drawing: One of two drawing modes. When drawing is immediate, G2 updates the display as soon as some visible change is made to the KB, either interactively or programmatically. Drawing in immediate mode temporarily defers other KB processing until drawing is complete. *Contrast with* scheduled drawing.

implicit local name: An undeclared local name.

independent-for-all-compilations: One of two G2 compilation configurations. For an item whose attributes refer to another item that is declared stable-for-dependent-compilations, this configuration directs G2 *not* to compile the item to take advantage of the other item's stability. *Contrast with* stable-for-dependent-compilations.

indexed attribute: A user-defined attribute whose value G2 retrieves using an internally maintained hash table. G2 can perform efficient searches to locate an item of a user-defined class by a particular value of one of its indexed attributes.

inference engine: The G2 component that monitors events and reasons about changing conditions while invoking rules by means of forward chaining, backward chaining, event detection, focusing, and scanning.

inheritance: An important property of object-oriented development environments. A class inherits the attributes of its superior. Inheritance facilitates rapid development, eliminates redundancy in an application, and builds reusable application components.

inherited attributes: Attributes that a class has by inheriting them, rather than by defining them in the class's definition.

initial value: The value that a variable or parameter has on instantiation. Resetting G2 restores this value. *See also* default value.

initialization mode: A GFI input mode that typically provides a KB with values read in as soon as the KB begins execution.

input interface object: Used by GFI to read data from an external data file and to provide data service for variables and parameters in a knowledge base.

Inspect facility: A component of the G2 developer's environment that supports searching for items based on their type, class, attributes, and location. When Inspect locates an item, you can conveniently navigate to it or simply modifying its attributes using an attribute table. To open the Inspect facility, select the Inspect choice on the Main Menu.

Inspect workspace: The workspace that displays the results of Inspect commands.

installed system tables: The system tables whose values are in effect for the current KB. In a modularized KB, there are as many sets of system tables as there are modules; however, only one set of system tables is installed at a time.

instance: One of a class of items, for example, pump-1 is instance of the pump class.

instance configuration: A configuration whose scope is determined by the current KB's class hierarchy. Instance configuration statements are contained in the instance-configuration attribute of an object, connection, and message definition. *Contrast with* item configuration.

instantiate: To create an instance of one or more classes.

Intelligent Communications Protocol: *See* ICP.

interface object: An object that GFI uses as an interface between a knowledge base and an external data file.

interpolated value of expression: The interpolated value of a variable or parameter at some time in the past. G2 performs a straight-line interpolation and requires that you keep a history of the values for the parameter or variable.

inverse relation: A relation between the relation object (the second class) and the relation source (the first class), as defined in a relation definition. You specify the name of the inverse relation in a relation definition's inverse-of-relation attribute.

item: An entity in G2 that represents a set of knowledge that has identity and that persists. Each item represents a set of information that is distinct from other information and that you can reference directly or indirectly.

item configuration: A configuration whose scope is determined by the current KB's workspace (or containment) hierarchy. Item configuration statements are contained in the item-configuration attribute of any item. *Contrast with* instance configuration.

item hierarchy: The hierarchy of the item class and its subclasses.

item layer position: For an item upon a workspace, the relative position of an item on top of or beneath other items that are upon the same workspace.

item layering: Whether items upon the same workspace appear on top of or beneath each other.

item passing: For G2-to-G2, or G2 to a bridge process, the ability to pass a copy of any item as a reference, using a network handle.

J

junction block: An object that represents the junction of two or more connections. A connection definition specifies a junction box class.

K

KB: *See* knowledge base.

KB file: The file that G2 writes when you save the current KB. This file contains only ASCII characters, and thus is portable to any G2 of a compatible version or earlier that runs on any supported platform. By default, a KB file's name has the extension `.kb`.

KB snapshot file: The file that G2 writes when the current KB invokes the g2-snapshot system procedure. A KB snapshot file contains a copy of all permanent and transient knowledge (including the run-time information for all executing rules and procedures, the contents of all lists and arrays, and all history values for all variables and parameters) that existed in the current KB at the moment g2-snapshot was invoked.

KB workspace: An item of the kb-workspace class, which is the only kind of workspace in G2 that the current KB can work with programmatically. *Contrast with* workspace.

KL: *See* knowledge library.

knowledge base: A set of items that is either contained in G2's memory (the current KB) or stored in a KB file.

knowledge library: A set of items, such as standard object definitions, that contain information for use in more than one G2-based application. A knowledge library is stored in a KB file whose name has the extension `.kl`. In G2 Version 3.0 and later, modules replace the use of knowledge libraries; however, Gensym distributes knowledge libraries with G2 that contain language-specific extensions to the product.

L

lagged values: For G2-meters, values that are averaged over time, smoothing out transient excursions and thereby clarifying their overall behavior.

last recorded value: The last value that G2 recorded for a variable or parameter. The last recorded value may be current or expired.

layer: In the Icon Editor, a component of an icon that contains one or more graphic elements. Layers can be grouped into regions.

layers pad: The part of the Icon Editor that allows you to view the layers of an icon and work with them individually.

linearization: The process of ordering the ancestors of a multiple inheritance class into a sequential list that can be used to search for inherited definitions and resolve conflicts among them.

literal value: In an expression, a series of characters that literally signifies a value of type integer, float, truth-value, text, or symbol.

local data server: Anything within G2 that supplies computed, inferred, or simulated values. The G2 simulator and the G2 inference engine are examples of local data servers.

local emulator: Within a g2-to-g2-data-interface object, the name of a connection specification that specifies that the interface is connecting to the current G2 process, rather than a remote process.

local name: A non-reserved symbol that represents an item or value in an expression. In rules and procedures, you can use implicit, or undeclared, local names. In procedures, you must declare local names that can receive an assignment. Use a local name to represent an item of any class or a value of any G2 type.

local window: The visible window that is a client of a G2 process. Among computers running the X Windows window manager, a G2 process's local window can be displayed either on the screen of the computer where G2 is running or on another computer's screen. *Contrast with* remote window.

log file: A file to which G2 can optionally write informational messages.

M

main simulation procedure: A special user-defined procedure for the G2 simulator. The simulator executes the main simulation procedure once each simulation cycle.

many-to-many: The cardinality of a relation, where more than one instance of the first class can be related to one or more instances of the second class.

many-to-one: The cardinality of a relation, where more than one instance of the first class can be related to one instance of the second class.

margin: The distance in workspace units between the outermost items upon a workspace and the workspace's border. Each workspace's margins are automatically maintained by G2. Set the default margin for all new workspaces in the Miscellaneous Parameters system table.

memory leak: A loss of usable storage space caused by the application's failure to correctly reclaim the memory occupied by the current KB's transient items.

menu: A list of choices that G2 displays when you click the mouse on the G2 window's background or on an item. Selecting some menu choices causes G2 to display another menu, called a submenu.

Message Board: A named KB workspace that can be the destination of messages from an inform action.

metacharacters: Special characters that let you specify a wildcarded name expression. For example, the asterisk (*) matches zero or more characters.

metacolor: A symbol that indirectly assigns a color value to a color attribute of an item. There are three metacolors: transparent, background, and foreground.

meter: A display item showing the value of an arithmetic expression as a vertical bar along a numeric scale.

meter lag time: A time interval that represents how much data G2 should average when computing values displayed by G2 meters. Set this time interval in the meter-lag-time attribute of the Timing Parameters system table.

method: A specially constructed procedure that performs a generic operation in a class-specific manner.

mixin class: An abstract extensible class that you use in conjunction with a foundation class to define subclasses via multiple inheritance. Mixin classes define sets of attributes that would not be useful alone, but can be used to customize other classes to serve particular purposes.

modularized KB: A KB is modularized as long as the top-level-module attribute of its installed Module Information system table has a value other than the symbol unspecified and the KB's modules are consistently modularized.
A current KB that contains more than one module also contains more than one set of system tables; the Module Information system table for each set represents one module. If the current KB is modularized, the installed Module Information system table is associated with the current KB's top-level module.

If the current KB is consistently modularized, G2 can save each module it contains (and the items associated with that module) to a distinct KB file. If the current KB is not consistently modularized, G2 can only save the current KB as a whole into one KB file. If a stored KB file is consistently modularized, it contains only one set of system tables. If a stored KB file is modularized but not consistently, it contains one set of system tables for each module that it contains.

module: An entity in a knowledge base that G2 automatically associates with a set of system tables and optionally with a set of top-level workspaces. In this set of system tables, the Module Information system table names the module. If the current KB contains modules, it is *modularized*.

module hierarchy: The network of directly requires relationships among a set of modules. A module hierarchy consists of one top-level module, and optionally one or more modules that are directly required by the top-level module, modules that are directly required by those modules, and so on.

module search path: A list of directories that G2 searches to load a modularized KB file that contains a directly required module.

module-map file: An ASCII text file in which each line associates the name of a module in the current KB with either a directory pathname or a fully qualified filename. G2 uses the module-map file when you direct G2 to save a module in the current KB to its own KB file.

mouse click: The act of pressing and releasing any button on the hand-held mouse.

mouse-tracking procedure: A user-defined G2 procedure that can be invoked from a configuration statement that includes the phrase pressing ... on ... start. The procedure exists to respond to a change in the mouse pointer's location *within a particular window*, until the next mouse-click event within that window.

multiple inheritance: In object-oriented programming, the practice of allowing a class to have two or more direct superior classes, and inheriting the properties of them all.

multiple inheritance class: A class that has, or inherits from any class that has, more than one direct superior class.

N

name box: The display containing the name of an item. You can move the name box by dragging it with the mouse, and you can edit an item's name by clicking the mouse over the name box. You can also apply configurations to an item's name box. *Contrast with* attribute display.

natural language prompts: In the Text Editor, the prompts that G2 displays below the edit area to guide you through the statement syntax and available options.

network handle: For item passing, an integer value, obtained by registering any item as a network entity.

no value **condition:** The condition that results when G2 cannot successfully evaluate an expression. If evaluating an expression produces a *no value* condition, G2 signals an error. G2 displays a *no value* condition as the reserved symbol none.

non-KB workspace: A workspace that G2 creates for some specific purpose, but is not saved in the KB. Text Editor workspaces, Operator Logbook pages, and temporary workspaces are examples of non-KB workspaces.

non-menu choices: Operations that G2 performs in response to user actions, including: dragging the mouse, characters entered at the keyboard, and showing, hiding, resizing, and scaling workspaces. Configurations can customize G2's behavior in response to non-menu choices.

non-standard characters: Characters that are not standard characters including: non-alphanumeric characters other than hyphens, underscores, apostrophes, and periods; permanently lowercase letters; and special characters.

normalized scale: The ratio of workspace units per pixel for a G2 process. By default, the current scale of a new workspace is this G2's normalized scale.

O

object: An abstract or concrete thing of interest in your application, such as a product, space station, bottle, event, or workstation. Every object has an attribute table, and may have an icon and connection stubs. Every object is an instance of a class, which is defined through an object definition.

object passing: The ability to pass a copy of an object from one G2 process to another via an external interface. Object passing is accomplished through the use of a remote procedure declaration to specify which attributes of the object to send.

offline license: A fundamental G2 license type providing G2 for a stand-alone system.

online license: A fundamental G2 license type providing the capability to communicate or access other systems.

one-to-many: The cardinality of a relation, where one instance of the first class can be related to any number of instances of the second class.

one-to-one: The cardinality of a relation, where one instance of the first class can be related to, at most, one instance of the second class.

operand: In an expression, a term that participates in an arithmetic, class-qualified, concatenation, logical, or relational operation.

operation: In object-oriented programming, a function or transformation that can be applied to instances, typically in different ways for members of different classes.

operator: In an expression, a reserved symbol or character that specifies a type-specific operation.

Operator Logbook: A special workspace for displaying informational messages and signalling G2 errors. You control the placement and other properties of the logbook workspaces using the Logbook Parameters system table.

output frequency: In GFI, the interval at which to write data to a GFI output file.

output interface object: An object that GFI uses to obtain values from variables and parameters in a knowledge base and write them to an external data file.

overlay file: The output file created after using the Overlay utility, described next.

Overlay utility: A Gensym-provided utility for creating an overlay C source file from a template file as one of the steps in using foreign functions.

P

package preparation: Your activity of removing source code or making workspaces proprietary. Package preparation is typically the last step in preparing a KB for delivery to customers.

package preparation mode: A system-defined user mode that permits making a knowledge base proprietary.

paint mode: The default G2 drawing method.

palette: A menu that presents G2's default colors. Also, a workspace that contains each type of object that you can create in a knowledge base. You can create new objects by cloning from the palette.

permanent item: An item that continues to exist in the current KB after the KB is reset or restarted. When you save the current KB to a file, only the KB's permanent items are stored in the KB file. Items that you create interactively are permanent by default. *Contrast with* transient item.

permanent knowledge: The version of an item's knowledge that persists after you reset the current KB and that are saved when G2 stores the current KB in a KB file. *Contrast with* current knowledge.

permanent-membership list or array: A list or array whose elements remain elements when G2 is reset, provided that the list and elements are permanent, and are saved when the KB is saved, provided that the list is permanent and the elements are both permanent and uniquely identifiable. Permanent membership does not affect the permanence of the list or array itself, or of any items that are members of it.

permanent relation: A relation that persists when G2 is reset, provided that the related items are permanent, and is saved when the KB is saved, provided that the related items are permanent and uniquely identifiable.

platform: A combination of a brand of computer and a brand of operating system. G2 runs on several Unix and Windows platforms.

polymorphism: In object-oriented programming, the technique whereby the same operation means different things, depending on the class of the operand to which it is applied.

port: The place on an item's icon where a connection attaches. You can provide names to ports, portnames, and refer to those names in connection expressions.

precedence (of configurations): G2's rules that determine the order in which it searches among conflicting configurations that pertain to the same item.

primary definition: When class definitions are merged, the definition that the secondary definition is merged into.

primary direct superior: The first class in a list of multiple direct superior classes given in a class definition.

procedure: A list of statements that G2 can execute, either in sequence or concurrently, on zero or more arguments supplied when the procedure is invoked.

procedure invocation: An item that represents the invocation of a procedure. If you set the class-of-procedure invocation attribute of a procedure to procedure-invocation, G2 creates a procedure invocation when the procedure starts and deletes the invocation when the procedure ends.

profile data: Information about the duration and order in which G2 performed an identified set of executable items in the current KB.

proposition: A statement, such as valve-is-broken or tank-is-overflowing, that is either true or false.

proprietary item: An item *or* its object definition that resides upon a proprietary workspace.

proprietary KB: A knowledge base that contains proprietary workspaces that require authorization.

protocol: A specification of the format and content of information sent between one process and another, such as between G2 and Telewindows.

Q

qualified filename: A filename that includes an extension.

qualified name: The name of an attribute prefixed by the name of the class that defines it, or the name of a method prefixed by the name of the class to which it applies. Syntax: *class-name:: attribute-name*, or *class-name:: method-name*. Used in expressions to refer unambiguously to attributes and methods.

quantifier: In a generic reference expression, a reserved symbol that indicates whether the expression produces *one*, *one and only one*, *at least one*, or *any number* of items or values.

R

real-time clock: The time clock kept by the computer on which the G2 process is running. The default value for the scheduler-mode attribute in the Timing Parameters system table of G2 is real-time, indicating that G2 time is passing in relation to true clock time.

region: A named group of one or more icon layers. All the layers in a region have the same color. You can use the change action to change the color of any named region.

Region 1: A block of memory used by G2 to hold items and non-symbolic values.

Region 2: A block of memory used by G2 to hold symbols and related internal data.

Region 3: A block of memory used by G2 to hold any external images used in icons and as the backgrounds of workspaces.

Region Indicator: In the Icon Editor, a display that names the region (if any) to which the current layer belongs.

relation: An actual association of a particular kind between two items in the current KB. *Contrast with* relation definition.

relation definition: The definition of a kind of association between two classes of items. A relation definition has a name, cardinality, a first class, a second class. A relation definition can also be symmetrical. A relation definition can optionally name a kind of relation that is the inverse of this relation definition. *Contrast with* relation.

relation object: The second class in a relation definition.

relation source: The first class in a relation definition.

relative time-stamp: A GFI time-stamp format that consists of a non-negative integer that specifies a number of seconds since the base time of the file.

remote data server: A process external to G2 that supplies G2 with data or that accepts set actions from G2. A remote G2 can be a remote data server, for example, as can a G2 Gateway interface. *See Also* G2 Gateway standard interface.

remote window: The visible window that is a client of a Telewindows process connected to a running G2 process. *Contrast with* local window.

rendezvous failure: The inability to restore membership in a permanent-membership list or array, or participation in a saved permanent relation, when a saved KB is reloaded.

representation: The appearance of an item, which is of a particular representation style.

representation style: One of several forms in which class of items appear. Each system-defined class in G2 has a representation style. A class's representation style determines whether items of that class appear as icons, text boxes, displays, connections, and so on.

request mode: A GFI input mode in which GFI provides a KB with data when the KB requests it via data seeking.

required module: A module that contains items required by another module.

reserved classes: System-defined classes in G2 that can be inherited only by system-defined subclasses. An example of a reserved class is variable.

reserved symbol: A symbol that cannot serve as a user-defined name in G2. In the Text Editor, the prompt any unreserved-symbol indicates where you must enter a user-defined name.

reserved words: Symbols that cannot be used as names or symbol values in expressions.

root class: The topmost class in a class hierarchy, or the topmost class of interest in a subset of a class hierarchy.

rule: An item whose text expresses a programmatic response to a set of conditions. A rule's text contains a two-part statement, which can take one of several forms; for example: when fire-alarm is sounding then invoke fire-safety rules. G2 invokes rules using several different mechanisms.

rule invocation: A executing copy of a rule. When G2 invokes a specific rule, G2 creates one rule invocation. When G2 invokes a generic rule, G2 creates one rule invocation for each item or value that meets the conditions specified in the generic reference expression in the rule's antecedent. Consequently, several invocations of the same generic rule might execute simultaneously.

run-state: Whether the current KB is running, paused, or reset. Changing the current KB's run-state affects the KB's knowledge.

run-time validation: Instructions that G2 includes, by default, in a compiled attribute. When the compiled attribute is invoked, evaluated, or referenced, the attribute's run-time validation instructions cause G2 to verify whether the attribute's own assumptions about the referenced item–that is, its name, its class, and so on–are still true.

S

scan interval: An attribute that gives the rate at which G2 should invoke a rule. For many applications it is more efficient to invoke rules through forward chaining than through scanning.

scanning: For a rule, the process of invoking the rule at some regular interval specified in the scan-interval attribute. Use scanning as a means of monitoring conditions in an application.

scheduled drawing: One of two drawing modes. Scheduled drawing lets you control *when* KB drawing occurs. You can control KB drawing by calling the g2-work-on-drawing system procedure from within any procedure where you want drawing to occur. *Contrast with* immediate drawing.

scheduler: The part of G2 that directs all other processing. The scheduler works in clock ticks; within a single clock tick it does the following: schedules new tasks, runs tasks that have been scheduled (running a rule, for example), services external data servers, services the user interfaces, and services the simulator.

schematic: A picture of the application; consists of objects and their connections.

scope: For a configuration, the set of items to which the configuration pertains.

scrapbook workspace: A special workspace used for holding pieces of text used for insertion in the Text Editor.

secondary definition: When class definitions are merged, the definition that is merged into the primary definition.

secondary direct superiors: The second and later classes (if any) specified in a list of direct superior classes in a class definition. The first class in the list is called the *primary superior class*. All remaining classes in the list are *secondary superior classes* with a declining order of precedence.

secure G2: A G2 process that requires its users to log in. If the installed G2 product has a valid license, you run a G2 process that is secure by creating an authorization file (the `g2.ok` file) and including it in entries for this G2's users.

scrollable text editor: One of two interfaces to the G2 Text Editor, for editing attributes that typically have large amounts of text, such as procedures and methods. The scrollable editor allows you to scroll the contents of the editor's edit area and to enter newline characters more easily.

sequential execution: One of two methods of computational execution, the process of executing commands one after another, using the in order statement. Each command completes before G2 continues to process the next. *Contrast with* concurrent execution.

shallow simulation: A collection of simple heuristics that model observed behavior.

shrink wrapping: For a workspace, the operation of decreasing its extent, yet with its borders still outside those items upon the workspace that are farthest apart vertically and horizontally.

simple attribute: An attribute that is neither given by a parameter or variable, nor by an instance of a class. A simple attribute can have values of any type. No type-checking occurs.

simulation formula: For a variable, a formula entered in the simulation-formula attribute of its simulation subtable, which the simulator uses to calculate a value.

simulation procedure: A procedure that the simulator calls either when the application is initialized or once each simulation cycle. Although there is no separate subclass of procedure used specifically for simulation, a simulation procedure behaves in a slightly different manner from other procedures.

simultaneous execution: *See* concurrent execution.

single inheritance: In object-oriented programming, the practice of allowing a class to have only one direct superior class.

single inheritance class: A class that neither has, nor inherits any class that has, more than one parent.

source G2: For a remote procedure call, the G2 from which a remote procedure call originates. *Contrast with* target G2.

specific formula: For a variable, a formula entered in its formula attribute to calculate a value for a specific variable. G2 evaluates a specific formula as a result of data seeking. A specific formula overrides any generic formulas for the variable, and requires that the data server be the inference engine. *Contrast with* generic formula.

specific type: An actual value must be of a specific type. G2 offers these specific types: integer, float, symbol, text, and truth-value. *Contrast with* abstract type *and* composite type.

spreadsheet format: A GFI file format in which data is stored in rows and columns that resemble the layout of a spreadsheet.

stable-for-dependent-compilations: One of two G2 compilation configurations. This configuration declares that an item is not subject to further change, and allows G2 to compile more efficiently other items that refer to the configured item. *Contrast with* independent-for-all-compilations.

standard characters: Letters, digits, hyphens, underscores, apostrophes, and periods.

standard output messages: A report of the initial allocations for G2's memory regions and network port numbers. As a new G2 process starts up, it displays standard output messages. A G2 process also produces standard output messages when it must attempt to allocate additional memory allocations and when it detects an internal error.

state variable: A variable that depends on its own previous value. In G2, state variables are explicitly created only within simulation formulas.

statement: A rule, a regular formula, a simulation formula, a user-defined function, or an executable instruction in the body of a procedure.

status: One of several pieces of knowledge that an item contains, about its ability to participate in the current KB's processing. An item's statuses are permanent/transient, active/inactive, enabled/disabled, and OK/bad/incomplete. An item's statuses are distinct from its attributes, though G2 reports item statuses in an item's notes attribute.

Status Indicator: In the Icon Editor, a display that shows various message that describe the current state of the editor or prompt for user input.

stubs: An extendable connection segment at the edge of an object. Object definitions define the number, appearance, and placement of connection stubs on a class of objects. You can also add and delete stubs for individual objects.

subattribute: An attribute that is part of another attribute's value. For example, History-keeping-spec of a variable or parameter has three subattributes:

- maximum-number-of-data-points
- maximum-age-between-data-points

- minimum-interval-between-data-points

subattribute reference: The expression to reference any subattribute of an attribute, which may itself consist of other subattributes, such as:

the minimum-interval-between-data-points of the history-keeping-spec of x

subclass: A class subordinate to another in the hierarchy of classes. A class can have any number of subclasses.

subexpression: An expression that is a term of another expression.

subobject: The object contained in the attribute of an item.

subtable: The attribute table of any subobject.

subworkspace: A workspace that is subordinate to an item. Only items of certain classes can have a subworkspace, and an item of the proper class can have only one subworkspace. A subworkspace can be affected by the activatable-subwork-space configuration property on its superior item. (*See* activatable subworkspace.) *Contrast with* top-level workspace.

superior class: A class that is at a higher level than another in the hierarchy of classes. Classes inherit attributes from their superiors.

symmetric relation: A relation definition whose relation-is-symmetric attribute contains the symbol yes. This means that concluding a relation of this kind also concludes an inverse relation of the same name and kind.

system-defined attribute: An attribute provided by one of G2's system-defined classes. *Contrast with* user-defined attribute.

system-defined class: A class that G2 provides by default, and which is a member of the default class hierarchy. For example, object, variable, procedure, and rule are system-defined classes. You cannot remove a system-defined class from G2. *Contrast with* user-defined class.

T

table: *See* attribute table.

tabular function: The common term for an item of the tabular-function-of-1-arg class.

target G2: For a remote procedure call, the G2 that executes the procedure. *Contrast with* source G2.

task: In scheduling, the smallest unit of activity that G2 performs.

TCP/IP: A transport layer protocol for communications between computers. TCP/IP is one of two communication protocols supported by G2 and related Gensym products.

Telewindows: A Gensym product. When running, allows a user to connect to a running G2 process and view a window that displays the contents of the G2's current KB. Multiple Telewindows users can access one G2.

template file: A file external to G2 that declares the arguments and return values of each foreign function in a foreign image.

term: A syntactic element of an expression. In an expression, each term is either a literal, an operator, or another expression.

Text Editor workspace: The workspace displayed by the Text Editor for editing text.

text inserter: A type of free text item. When you click on the text of a text inserter while a text-edit workspace is active, G2 enters some or all of the text into the editor at the cursor. There are four types of user-created text inserters: text, word, character, and character sequence.

text stripping: The process of removing source code from items such as procedures and rules, as part of making a KB proprietary for customer distribution.

text-color color attribute: Determines the color of the text that appears in an item whose class has the text box representation style.

time-stamp: In an expression, a value that represents an instant in time. In a GFI file, there are two types of time-stamps: *relative time-stamp* and *explicit time-stamp*.

title block: The workspace that G2 displays at start-up in its local window. The title block reports: the G2 version, the platform, network ID (or host name) of this computer, and the TCP/IP port number on which this process listens for network connections.

top-level module: The module at the top of the current KB's module hierarchy. This module is not directly required by any other module in the current KB. A top-level module can directly require other modules.

top-level workspace: A workspace that has no superior item. A KB can contain any number of top-level workspaces. You specify an association between a top-level workspace and a module in the workspace's module-assignment attribute. *Contrast with* subworkspace.

transaction: A sequence of KB processing in which the set of values in use must remain valid, unchanged, and consistent with respect to each other.

transaction scope: The set of G2 actions and statements that correspond to a transaction.

transient item: An item that does not continue to exist in the current KB after the KB is reset or restarted. When you save the current KB to a file, the KB's transient items are *not* stored in the KB file. Items created with the create action are transient by default. *Contrast with* permanent item.

two-dimensional array: A nested array of arrays. You can create a g2-array or an item-array, whose elements consist of arrays.

type: A value's type determines the set of valid operations in which it can participate. G2 offers these types: item-or-value, item, value, quantity, integer, float, symbol, text, truth-value, sequence, and structure. Each type is abstract, specific, or composite. *See* abstract type, specific type, *and* composite type.

U

unit of measure: A unit such as meters or seconds suffixed to a quantity. G2 provides system-defined units of measure, and you can define additional units of measure as needed.

unmodularized: A KB file or the current KB is unmodularized when it contains no modules, or when it contains modules that are not consistently modularized. *Contrast with* consistently modularized.

unqualified filename: A filename without an extension.

user menu choice: An item that associates an action and menu choice text with a class. You can select a valid user menu choice from the menu of an eligible (sufficiently configured) item of the specified class or any of its subclasses. You can select a user menu choice only while the current KB is running and while the expression specified in the user menu choice item's condition attribute produces the truth-value true.

user mode: A non-reserved symbol that represents a category of usage or level of access to the current KB's knowledge. Declare a user mode simply by naming it in at least one configuration statement in at least one item in your KB. A G2 authorization file associates each authorized user name with a user mode.

user modes clause: A syntactic element of a configuration statement that names a user mode.

user-defined attribute: For a user-defined class, an attribute that is defined in the class-specific-attribute attribute of this class's definition item, or in the same attribute in the definition of some user-defined superior class of this class. *Contrast with* system-defined attribute.

user-defined class: The class that is defined by an object definition, connection definition, or message definition. Create a user-defined class to represent a template for set of knowledge not provided by instances of G2's system-defined classes. *Contrast with* system-defined class.

user-defined function: The result of creating and specifying a function definition item. A user-defined function names an operation that takes zero, one, or more than one item or value as arguments, and returns the item or value that results from G2's using the arguments to evaluate the function's specified expression.

V

validity interval: For the value of a variable, the length of time after its collection time that G2 considers the value to be valid. G2 uses the validity interval to compute the expiration time of a variable's value.

value: In the G2 type hierarchy, the value type includes all types except item-or-value and item. When used as an argument, *value* and *value-expression* include only the simple types (integer, float, symbol, text, truth-value), not the composite types (sequence and structure).

value passing: The process of obtaining a value for a g2-to-g2-variable from a remote G2 process through the use of an external interface.

variable failure: The result of an attempt to obtain a new current value for a variable, where the variable does not receive a value within its specified time-out period. The variable can also fail to obtain a new current value after G2 has used all appropriate means to obtain the value and has failed to do so.

W

wait state: For a procedure, the occurrence of a suspension of the procedure's execution, during which other KB processing can occur. A wait state occurs, for example, when a procedure executes a collect data statement or allow other processing statement.

wakeup: For the G2 scheduler, the activity of resuming execution of a task after a previously requested value for a variable arrives from the variable's data server.

warmboot: The activity of resuming execution of the current KB after loading its knowledge from a snapshot file.

wildcard: When specifying an operating system directory to search or a KB file to load or merge, use a wildcard, or reserved character, to signify one or more other characters in a specified directory name and/or filename. G2 uses wildcard characters and conventions that are independent of the operating systems of the platforms on which G2 runs.

window: A display on a computer screen that is a client of a process, such as G2 or Telewindows. A window's user interface (for example, whether and how it can be moved, resized, or iconized) is determined by the window manager software in use on your computer.

workspace: An item in G2 that organizes a set of items within an abstract, three-dimensional region. A workspace appears in a G2 window or Telewindows window as a bounded rectangle. A workspace both contains items and arranges them schematically with respect to each other. A workspace can be the inferior item of another item; thus, your KB can contain a hierarchy of workspaces. *Contrast with* KB workspace.

workspace hierarchy: The containment hierarchy in the current KB: the KB's top-level workspaces, the items upon those workspaces, the items contained in the attributes of those items, the subworkspaces of those items, the items upon those subworkspaces, and so on.

workspace origin: The location with coordinates (0, 0) in workspace units. The origin of a new workspace is its center. After placing items upon a workspace, moving them, and shrink wrapping the workspace, the workspace's origin might no longer be within the workspace's displayed portion, or *extent*.

workspace unit: A unit of measure within a workspace that is equivalent to one pixel (one screen dot) when the workspace is shown at full size, which is proportionately larger or smaller as the workspace's scale is increased or decreased, respectively.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

O

P

Q

R

S

T

U

V

W

X

Y