G2 Foundation Resources

User's Guide

Version 2015

G2 Foundation Resources User's Guide, Version 2015

December 2015

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: DOC021-1200

Contents Summary

Contents

Preface

Introduces this document and the conventions that it uses.

About this Guide

This guide contains complete information about G2 Foundation Resources (GFR) and shows you how to use the module at any supported level. This guide:

- Introduces GFR and describes the functions, classes, and associated capabilities that it provides.
- Describes the GFR application programmer's interface (API) and shows you how to use GFR functions programmatically.
- Lists all GFR API functions in a reference dictionary.

Audience

This guide assumes you are generally familiar with G2 terminology and practices, but does not require a thorough understanding of G2. If you encounter G2 terms or concepts that you do not understand, see the *G2 Reference Manual*.

This guide assumes you have a general familiarity with designing and creating user interfaces, objects, and text messages. It does not assume an understanding of user interface internals of any kind.

Organization

This guide contains eight chapters in four parts:

	Title	Description
Part I	**What is GFR?**	
1	Overview of G2 Foundation Resources	Introduces the capabilities of the G2 Foundation Resources module.
Part II	**Using GFR**	
2	Managing Modules	Discusses how GFR manages KBs that consist of multiple modules.
3	Handling Errors and Communications	Describes the model of communications handling used in GFR.
4	Localizing KBs	Describes the localization facilities of GFR.
5	Managing Palettes	Discusses how to use the palette creation utilities provided by GFR.
6	The Universal Unique ID System	Discusses how GFR generates and manages universal unique identifiers (UUIDs).
7	Additional GFR Utilities	Discusses file parsing and other GFR utilities.
Part III	**API Procedures and Functions**	
8	Application Programmer's Interface	Describes the Application Programmer's Interface (API) to the GFR module.

A Note About the API

The GFR API, as described in this guide, is not expected to change significantly in future releases, but exceptions may occur. A detailed description of any changes will accompany the GFR release that includes them.

Therefore, it is essential that you use GFR exclusively through its API, as described in this guide. If you bypass the API, you cannot rely on your code to work in the future, since GFR may change, or in the present, because the code may not correctly manage the internal operations of GFR.

If GFR does not seem to provide the capabilities that you need, contact Gensym Customer Support at 1-781-265-7301 (Americas) or +31-71-5682622 (EMEA) for further information.

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
text-string	Return values of G2 procedures and methods in syntax descriptions

Convention Examples	Description
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

What is GFR?

Chapter 1: Overview of G2 Foundation Resources

Introduces the capabilities of the G2 Foundation Resources module.

1

Overview of G2 Foundation Resources

Introduces the capabilities of the G2 Foundation Resources module.

Introduction

G2 Foundation Resources (GFR) is a G2 knowledge base (KB) module that provides tools for implementing multiple-module KBs. By adopting GFR, the modules and applications you develop will gain several advantages:

- Increased compatibility and consistency with modules written by other authors.
- Ability to combine different sets of existing modules to form new applications, without introducing conflicts.

- Reduction of the development time required to deploy your applications.
- Increased reliability and quality stemming from the use of well-tested and supported software, written by experienced G2 developers.

While GFR does not represent a comprehensive solution to all issues in modular KB development, GFR provides many basic module integration issues and will continue to expand and address more issues faced by KB designers and developers. For this reason, Gensym now recommends that all G2 KBs include GFR, directly or indirectly, as a required module, and that users adopt the GFR standards into their own development practices.

The main features of GFR include:

- Module management utilities, including version control, coordinated startup and warmboot, and a way to define user-customizable module settings.
- Communications and error handling.
- Localization utilities.
- Palette management.
- Object identification.
- Text parsing and other miscellaneous utilities.

Module Management Utilities

GFR is a set of related module management utilities that help organize and coordinate the actions of multiple-module KBs. When you are developing or using multiple-module KBs, especially during team development, you may find it difficult to document and enforce version dependencies between modules.

GFR's utilities determine and validate the version dependencies between modules, and between the modules and G2.

The version checking system warns the user if an incompatible set of modules is loaded and also dispatches user-defined upgrade procedures when a new version of a supporting module is loaded.

When starting a multiple-module KB, initialization activities must occur in a defined sequence. GFR organizes startup activities so the initialization of modules is consistent with the module hierarchy. GFR also provides organized startup when modules are merged into a running G2 and when snapshot files are loaded.

Modules often feature user-customizable settings that control certain aspects of the module's behavior or define the resources to be used by the module. In modular KBs, competitions for the use of these settings may arise, for example, when two modules attempt to customize the settings of a common required module. GFR offers a standard approach for defining module settings, the classes

that represent preferences for these settings, and a method of resolving competitions between modules for these settings, should they occur.

For details, see Managing Modules.

Communications and Error Handling

One of the most important shareable resources in a multiple-module KB is the user interface. If every module uses the interface as it sees fit, the end result can be chaotic. The GFR communications and error handling model enables you to "soft-code" the handling of user communications such as error messages, alerts, and confirm dialogs, so that the end user can control and customize the user interface according to his or her unique specifications. As a module developer, if you structure all user communications according to this model, you will greatly enhance the flexibility and utility of your module.

For details, see Handling Errors and Communications.

Localization

GFR provides the basic tools for engineering KBs so that the user interface is easily translated into other languages. While G2 enables you to translate the texts appearing on its menus, it does not provide a structured approach for translating free texts and messages used in an application's user interface.

GFR contains classes for storing and organizing language-specific texts, a simple application programmers interface (API) to retrieve language-specific text, and classes of localizable messages. GFR's text substitution facility enables you to implement localizable messages that contain substrings whose values are determined at run time.

For details, see Localizing KBs.

Palette Management

GFR enables you to implement palettes, which are workspaces with click-to-clone objects. Palettes are a convenient way to create instances of the objects the icon represents.

GFR palettes optionally provide pop-up "bubble help" that describes the object currently under the mouse. You can make GFR palettes proprietary to prevent the end user from making unauthorized changes. GFR palettes also automatically protect against the user re-depositing the cloned item back onto the palette.

For details, see Managing Palettes.

Unique ID Facility

Unique identifiers are often required when items participate in persistent data structures involving other items. The ID is the mechanism for identifying the parts of the structure, similar to a persistent pointer. Beginning with Version 5.0, G2 provided a mixin class, named unique-identifier, which gives each instance of the class with an identification string. The format of this ID string assures uniqueness, to a very high level of probability.

Beginning with Version 4.1, GFR provided a similar facility implemented at the KB level. This facility is separate from G2's unique identification facility. There are differences in how and when IDs are assigned, when objects are created, how objects are initialized, and how they can be used in permanent data structures.

Beginning with Version 6.1, G2 assigns a uuid attribute to every item when the item is created. In addition, you can generate a UUID, using the g2-make-uuid system procedure. Therefore, the GFR unique identifier facility is now redundant.

For details, see The Universal Unique ID System.

Text Parsing and Other Utilities

G2 provides basic file management utilities, such as the ability to open and close files, and to read and write lines of text. When, using these utilities, you read a file, you are forced to write parsing routines to separate the resulting text string into its lexical elements.

While GFR does not address this problem directly, it does provide a utility that converts the contents of a file into a list containing G2 value types (floats, integers, symbols, texts, and truth-values). Once a file has been converted into a value list, further text parsing is usually unnecessary.

For details of this and a small set of unrelated utilities provided by GFR, see Additional GFR Utilities.

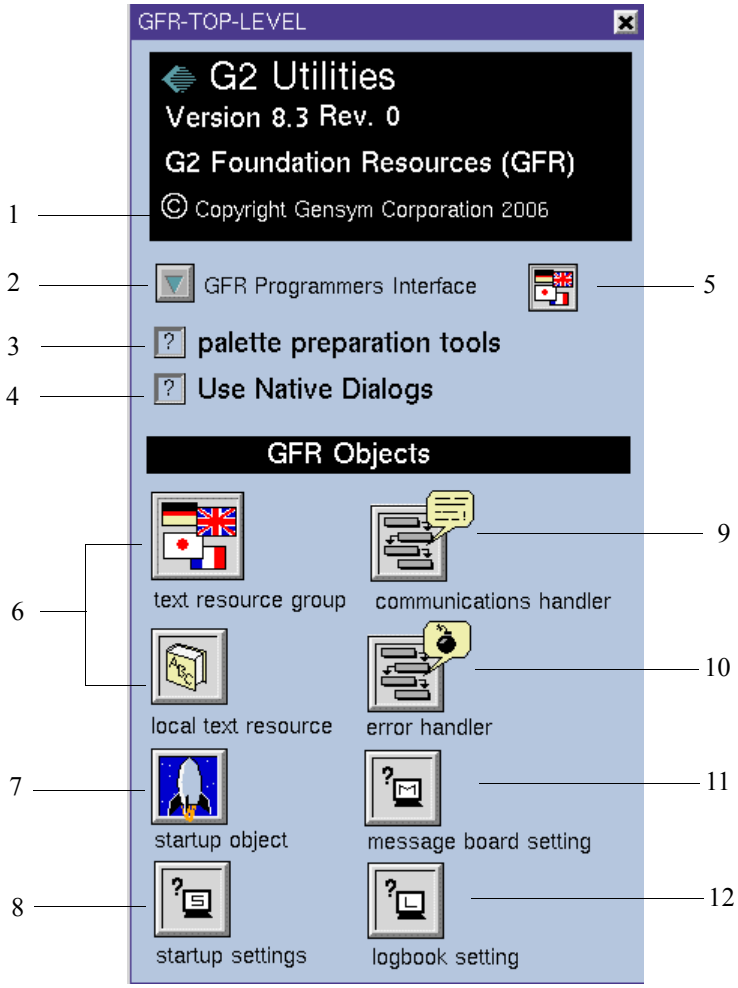
Loading and Running GFR

To load and run GFR:

1. Load the file named *gfr.kb* located in the *utils* subdirectory in the *kbs* directory under the *g2* directory.
2. Start G2.

   GFR does not operate when G2 is reset or paused.
3. Display the top-level workspace named gfr-top-level.

Here is the GFR top-level workspace:



The main features of the workspace are:

1 Click the copyright symbol to display the module version information. For details, see Module Version Control.

2 Click this button to display the Application Programmer's Interface (API) for GFR.

3 This checkbox activates the menu choices related to palette preparation. Normally, these menu choices are not shown. Click this checkbox when you are constructing palettes.

4 This checkbox indicates whether to use native dialogs when viewing dialogs for communication objects through Telewindows.

5 Click this icon to access the English-language text resource for GFR. If you want to translate GFR into a language other than English, clone the English-

language text resource and translate the English-language text strings of cloned item. Place the cloned text resource in a module other than GFR. For details, see Storing Texts in Resource Objects.

6 These icons are palette objects for the language localization facility. The large icon represents a group of localizable texts and the book icon represents an individual language resource.

7 Use this object to start up modules. Each module whose startup is managed by GFR must contain one startup object.

8 Use startup settings to specify the level of user interactions during startup.

9 The communication handler is a subclass of a procedure that enables you to customize how communications such as alert dialogs are presented to the user. To add a custom communications handler, clone this object and place it in your module.

10 The error handler is a subclass of a procedure that enables you to customize how errors are presented to the user. To add a custom error handler, clone this object and place it in your module.

11 Use the message board handler setting to override the normal message board behavior. Clone an instance of this item and place it in the module that wants to override the normal message board behavior. When you load or merge an application, GFR registers the message board handler specified in the highest module in the module hierarchy.

12 The logbook handler icon performs the same role as the message board handler icon. Assign a custom logbook handler, clone this icon and place the resulting instance in another module.

Note To clone these objects, select the item you wish to clone with a full mouse click. A second click transfers the item from the mouse to the desired workspace destination.

Application Programmer's Interface

All programmatic interactions with GFR take place through a small set of specially-designated "public interface" procedures, functions, classes, and attributes. These items are referred to as the Application Programmer's Interface (API) and are described in Part III, API Procedures and Functions.

Because G2 does not have a mechanism to distinguish public and private classes and attributes, GFR uses a naming convention to help you differentiate the API from the internals of GFR.

The convention is the following statement:

> *Items and attributes whose names begin with the prefix _gfr- are private. You may not refer to, alter, or subclass any item or attribute whose name begins with _gfr-.*

Caution Using the Inspect Facility, you may be able to navigate to or view private items and attributes. However, you must never refer to these items programmatically or edit them manually.

When building your own KBs, Gensym recommends that all users adopt the naming convention of using a leading underscore to distinguish the internals of a module. When you follow this convention, users of your module understand your intentions.

Following this convention also helps you, as a KB designer, to focus on the interface your module presents to other modules. A well-defined interface allows you to make improvements and changes to the module, without the risk of introducing incompatibilities, if you maintain the API unchanged. Without formal module interfaces, you cannot realize the benefits of modularity.

Only items whose names begin with *gfr-* are part of the public interface to GFR.

The API to GFR consists of G2 procedures and methods, which you access by writing your own G2 procedures that call the API procedures. In some cases, you may create subclasses of GFR classes and write methods on your subclasses. For more information about G2 procedures, subclassing and methods, refer to the *G2 Reference Manual*.

Part II

Using GFR

Chapter 2: Managing Modules

Discusses how GFR manages KBs that consist of multiple modules.

Chapter 3: Handling Errors and Communications

Describes the model of communications handling used in GFR.

Chapter 4: Localizing KBs

Describes the localization facilities of GFR.

Chapter 5: Managing Palettes

Discusses how to use the palette creation utilities provided by GFR.

Chapter 6: The Universal Unique ID System

Discusses how GFR generates and manages universal unique identifiers (UUIDs).

Chapter 7: Additional GFR Utilities

Discusses file parsing and other GFR utilities.

2

Managing Modules

Discusses how GFR manages KBs that consist of multiple modules.

Introduction

When you create a KB of any significant size, Gensym recommends that you break the KB into multiple modules to organize your development hierarchically. Ideally, each module you create has a clearly-defined function, as well as a clean interface with the user and with other modules.

By creating module hierarchies, you can build modules upon other modules to achieve sophisticated functionalities. The hierarchy defines the dependencies between modules. The higher-level modules depend on the modules below, and those below provide services to higher-level modules.

When a module is operating in a hierarchical environment, you must coordinate its interaction with the other modules in the hierarchy. The interactions that you must coordinate include version control, organization of startup activities, and sharing and allocation of common resources.

GFR provides the following tools for managing multiple-module KBs:

- Version control

  GFR's version control system provides a mechanism for determining and validating the version dependencies between modules, and between the modules and G2. The version checking system warns the user if an incompatible set of modules is loaded. It also dispatches user-defined upgrade procedures when a new version of a module is loaded. For details, see Module Version Control.

- Startup facility

  The GFR startup facility assures that module initialization activities proceed in an organized fashion from lower-level modules up the hierarchy to higher-level modules. This facility also provides for organized dispatch to multiple warmboot procedures following the loading of your KB from a snapshot file. For details, see Using Module Startup Objects.

- Module setting utility

  This utility provides module settings that are analogous to G2 System Tables. Each module can define user-customizable operational parameters, called *module settings*. Its client modules (higher-level modules requiring that module) can determine these settings. At startup, GFR determines which module settings should be active and installs those settings in the appropriate modules. For details, see Managing User-Settable Parameters for Modules.

In addition, GFR includes a set of API procedures that provide information about the module hierarchy, and a standard technique for storing objects in other modules. For details see Application Programmer's Interface.

Module Version Control

GFR provides a standard method for recording version information in a KB, using objects of the class gfr-version-information-object. The GFR palette's version information object is the copyright symbol. GFR actively employs version information objects to:

- Provide internal documentation.
- Assure consistency between the modules comprising the KB and between these modules and G2.
- Launch upgrade procedures.

Note Each module dependent on GFR must contain *exactly one* version information object.

If a module that requires GFR does not contain a version information object, GFR automatically creates one and places it in the module's public bin. Although you will find a version information object on GFR's palette, in most cases, you do not need to clone this icon from the GFR palette because GFR will have automatically created a version information object. You can retrieve and move this version information object and manually edit the attributes.

To retrieve a version information object created by GFR:

1. Type the following in Inspect:

   show on a workspace every gfr-version-information-object

2. Choose the go to original user menu choice in the appropriate version information object's pop-up menu.

The version information object is located on the subworkspace of *module_name*-public-bin, where *module_name* is the name of your module. You can move the version information object to any other workspace in the same module by using operate on area.

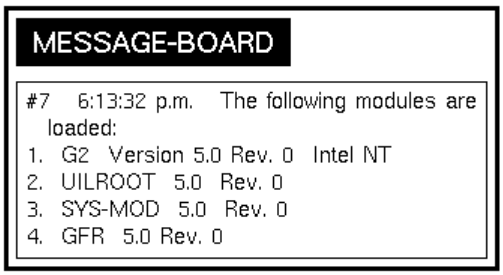
The icon for the version information object is the copyright symbol (©). You can use the version information object to display copyright information in addition to the object's version control function.

Tip If your application has a "nameplate" workspace, Gensym recommends that the version information object be placed there or in another location easily accessible to the user.

To retrieve version information for G2 and all loaded modules:

1. Display the pop-up menu for any version information object.
2. Choose the show all versions user menu choice.

The version information for G2 and all loaded modules appears on the Message Board, for example:



Version Information Object Attributes

The following is a list of attributes of the version information object:

Attribute	Description
gfr-module-name	The name of the module described by this version information object, which must be the same as the module containing the object.
Allowable values:	Any symbol naming a G2 module
Default value:	unspecified
gfr-package-name	A text describing the bundling of modules into a package, for example, GFR is part of the G2 utilities package.
Allowable values:	Any text
Default value:	""
gfr-version-description	A text describing the current version, for example, "Version 1.0 Rev. 2".
Allowable values:	Any text
Default value:	""
gfr-minimum-g2-version	A text describing the minimum required version of G2 needed to run this module, for example, "Version 2015 Rev. 0".
Allowable values:	A text in the format returned from g2-get-software-version
Default value:	""

Attribute	Description
gfr-version-number	A release sequence number that represents how the current version of the module fits into sequence with other versions of the same module.
Allowable values:	A non-negative integer
Default value:	0
gfr-oldest-compatible-version	The release sequence number of the oldest version of this module that can support applications developed using the current version of this module.
Allowable values:	A non-negative quantity
Default value:	0
gfr-upgrade-procedure	The name of the procedure to call to upgrade an application module developed in an older version of this module.
Allowable values:	The name of an upgrade procedure, or the symbol unspecified
Default value:	unspecified
gfr-copyright-information	Copyright information for this module.
Allowable values:	Any text
Default value:	""
gfr-build-information	Additional information about the version of this module, such as the build date.
Allowable values:	Any text
Default value:	""

When a version information object is not on a proprietary workspace, you can manually edit the attributes of the version information object. To prevent accidental edits, make the workspaces containing your version information objects proprietary before release.

You must be in administrator mode to carry out manual edits. In any user mode other than administrator, only the attributes gfr-module-name, gfr-package-name, gfr-version-description, gfr-minimum-g2-version, and gfr-copyright-information are visible.

Note You must manually specify the attributes of the version information object before distributing your module.

Specifying the Module Version

The version information object contains information on the module version:

- gfr-module-name (a symbol) – The name of the module containing the object.
- gfr-package-name (a text) – The name for a collection of modules to which this module belongs.
- gfr-version-description (a text) – The version description that identifies the release for the user in the terminology of your choice.
- gfr-version-number (an integer) – The version number that places this release in sequence with other releases of the same module. "Version 1 Rev. 0" might be version number 1; "Version 1 Rev. 1" might be version number 2; and "Version 2 Rev. 0" might be version number 3.
- gfr-copyright-information (a text) – The copyright information for the module.
- gfr-build-information (a text) – Additional information on the module.

Usually, the version numbers increase with each successive release, although in some cases, the sequencing of releases might be more complex, particularly when different major versions are simultaneously being supported. For example, you might release Version 1.1 Rev. 1 sometime after Version 2.0 Rev. 0 to support customers still using Version 1.1, but you want Version 1.1 Rev. 1 to have a smaller version number than Version 2.0 Rev. 0.

You may leave gaps in the version number sequence between major versions to allow for future revisions of older versions. Gaps in the version number sequence do not affect GFR's version validation.

Specifying the Minimum G2 Version

By setting the gfr-minimum-g2-version attribute, you are assured of using the correct version of G2.

To set the minimum G2 version, specify a value for gfr-minimum-g2-version.

If you do not enter the G2 version correctly, GFR may be unable to validate that the correct version of G2 is being used.

GFR assumes that higher, newer versions than the gfr-minimum-g2-version can be used. For example, if the minimum version is "Version 6.1 Rev. 0," then Version 6.0 Rev. 2 would be acceptable.

GFR also checks the module sys-mod, which is always loaded when GFR is present. Since sys-mod version numbering matches that of G2, the sys-mod version must also conform to the minimum G2 version. GFR signals an error at startup time if you use an invalid version of G2 or sys-mod.

Specifying the Oldest Compatible Module Version

By setting the gfr-oldest-compatible-version-number attribute, you can control the loading of applications of your module into previous versions of your module.

To specify the oldest compatible module version, specify a value for gfr-oldest-compatible-version-number.

For example, if the current version of your module is Version 1.1 Rev. 1, and it is permissible to load an application developed in Version 1.1 Rev. 1 back into Version 1.1 Rev. 0, then the gfr-oldest-compatible-version-number should be the version number of Version 1.1 Rev. 0.

The ability to move an application backwards to previous versions of a supporting module is not often supported, so in many cases the gfr-oldest-compatible-version-number is the same as the gfr-version-number. GFR signals an error at startup time if the user tries to "move backwards" to a previous version of your module that is older than the gfr-oldest-compatible-version-number.

Note In rare special cases, you may need to disable version checking. Disabling allows the startup to complete regardless of version incompatibilities. To disable version checking before merging modules into a running G2, call gfr-disable-version-checking. To turn off version checking for startup, use the startup settings object. Use this option with caution, since the effects of running incompatible versions and omitting required upgrades is unpredictable.

Getting Version Information

GFR provides a procedure, gfr-get-g2-version, that returns the G2 version information as a float, indicating the major and minor versions, and an integer, indicating the revision number. The system procedure g2-get-software-version, returns this information as a text, which you normally would have to parse to get information useful to a procedure or rule.

GFR also provides a procedure that returns its own version, gfr-get-version. For details on these procedures, see File Parsing and Miscellaneous Functions and Procedures.

Providing an Upgrade Procedure

In general, when you release a new version of your module, users load it with G2's "automerge" facility (loading or merging automatically resolving conflicts) the first time they load their applications with the updated module. Automerge automatically alters instances in the application module so that they conform to the current class definitions.

In many cases, nothing further is required to upgrade an application to a new release of a supporting module. However, when data structures have been fundamentally redesigned, it may be necessary to take additional steps to convert old representations to new ones.

Whenever a module developed in an older version of your module is loaded, GFR automatically calls the procedure named by the attribute gfr-upgrade-procedure.

For example, when an application developed in Version 1.1 Rev. 0 is loaded into a newer version of your module, say Version 2.0 Rev. 0, the upgrade procedure in Version 2.0 Rev. 0 is called.

The signature of the upgrade procedure is:

my-upgrade-procedure
(*module-to-upgrade*: symbol, *version-developed-in*: integer,
client: class object)
-> *save-required*: truth-value)

where:

module-to-upgrade: The name of the application module undergoing the upgrade.

version-developed-in: The version number (not description) of the supporting module that was used to develop the application module.

Note This number is not the same as the version number of the application module. It is the version number of your module that was present when the application module was last loaded.

---

client – A g2-window or other object representing the source of this call.

save-required – A truth-value indicating whether changes were made that require the user to save the application module.

The upgrade procedure you supply may take different actions depending on the version number associated with the application module. For example, if the current version of your module is 2.0 Rev. 1, you may not have to take any actions when loading an application developed in 2.0 Rev. 0, but certain actions might be required when loading an application developed in Version 1.1, and yet another set of actions for applications developed in Version 1.0.

Module developers must keep track of the version history of this module and the upgrade actions required to move between versions. If you cannot successfully upgrade the application module (perhaps it is too old to be upgraded), then your upgrade procedure should signal an error.

For more information on upgrades and version checking performed as part of the startup sequence, see Using Module Startup Objects.

Using Module Startup Objects

GFR startup objects help you initialize modules in the correct order. Lower-level modules should always start before higher-level modules. To make sure your module initializes and warmboots in the correct order according to its place in the module hierarchy, GFR provides an object of the class gfr-startup-object. GFR startup objects also provide support for warmbooting.

The icon for the startup object appears in the following figure:

Gensym recommends that you coordinate startup activities using GFR startup objects, rather than initially rules. You cannot assure the correct order of startup using G2 initially rules, unless the end user carefully adjusts the priorities of the rules to reflect the module hierarchy and maintains these priorities whenever the module hierarchy is changed. Adjusting rule priorities is impractical and infeasible if some of the modules involved are proprietary.

GFR uses GFR startup objects in three situations:

- When G2 is initially started.
- When new modules are created or modules are merged into a running G2.
- When a KB that was saved using G2's snapshot facility is loaded using the "warmboot afterwards" option.

If you have a startup object in your module, your module does not have to be concerned with detecting startup, warmboot, or merge events because GFR detects them for you.

To add a startup object to your module:

➔ Clone a gfr-startup-object from the GFR palette and place it on any workspace in your module.

Each module that requires GFR should contain **at most one** startup object.

Attributes of the Startup Object

The attributes of a gfr-startup-object object are summarized in the following table:

Attribute	Description
startup-procedure	A symbol naming the startup procedure to be called when the module containing the object is activated.
Allowable values:	Any procedure name
Default value:	unspecified
warmboot-procedure	A symbol naming the procedure to be called when the module warmboots after loading from a snapshot file.
Allowable values:	Any procedure name
Default value:	unspecified
module-is-active	A read-only attribute that indicates if the module startup procedure has been run.
Allowable values:	true or false
Default value:	false

The startup-procedure attribute names the procedure you want to have called when G2 is started and when modules are created or merged. The signature for your startup procedure must be:

```
my-startup-procedure
    (object: class gfr-startup-object, initial-startup: truth-value,
     modules: class symbol-list, client: class object)
```

where:

object: The startup object that generated the call to the procedure. You can determine if the module has been previously started by looking at the module-is-active attribute. After the startup has been run once, GFR sets the module-is-active attribute to true.

initial-startup: A truth-value indicating whether this is the first startup managed by GFR after G2 was started.

modules: A symbol list that contains the names of all modules dependent upon the module that should be started up at the time of the call. This list can contain the module itself.

client: An object that represents the source of the call.

The startup procedure in your module performs a dual purpose:

- To initialize your module.
- To initialize the modules dependent on your module.

These two activities might be performed together, or separately, depending on whether the procedure is being called when G2 initially starts, or after modules are created or merged.

When the module-is-active attribute of the startup object is false, your module should perform whatever activities it needs to do to get itself running. Typical startup activities include:

- Making external connections to data sources.
- Starting monitoring procedures that need to be continuously running.
- Presenting welcome screens.

The *Modules* argument indicates all the dependent modules that should be started up. This list includes the name of your module if it has not yet been activated. Your startup procedure should initialize the application objects defined by your module and found in the modules named in the *Modules* list. These activities typically include:

- Setting initial states of application objects.
- Restoring non-permanent data structures associated with application objects.

When *InitialStartup* is true, your module must initialize itself and all objects in the KB defined by your module. You do not have to check for the module assignment of the objects you initialize. If *InitialStartup* is false, your startup procedure is being called either because a new module was created or because modules were merged while G2 was running. In this case, you should initialize only the objects found in modules explicitly given by the *Modules* list; otherwise, you may re-initialize an object unnecessarily.

Note You can use the function gfr-get-module-of-item to determine the module assignment of items.

Note The startup procedure is also called when your module is merged into a running G2. If GFR is merged at the same time as your module, *InitialStart* = true. If GFR was previously loaded, then *InitialStart* = false and *module-is-active* = false. Consequently, your module must start itself and all dependent modules in the *Modules* list.

Whenever a G2 is started or modules are merged into a running KB, GFR gathers a complete list of modules to start. It then performs the following series of startup actions in the following order:

1. If GFR has not already been started, GFR installs startup, Logbook, and Message Board settings (see Managing User-Settable Parameters for Modules).
2. GFR obtains a list of all modules arranged in bottom-up order.
3. For each module in the complete module list, GFR:
   - **a** Validates the G2 and sys-mod versions for the specified module, if this module is in the list of modules to start.
   - **b** Validates and upgrades all modules requiring the specified module, if the requiring modules are in the list of modules to start.
   - **c** Installs the specified module's settings if any active module settings have changed.
   - **d** Calls its startup procedure (if there is one), if this module is in the list of modules to start.
4. GFR prompts the user to save any modules that required an upgrade.

Warmbooting

To assure that your module is warmbooted in the correct order, use the warmboot-procedure attribute. This attribute names the procedure you want to have called when your KB is loaded from a snapshot file. This procedure should *not* be named "warmboot".

Typically, the warmboot procedure does things like restart procedures that were running when the snapshot of the KB was taken. The signature of your warmboot procedure must be:

```
my-procedure-name
    (object: class gfr-startup-object)
```

where:

object: The startup object that generated the call to the procedure.

When GFR's warmboot procedure is called, it first calls all other procedures in the KB named warmboot. Then, it traverses the module hierarchy from the bottom up, calling the procedures named by the warmboot-procedure attribute of each startup object.

Caution To use GFR's warmboot procedure, GFR must be the first module containing a procedure named warmboot loaded into the KB. Otherwise, G2 does not call GFR's warmboot procedure, because G2 only calls the first procedure named warmboot when a snapshot file is loaded.

Caution Loading or merging a KB using the resolve all conflicts option can result in automatic deletion of procedures named warmboot, if there are any outside the module GFR. Because GFR contains a procedure named warmboot, do *not* name other procedures warmboot in the KB. Before you merge GFR into an application for the first time, rename any procedures named warmboot to prevent their accidental deletion.

Starting Up When a KB is Not Consistently Modularized

GFR starts up a KB in most circumstances, even if your KB is not consistently modularized. For example, if modules are present that are not required by the KB, they are still started up in a consistent order.

However, GFR cannot start up modules that directly or indirectly require other modules that are not present in the KB. A module that is missing required modules has the attribute module-is-active = false, and its startup procedure is not called.

Managing User-Settable Parameters for Modules

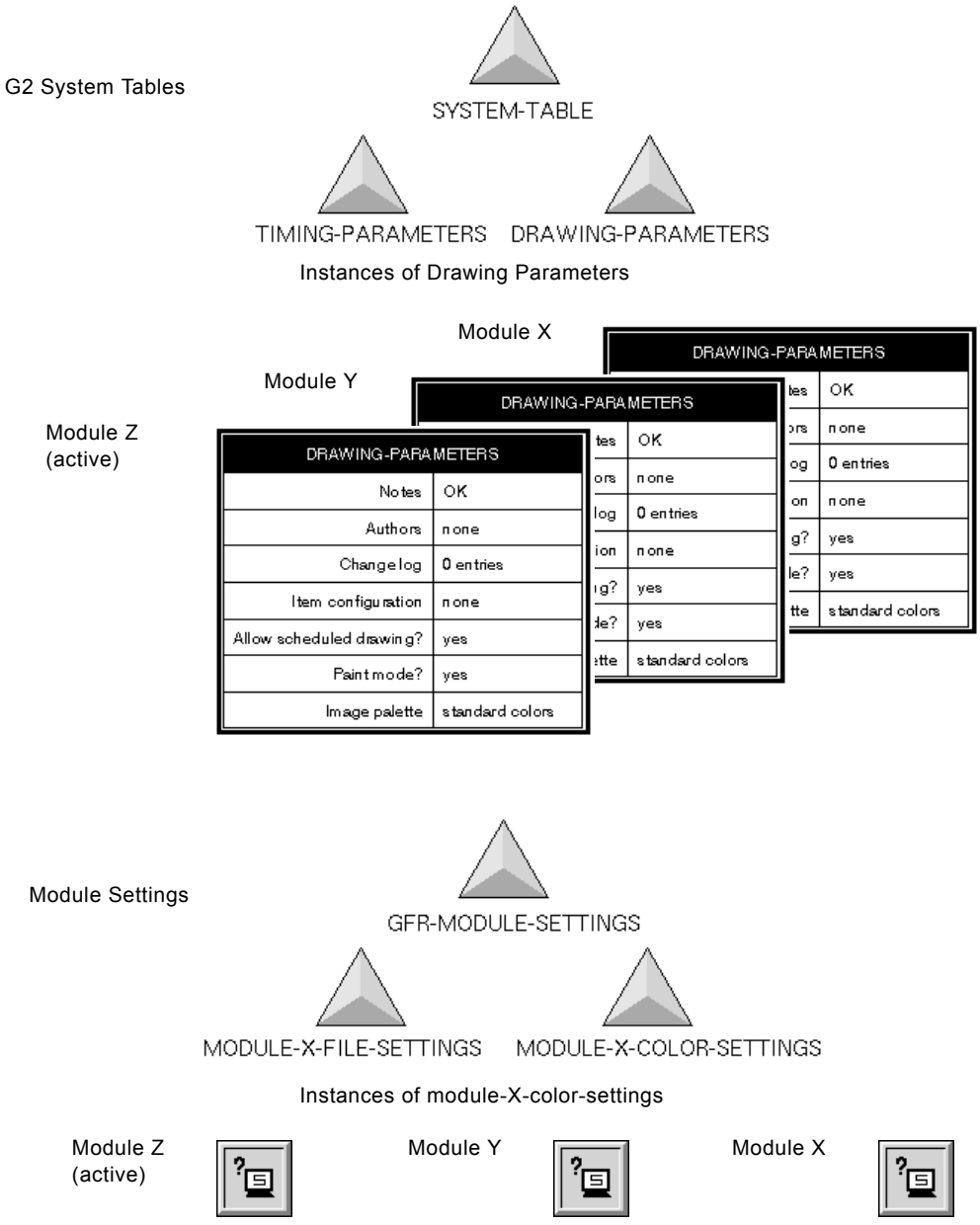
Modules often provide user-settable parameters that tailor certain module behaviors. For example, a module might let the user customize the colors it uses or provide default file pathnames.

The user-settable parameters of G2 appear in the System Tables, which allow the user to tailor the behaviors of G2 itself, including the choice of fonts, timing parameters, drawing parameters, and many more. For modules, there is no built-in mechanism akin to G2 system tables.

GFR provides a standard approach for modules to manage their user-settable parameters, which are called **module settings**. This facility is analogous to G2's System Tables for the following reasons:

- A module can have different types of module settings like different types of System Tables.
- While there may be more than one module setting of a given class present in the module, exactly one module setting of each class is active at any time.
- The active module settings are determined by the module hierarchy, with precedence given to higher-level modules.

The following figures illustrate the analogy between G2 system tables and module settings.



The module settings figure assumes that module Z is the top-level module, and module X defines two types of module settings for itself: the module-x-file-settings and the module-x-color-settings.

One instance of module-x-color-settings exists in each module X, Y and Z, just as one drawing-parameters system table exists in each module. The module-x-color-settings instance assigned to the top module (Z) is active.

The following differences exist, however, between the system tables and module settings:

- While each module comes equipped with a complete set of system tables, modules do not contain module setting objects unless they are specifically created and placed there.
- The active system tables are always those in the top-level module. In contrast, the active module setting of a given type is the one highest in the module hierarchy, even if it is not in the top-level module.

Loading and Activating Module Settings

When G2 is started or when modules are merged, GFR searches the module hierarchy for module settings objects and installs them in the modules where they belong. Thus, when GFR activates the module, its active module settings are already determined.

GFR determines which module settings to install in a given module by looking at the subclasses of gfr-module-setting defined in that module. For each setting class defined in the target module, GFR determines which instance of that class should be active, using the module hierarchy precedence described in the previous section.

Caution You should always provide a default instance of every class of module setting you define, so that it is guaranteed that GFR will find a suitable module setting to install.

When GFR locates the instance, it calls the method gfr-propagate-module-setting-information. The purpose of this method is to take the actions necessary to implement the settings contained in the active setting object. You may, for example, copy information in the setting object into private data structures within the target module.

The signature of the gfr-propagate-module-setting-information method is as follows:

gfr-propagate-module-setting-information
(*setting*: class gfr-module-setting)

where:

setting: The module setting object that is being activated. Optionally, you can provide a method of this name for each subclass of gfr-module-setting you define.

Note Your code must call the gfr-propagate-module-setting-information whenever the attributes of the active setting are edited by the user, if these changes are to take effect immediately. If the active setting changes because the user merges modules, the new modules settings are automatically installed.

When you want to retrieve the active setting of a particular class of module setting, use the API procedure gfr-get-active-setting. If you create a new instance of a module setting that you intend to become the active instance, call gfr-install-module-settings.

Example

As shown in the previous figure, module X defines the module setting class module-x-color-settings. Suppose this object has an attribute named highlight-color, and an internal parameter in module X named active-highlight-color needs to be set to this color. You write the following method:

```
gfr-propagate-module-setting-information(Setting: class module-X-color-
    settings)
begin
    conclude that active-highlight-color = the highlight-color of Setting;
end
```

To reference the active color setting object at any time, you would include the following line of code:

```
Setting = call gfr-get-active-setting(the symbol module-X-color-settings, Win);
```

Using GFR's Module Settings

GFR defines three module settings found on GFR's palette:

- gfr-startup-settings – Allows you to specify the level of messages produced by GFR during the startup process.
- gfr-message-board-handler-setting – Allows you to specify a custom message board handler.
- gfr-logbook-handler-setting – Allows you to specify a custom logbook handler.

To use GFR module settings:

➔ Clone an instance of the desired setting from GFR's palette and place it in your module.

If your module is the highest in the module hierarchy that defines the specified class of settings, GFR installs your settings.

Attributes of gfr-startup-settings

The attribute settings for the gfr-startup-settings class are specified in the following table:

Attribute	Description
gfr-confirm-before-upgrade	A truth-value that enables you to specify whether GFR prompts the user to confirm or prevent the launch of an upgrade procedure on a module. False means the user is not prompted.
Allowable values:	true or false
Default value:	false
gfr-prompt-for-save-after-upgrade	A truth-value that enables you to specify whether GFR prompts the user to confirm or prevent the saving of an upgraded module after all required upgrade procedures have been called. False means the user is not prompted.
Allowable values:	true or false
Default value:	true
gfr-trace-startup	A truth-value that enables you to specify that GFR display trace messages the steps G2 is taking on every startup. The messages are displayed on the Message Board. False means that trace messages are not displayed.
Allowable values:	true or false
Default value:	false

Attribute	Description
gfr-error-handling-enabled	A value of true enables GFR error handling; a value of false disables it.
Allowable values:	true or false
Default value:	false
gfr-version-checking-enabled	A value of true enables GFR version checking; a value of false disables it.
Allowable values:	true or false
Default value:	true

Attributes of gfr-message-board-handler

The attribute setting for the gfr-message-board-handler class is specified in the following table:

Attribute	Description
gfr-message-board-handler	A symbol naming the message handler procedure to be called when the module needs to send messages to the Message Board.
Allowable values:	The name of a procedure that takes one text argument and returns none
Default value:	unspecified

Attributes of gfr-logbook-handler

The attribute setting for the gfr-logbook-handler class is specified in the following table:

Attribute	Description
gfr-logbook-message-handler	A symbol naming the logbook message handling procedure to be called when the module needs to send messages to the logbook.
Allowable values:	The name of a procedure that takes one text argument and returns none
Default value:	unspecified

Using Module Management Procedures and Functions

GFR provides several procedures and functions for module management. These procedures and functions allow you to determine the precedence of modules in the module hierarchy and provide a mechanism for depositing items in another module.

Getting Information on the Module Hierarchy

Several procedures and functions allow you to determine the module of any item and the precedence of modules in the module hierarchy:

- To determine the modules that are directly required by a module, use gfr-get-directly-required-modules.
- To determine all modules that are directly or indirectly required by a module, use gfr-get-required-modules.
- If you need to know which modules directly require a module, use gfr-get-directly-requiring-modules.
- If you want to know which modules directly or indirectly require a module, use gfr-get-requiring-modules.
- To determine the top-level module, use gfr-get-top-level-module.
- To return the module hierarchy flattened into a list, use gfr-get-linearized-module-hierarchy. The list contains all modules sorted so that if a module X is required by another module Y, module X always appears after module Y in

the list. The linearization follows the same rules as the linearized class inheritance path for object definitions with multiple inheritance.

- To determine the current module assignment of any item, use gfr-get-module-of-item.

For more details on these procedures, see Module Management Utilities.

Managing Cached Module Information

Because the module hierarchy is relatively static, GFR stores (caches) data on the required and requiring modules at startup time in a form that GFR can access efficiently. When GFR receives a request for information on the module hierarchy, GFR does not re-analyze the module hierarchy, but instead, returns information based on its stored data.

GFR automatically updates the cached module information whenever the module hierarchy is changed.

If you change the module hierarchy programmatically, GFR does not de-cache the stored module information *immediately*. GFR de-caches the stored module information only after a rule (scheduled at priority 6) gets to run.

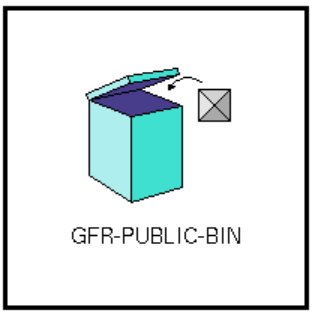
If you want to immediately access the revised module hierarchy after making a programmatic change, you must force GFR to de-cache the stored information by calling the procedure gfr-invalidate-module-information, which causes GFR to recompile its cached module hierarchy information. If you fail to make such a call, GFR will return outdated information based on the module hierarchy before it was changed.

Depositing Items in Other Modules

For a variety of reasons, a module may have to place an item into another module, on a permanent or transient basis. Although you can programmatically create a new workspace, assign it to the module, place whatever object needs to be stored upon it, in time and with enough modules, this practice can lead to a multiplicity of these "bin" workspaces inside each module.

Instead of having multiple modules independently creating workspaces within a module, GFR provides for a "public bin" for each module where one is required. GFR provides this procedure gfr-deposit-item-in-public-bin as a standard method for placing an item into a module's public bin.

By convention, the bin is the subworkspace of an item of the class gfr-public-bin named *module_name*-public-bin. You can retrieve the public bin for a module, using the procedure named gfr-get-public-bin-for-module. The icon for the public bin appears in the following figure:



GFR does not automatically make permanent items placed in the public bin. If you want to make an item permanent, do so after it is placed in the bin. You must retain some sort of handle to the items you place in the bin, because no procedure lets you remove an item from the bin. When you want to remove an item from the bin, you do so directly.

The arrangement of items on the bin workspace is random, not laid out in any regular pattern. Making an artistic layout in the bin takes considerable computational time, if the number of items in the bin gets very large.

If you change the name of a module, you must also change the name and module assignment of the public bin assigned to that module.

3

Handling Errors and Communications

Describes the model of communications handling used in GFR.

Introduction

The user interface is one of the most important shareable resources in a multiple-module system. In the absence of standards, the user interface of a multiple-module system can easily become a melange of inconsistent interface styles. One module might route its messages to the logbook, another might use the Message Board, another might use scrollable queues, and yet another might use dialogs.

While it might appear that this is a cosmetic problem, more serious design issues are at stake. For example, a monitoring system for a power plant might display critical safety parameters in the center of the screen. If a dialog from a supporting module pops up in the center of the screen, safety could be compromised.

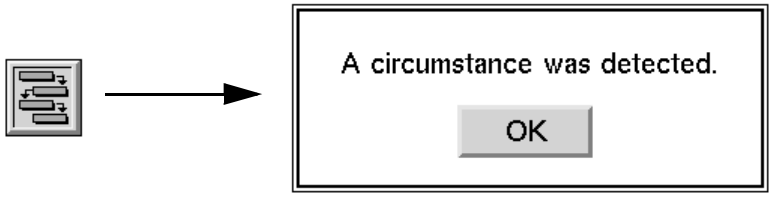
The specific requirements for the end-user interface vary from application to application. Yet you must often write support modules before these requirements are known.

GFR introduces a style of communications applicable to a wide range of communications, including error messages, that allows low-level modules to

"soft-code" accesses to the user interface. Using this system, higher-level modules can override or customize user interface behaviors defined in lower-level modules.

In GFR, you model user communications as objects whose ultimate representation in the user interface is controlled by a special class of procedures called *communication handlers*. If you want to override how a communication is presented to the user, you provide your own communication handler and assign it to a specific type of communication.

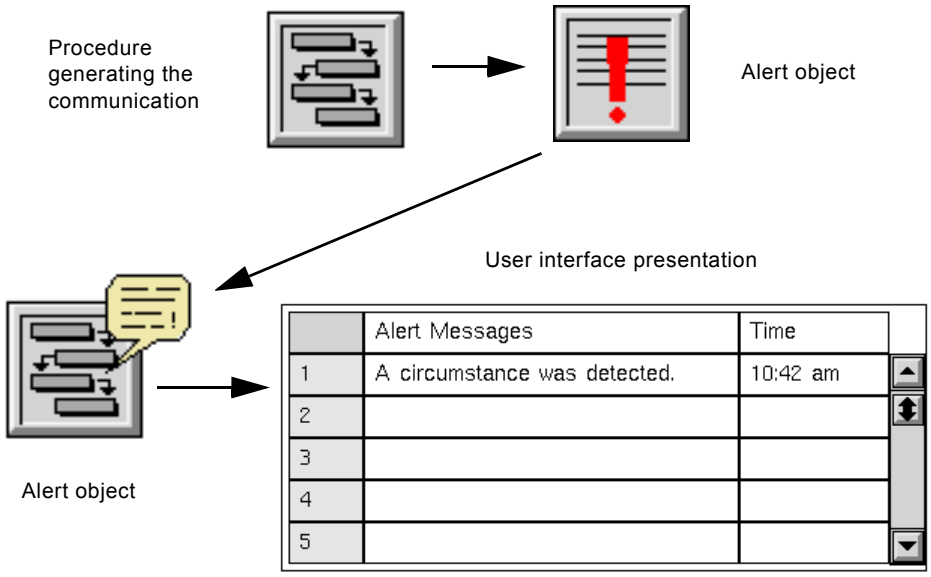
Suppose that a procedure needs to communicate to the user a special circumstance via an alert message. In the simplest and most direct approach, the procedure creates and manages its own user interface element, in this case by generating a borderless free text, putting it on a workspace with an action button, and showing it to the user:



Unfortunately, because the presentation of this communication is hard-coded, the behavior cannot be modified, except by modifying the source code. Even if you had access to all the source code, it would be very burdensome to modify every procedure that made direct reference to the UI.

By contrast, if you structure your communications according to GFR's design, users of your module have ultimate control over the style of the communications your module generates.

The following figure illustrates GFR's approach for structuring your communications:



In the figure, the procedure generating the communication creates an object that contains all the information required for the communication. In this case, the alert object is a gfr-alert.

GFR then selects a handler from the set of handlers for alert objects. GFR bases the selection on the handler's position in the module hierarchy and the class of communications it is able to handle. Handlers in higher-level modules take precedence over handlers in lower-level modules. For more information on how GFR assigns precedence, see Handler Precedence.

The selected alert handler uses the information in the alert object to present the information in its own style, for example, posting the alert to a message queue.

The three communication objects, gfr-alert, gfr-ok-cancel-confirm, and gfr-yes-no-cancel-confirm provide default handlers that display dialogs with various buttons. You have the option of viewing these dialogs as standard Windows dialogs when viewed through Telewindows on Windows platforms. To enable this feature, enable the Use Native Dialogs option on the gft-top-level workspace.

Communication and Error Handlers

GFR provides a special class of procedure that you use to define custom communication handlers. The class is called gfr-communications-handler. A similar class for defining error handlers called gfr-error-handler. The following figure shows the icons for a gfr-communications-handler and a gfr-error-handler:

gfr-communications-handler

gfr-error-handler

In addition to the normal attributes of procedures, the handler classes have the attribute gfr-applicable-class, which is a symbol naming the class of error or communication the procedure handles. Initially, this attribute is the symbol unspecified. You must change this attribute to the name of a class of communication or error, or GFR will ignore your handler. You can edit handlers like normal procedures. Each handler must have a unique name.

Note You cannot use normal procedures and methods as communication or error handlers. You must use a gfr-communications-handler or a gfr-error-handler procedure.

GFR controls the dispatch of communications to handler procedures through a procedure named gfr-dispatch-communication. When you create an instance of a gfr-communication, you send it to a handler by calling gfr-dispatch-communication. Gfr-dispatch-communication, like every communication handler you write, takes three arguments:

- The communications object (class gfr-communication)
- An initiating item (item or value)
- The client (class object)

Your communication handler procedure must return a structure, as described in Using Communications Objects.

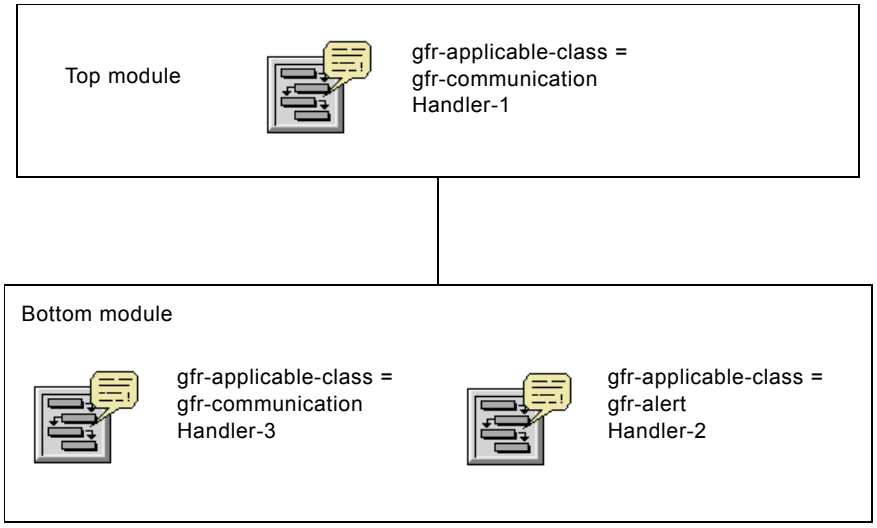
Every error handler must take one argument, the error being passed: class error. Error handlers do not return values.

Handler Precedence

When you call the gfr-dispatch-communication procedure, GFR selects a handler according to the following rules:

- The gfr-applicable-class of the selected handler must name the class of the communication or a superior class of the communication.
- Within a single module, handlers with a more specific applicable class reference are given precedence over handlers with more general class references.
- Handlers in higher-level modules are given precedence over handlers in lower-level modules, even if the class reference is less specific.

The precedence of communication handling is illustrated in the following figure:



In this figure, there are three possible handlers for a gfr-alert, which is a subclass of gfr-communication. The precedence order is: Handler-1, then Handler-2, then Handler-3.

Handler-1 is first because it is in a higher level module. Handler-2 is second because it is a more specific applicable class reference: an alert class. Handler-3 is third because it is a more general class reference.

Handler-1, even though it refers to the less specific class gfr-communication, overrides Handler-2 because Handler-1 is in a higher level module. By including a handler capable of handling all communications in the top module, the author of the top module has overridden the handling of all gfr-communication objects.

Note The precedence of communication handling also applies to error handlers.

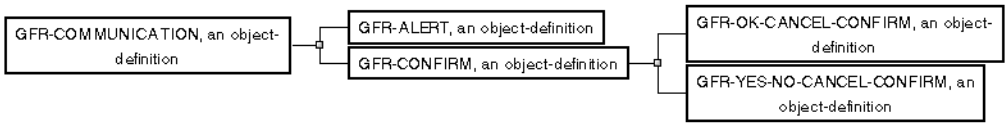
To see the current handler hierarchy for any communications class, select the menu choice show-handler-hierarchy, which appears on instances of gfr-communication and it subclasses, error and it subclasses, class definitions that define communications or errors, and handler procedures. Programmatically, you can access the same information, using the procedure gfr-get-handler-hierarchy.

GFR provides a facility for handlers that is similar to G2's call next method. You can call the next handler in the sequence by calling gfr-call-next-communication-handler. When you make this call, GFR finds the next-highest handler and calls it. Calling the next handler is optional. If your handler presents the communication in the desired manner, you would not need to call the next handler. If the next handler does not exist, and you attempt to call it, GFR will signal an error.

The call to the next error handler is gfr-call-next-error-handler. This call is also optional.

Using Communications Objects

GFR provides a small set of communications objects, including a root object, gfr-communication, and classes for alert and confirm messages, shown in the class hierarchy below:



When you define or use a gfr-communication object, you must consider return arguments. For example, a purely informational message to the user will not return any arguments, but a message that asks the user for permission before performing an action will return (at least) one true/false argument. All communication handlers return one argument of type structure to contain the appropriate returns. Gfr-dispatch-communication returns the structure returned to it by the highest-precedence handler.

To create your own types of communications:

➔ Subclass from gfr-communication.

To allow the creation of custom handlers, the subclasses you create and the attributes of these subclasses should typically be public. There are no other restrictions on the classes you create. You should document the number and type of return arguments associated with your class of communications, including the attribute name for each return argument.

Using gfr-alert

To create and specify the text of an alert communication:

1. Create a gfr-alert object, using the G2 create action.
2. Specify the localizable texts that appear as the prompt and button label by configuring the text proxies of the alert object.
3. Call gfr-dispatch-communication to send the alert to the appropriate handler.

These steps are illustrated in the following code fragment:

```
create a gfr-alert Alert;
call gfr-configure-text-proxy(the gfr-prompt-text of alert,
     the symbol gfr-test-messages, the symbol gfr-test-alert);
Response = call gfr-dispatch-communication(Alert, InitiatingItem, Client);
delete Alert;
```

Handlers for the gfr-alert class return an empty structure, so in this code fragment, the returned structure Response contains no attributes.

The default handler for gfr-alert produces a popup dialog that looks like the following figure:

As soon as GFR posts the popup dialog to the screen, the handler returns control without waiting for a response. In general, handlers do not delete the communications objects passed to them, so you must delete the alert object after the handler returns.

Note The attribute gfr-handler-class exists for compatibility with GFR 4.1 and is not used in the current version of GFR.

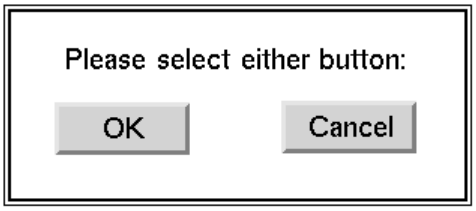
The gfr-alert class contains the following attributes:

Attribute	Description
gfr-handler-class	(not used)
gfr-prompt-text	A text proxy which you configure with the message of the alert.
Allowable values:	A gfr-text-proxy
Default value:	A gfr-text-proxy
gfr-button-label	A text proxy that you configure with the label of the acknowledgment button of the alert.
Allowable values:	A gfr-ok-text-proxy, which by default has the text resource group gfr-text-resource, and the message name gfr-ok
Default value:	A gfr-ok-text-proxy

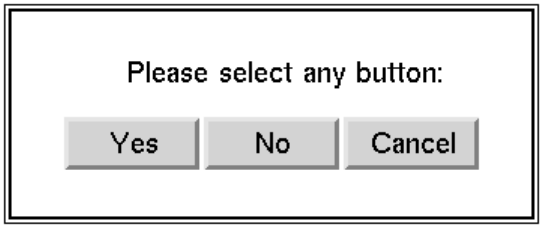
Using gfr-confirm

Unlike an alert dialog, a confirm dialog returns a value indicating which button was selected by the user. The thread of processing is suspended until the user responds or until the confirm times out.

The GFR default handler for gfr-ok-cancel-confirm produces a popup dialog that looks like the following figure:



The default handler for gfr-yes-no-cancel-confirm is similar, but has three buttons:



Handlers for the gfr-confirmation class return a structure with one symbolic attribute, named confirmation-result, indicating the button that was selected by the user, or the symbol timeout.

- For a gfr-yes-no-cancel-confirm, the confirmation-result is either the symbol yes, no, cancel, or timeout.
- For a gfr-ok-cancel-confirm, the confirmation-result is either ok, cancel, or timeout.

To create and specify the text of a confirm communication:

1. Create a gfr-confirm object, using the G2 create action.
2. Configure the prompt text proxy in the gfr-prompt-text attribute. Optionally, you can configure the button label text proxies to substitute your own button labels.
3. Call gfr-dispatch-communication to send the confirm to the appropriate handler. The call returns a structure.
4. Delete the confirm object.

These steps are illustrated in the following code fragment:

```
create a gfr-confirm Confirm;
call gfr-configure-text-proxy(the gfr-prompt-text of Confirm,
     the symbol gfr-test-messages, the symbol gfr-test-confirm);
Response = call gfr-dispatch-communcation(Confirm, InitiatingItem, Client);
inform the operator that “the response was
     [the confirmation-result of Response]”;
delete Confirm;
```

Handlers for the gfr-confirm class return a structure containing the attribute confirmation-result, which will contain the value.

Note The attribute gfr-handler-class exists for compatibility with GFR 4.1 and is not used in the current version of GFR.

The gfr-confirm class and its subclasses contain the following attributes:

Attribute	Description
gfr-handler-class	(not used)
gfr-prompt-text	A text proxy that you configure with the prompt text of the confirm.
Allowable values:	A gfr-text-proxy
Default value:	A gfr-text-proxy
gfr-confirmation-timeout	The maximum time that the handler is to wait for the user to confirm the message, before returning the symbol timeout.
Allowable values:	Any positive float, if the timeout is to be used; if the timeout is to be ignored, zero or any negative float.
Default value:	0.0
gfr-cancel-button-label	A text proxy that you configure with the label of the cancel button of the confirm.
Allowable values:	A gfr-cancel-text-proxy, which by default has the text resource group gfr-text-resource, and the message name gfr-cancel
Default value:	A gfr-cancel-text-proxy
gfr-ok-button-label (for gfr-ok-cancel-confirm)	A text proxy that you configure with the label of the OK button of the confirm.
Allowable values:	A gfr-ok-text-proxy, which by default has the text resource group gfr-text-resource, and the message name gfr-ok
Default value:	A gfr-ok-text-proxy

Attribute	Description
gfr-yes-button-label (for gfr-yes-no-cancel-confirm)	A text proxy that you configure with the label of the yes button of the alert.
Allowable values:	A gfr-yes-text-proxy, which by default has the text resource group gfr-text-resource, and the message name gfr-yes
Default value:	A gfr-yes-text-proxy
gfr-no-button-label (for gfr-yes-no-cancel-confirm)	A text proxy that you configure with the label of the no button of the confirm.
Allowable values:	A gfr-no-text-proxy, which by default has the text resource group gfr-text-resource, and the message name gfr-no
Default value:	A gfr-no-text-proxy

Using GFR's Error Handling Facility

GFR provides a method of handling errors that is similar to the communications handling model. Using GFR's error handling facility, you can define handlers for different classes of errors. This gives you added flexibility in the way errors are presented to the end user, and allows the end user to override default error handling defined in low-level modules.

In G2, errors can occur in two ways:

- G2 will generate an error when it is asked to do an illegal operation, such as referencing the value of an uninitialized variable, or referencing an object that does not exist. When G2 detects that an illegal operation has been attempted, it signals an error of the class g2-error or g2-rpc-error.
- The user can raise an error using the signal statement. Signal statements are typically used when an application detects condition that prevents normal processing. When you signal an error, you can signal a symbol and text argument, or signal the error object that is an instance of any subclass of error. If you signal a symbol and text, G2 will transparently perform a conversion into an error object of class default-error, if error handling code is written in an object-oriented form.

By default, GFR's error handling facility is off. If you want to use the facility, place an instance of a gfr-startup-settings in your module, and change the attribute gfr-error-handling-enable to true. This will take effect when you next start G2. To

activate GFR's error handling without restarting G2, use the API procedure gfr-enable-error-handling.

When the error handling facility is active, all errors that are not otherwise caught by on-error statements are dispatched by GFR. If an appropriate error handler is found, GFR calls that error handler. If there is no appropriate error handler, GFR routes the error to the logbook. GFR determines the precedence of error handlers, using the same rules as for communication handlers, namely:

- Within a single module, handlers with a more specific applicable class reference are given precedence over handlers with more general class references.
- Handlers in higher-level modules are given precedence over handlers in lower-level modules, even if the class reference is less specific.

When GFR looks for error handlers, only procedures that are instances of the class gfr-error-handler are considered. The gfr-applicable-class attribute of the handler determines what errors can be sent to the handler. A handler with the applicable class error can handle any error; one with the applicable class g2-rpc-error will only handle errors produced in remote procedure calls, etc.

In the error handling facility, there is no procedure analogous to gfr-dispatch-communication. When the error handling facility is active, all signalled errors that are not caught by on-error statements are automatically dispatched to handlers.

Caution When GFR error handling is active, and an error occurs for which there is no appropriate handler, the error message on the operator logbook does not support the "go to referenced item" menu choice normally associated with logbook messages.

The gfr-error Class

GFR provides a class of error, gfr-error, that provides language localization, using GFR's localization facility. The gfr-error class has the following attributes:

Attribute	Description
gfr-text-resource	The name of a text resource group where the text of the error message is defined.
Allowable values:	A symbol naming a gfr-text-resource-group.
Default value:	unspecified
gfr-message-name	The symbolic key for the message associated with the error.
Allowable values:	Any symbolic key defined by the text resource group.
Default value:	unspecified
gfr-localized-text	The localized text for this error.
Allowable values:	Not user specified (filled in by GFR)
Default value:	""

When your application signals a error that you want GFR to localize, you create a gfr-error, and set the attributes gfr-text-resource and gfr-message-name. GFR will fill in the localized text before the error reaches any handler (or the logbook, if no handler for the error is found). Because errors are not normally associated with particular G2 clients, the language used in the localization is G2's default language, as defined in the current-language of the language-parameters system table.

The following code fragment illustrates how to signal an error, using the gfr-error class:

```
create a gfr-error Error;
conclude that the gfr-text-resource of Error = the symbol my-resource;
conclude that the gfr-message-name of Error = the symbol
    my-error-message;
signal Error;
```

Writing Your Own Handlers

The purpose of the GFR communications handling model is to enable you, or users of your module, to override default handling of error and messages. To modify default methods of error or communications handling, you write override handlers.

To create a communication handler:

1. Clone a gfr-communications-handler from the GFR palette and place it in your module.
2. Set the gfr-applicable-class to the name of the class of communication that it is going to handle.
3. Edit the procedure, giving it a unique name and the following signature:

```
my-communication-handler(communication: class gfr-communication,
    initiating-item: item or value, client: class object) = (structure)
```

For example, you can create an override handler that send alerts to the Message Board.

To send alerts to the Message Board, instead of displaying a popup dialog:

➔ Create a gfr-communication-handler named my-alert-handler with the following text:

```
my-alert-handler(Alert: class gfr-alert, InitiatingItem: item or value,
    Client: class object) = (structure)
MessageText: text;
Response: structure = structure();
begin
    MessageText = call gfr-evaluate-text-proxy(the gfr-prompt-text of
        Alert, gfr-language(Client), Client);
    inform the operator that "[MessageText]";
    return Response;
end
```

Note that it is necessary to return an empty structure from your alert handler. All communication handlers must return a structure, even if they do not add attributes in the structure.

To create an error handler:

1. Clone a gfr-error-handler from the GFR palette and place it in your module.
2. Set the gfr-applicable-class to the name of the class of error that it is going to handle.
3. Edit the procedure, giving it a unique name, and the following signature:

```
my-error-handler(Error: class error)
```

For example, you can create a handler that sends g2-errors to the Message Board.

To send g2-errors to the Message Board, instead of the logbook:

➔ Create a gfr-error-handler named my-error-handler with the following text:

```
my-error-handler(Error: class g2-error);
begin
    inform the operator that "[the text of the error-description of Error]";
end
```

Note There are no return arguments from an error handler.

Using the Call Next Facility

GFR's call next facility gives you a way to add new behaviors to existing communication and error handlers, with a minimum of re-implementation.

Suppose, for example, you want to write all alert messages to a log file, using a procedure named write-to-log-file, and then use the default popup alert. In this case, you create a handler that calls the logging function, and then uses gfr-call-next-communication-handler to generate the popup alert, as follows:

```
log-and-alert-handler(Alert: class gfr-alert, InitiatingItem: class item,
    Client: class object) = (structure)
MessageText: text
Response: structure;
begin
    MessageText = call gfr-evaluate-text-proxy(the gfr-prompt-text of
        Alert, gfr-language(Client), Client);
    call write-to-log-file(MessageText);
    Response = call gfr-call-next-communication-handler(Alert,
        InitiatingItem, Client);
    return Response;
end
```

For communications that define return attributes, such as gfr-confirm, the call to gfr-call-next-communication-handler returns a structure containing one or more attributes. The handler you write must return a structure whose attributes are consistent with the class of communication that is handled.

For example, suppose you want to add an audible beep if a confirm dialog times out before the user responds. Your handler would look like this:

```
confirm-with-timeout-signal(Confirm: class gfr-confirm,
    InitiatingItem: class item, Client: class object) = (structure)
Response: structure;
begin

    Response = call gfr-call-next-communication-handler(Confirm,
        InitiatingItem, Client);
    if the confirmation-result of Response = the symbol TIMEOUT and
        Client is a g2-window then call g2-beep(Client);
    return Response;

end
```

For error handling, the procedure to call the next handler is named gfr-call-next-error-handler. The argument to this procedure is the error object, and there are no return arguments.

For example, if you want to log all G2 error messages to a file, before allowing the next error handler to post the error, you would write the following override handler, where write-to-log-file is a procedure you provide:

```
log-then-pass-error(Error: class g2-error)
begin

    call write-to-log-file(the text of the error-description of Error);
    call gfr-call-next-error-handler(Error);

end
```

4

Localizing KBs

Describes the localization facilities of GFR.

Introduction

The translation of texts in an application is called **localization**. User interface is a critical part of most applications, and text is an essential part of most user interfaces. Most of your KB modules have text components in the form of menus, dialogs, error messages, labels on workspaces, and the like. If you want the user-visible texts in your KB to be easily translatable into other languages, you must prepare your KB as described in this chapter.

To build localizable KBs, do not type text strings directly into the KB. Instead, everywhere you use text, include a "symbolic key" in its place. At runtime, GFR uses these keys, together with the client's language, to look up a language-specific text.

The language of the client is a property of the g2-window where the communication is taking place. GFR stores the language in the attribute of the window called g2-window-specific-language. For more information about g2-window objects, refer to the *G2 Reference Manual*.

Storing Texts in Resource Objects

To contain and group the text strings in your KB, use two types of GFR objects:

- Local text resources
- Text resource groups

Using Local Text Resources

GFR stores the language-specific texts in objects of the class gfr-local-text-resource. A local text resource object looks like the following figure:

A local text resource contains symbol-text pairs that associate a symbolic key with a text in a certain language. Although the data structure used to store the symbol-text pairs is private, you can think of the contents of a local text resource object as a table with two columns and any number of rows:

Key	Text Value
ALERT-MSG-1	"A sample alert message"
ALERT-MSG-2	"Delete this [1]?"
...	...
...	...

To create a gfr-local-text-resource:

➔ Clone it from the palette workspace gfr-top-level, or create it, using the standard G2 create action, in a procedure, rule, or action button.

A gfr-local-text-resource has the following attributes:

Attribute	Description
gfr-language	The language of the texts stored in the resource.
Allowable values:	Any symbol
Default value:	The symbol english
gfr-resource-group	The name of the gfr-text-resource-group that is associated with this object.
Allowable values:	The name of any gfr-text-resource-group
Default value:	The symbol unspecified
gfr-version	A text giving version information about the object.
Allowable values:	Any text
Default value:	The current version of GFR
gfr-file-location	An optional text string giving a file name where the resource can be saved to or loaded from.
Allowable values:	Any text which names a valid file location on your file system
Default value:	" " (the empty string)
gfr-preload-resource	A flag indicating whether a file containing the symbol-text pairs is to be loaded at G2 startup, if the resource is not permanently stored in G2.
Allowable values:	true or false
Default value:	false

Entering Symbol-Text Pairs into a Local Text Resource

The three ways to enter symbol-text pairs into a local text resource are:

- Create an external text file using any text editor, and then load it into the text resource object.
- Load the G2 XL spreadsheet module, which is provided with G2, and edit the text resource from inside G2.
- Programmatically add symbol-text pairs using the GFR API procedure, gfr-add-to-local-text-resource.

The first two options are explained in the following sections. For a description of the programmatic option, see Localization Operations.

Using an External Text Editor to Edit a Local Text Resource

GFR supports editing of local text resources outside of G2. This design enables someone who does not know G2 to translate your KB into a new language, using any word processor or text editor. Then, you can load the translation into G2 with almost no effort.

To prepare a new local text resource

➔ Create a text file.

The first three lines of the text file must contain the following, with each item on a separate line:

- The resource group name, as a symbol.
- A line of version information which may help you identify the file, as a quoted text.
- The language, as a symbol.

Starting on the fourth line of the file, type the symbol-text pairs, separated by a comma, one pair per line. Enclose the texts following the symbol keys in double quotation marks.

Note Do not type carriage returns in the body of the texts.

For example, the text file could look like this:

```
MY-ALERT-TEXTS
“Version 2015"
ENGLISH
ALERT-MSG-1, "A sample alert message"
ALERT-MSG-2, "Delete this [1]?"
```

For clarity, symbols are in capital letters, but this is not necessary.

If you have embedded quotation marks in any texts, use two sets of double quotation marks. For example, if the text to be loaded is abc "def" geh, represent this in the file as "abc ""def"" geh".

You can use any special or foreign characters that are part of the Gensym character set, as described in *G2 Reference Manual*. For example, the following text file is for English and includes a newline character, the copyright symbol, and an accented character for the word "café:"

```
MY-ALERT-TEXTS
"Version 2015"
ENGLISH
ALERT-MSG-1, "The first line.@LThe second line."
ALERT-MSG-2, "Copyright ~| 2015"
ALERT-MSG-3, "Caf~e"
```

To load the file contents into a local text resource object:

1. Start G2 and clone a local text resource from the palette workspace named gfr-top-level.
2. Edit the table of the local text resource object:
   a. Change the gfr-file-location attribute to the path of the file you want to load.
   b. Change the gfr-resource-group attribute to the resource group named in the file (MY-ALERT-TEXTS, in the example).
3. On the menu for the object, choose load text resource. This will load the data from your file, replacing whatever was previously stored.

Note The menu choices for loading and saving the resource to a file only appear when the gfr-file-location attribute is specified.

Using the G2 XL Spreadsheet to Edit a Local Text Resource

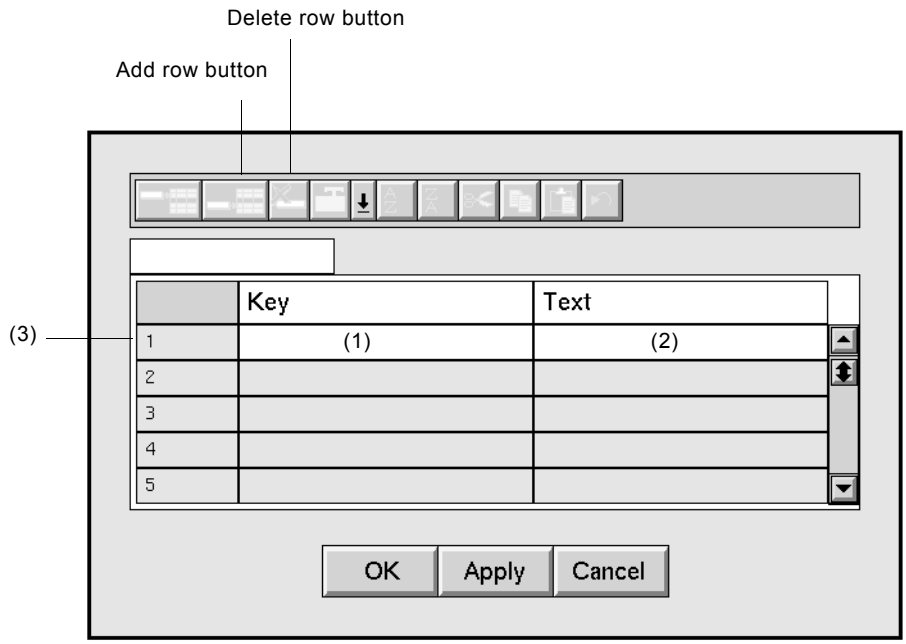
The G2 utility G2 XL Spreadsheet provides a convenient way to edit local text resources from within G2. GXL is located in the file named *gxl.kb* in the *utils* subdirectory in the *kbs* directory under the *g2* directory.

To use a spreadsheet to edit a local text resources:

1. Merge the module gxl into your KB.
2. Get the workspace named gxl-top-level and click the check box labelled array and list editing on this workspace.

   When you select array and list editing, G2 adds a new menu choice, edit resource to the menu of local text resource objects.

3 Choose edit resource to see the following spreadsheet workspace (the numbered callouts are referenced in the following procedures):



To enter the first key-text pair:

1 Click on the cell labelled (1) and type the desired symbolic key and press Return when you are finished.

2 Click on the cell labelled (2) and type the text. Do not use enclosing quotation marks when entering texts in the spreadsheet.

To enter special characters:

➔ Use keystroke commands, as described in the *G2 Reference Manual*.

For example, to enter a carriage return, type Ctrl + j.

To add a second key-text pair.

1 Click on the cell labelled (3) to select the first row of the spreadsheet.

2 Click the add row below selection button (second from the left on the toolbar) to add an empty row below the first row.

3 Repeat this process to add as many key-text pairs as you like.

When you exceed five rows, a vertical scroll bar appears on the right side of the spreadsheet to allow you to access rows not shown on the spreadsheet.

You can also delete a row by selecting the row and choosing the delete button (third button from the left on the toolbar).

When you are finished entering data, click the OK button. This will save the values you have entered into the local text resource. If you do not want to save your edits, click Cancel.

For information on sorting, cutting, pasting and other spreadsheet functions, see the *G2 XL Spreadsheet User's Guide*.

When you are finished editing resources, you can delete the spreadsheet module.

To delete the spreadsheet module:

➔ Choose Main Menu > Miscellany > delete module > gxl.

Choose the All option to also delete all workspaces in the module.

Storing Local Text Resources

You have several options on how G2 stores key-text pairs in a local text resource. The options are:

- Store the values permanently in the KB.
- Store the values permanently in a file and load them each time G2 is started.
- Store the values permanently in a file and load them only on demand.

Storing Local Text Resources in a KB

If you store the values permanently in a KB, you have the convenience of not having to manage auxiliary files, because all information is stored in G2. However, there is a certain memory penalty for storing the information in G2 on a permanent basis. The penalty is roughly about 200 bytes per key-text pair, although this depends on the length of the texts.

To store the information in a local text resource as a permanent part of a KB:

➔ Choose make resource permanent from the menu of the resource or call gfr-make-local-text-resource-permanent.

For more details, see Application Programmer's Interface.

Loading a Local Text Resources File at Startup

If you choose to load local text resources at G2 startup, the memory requirement is reduced to about 100 bytes per key-text pair, depending on the length of the texts. However, starting up G2 is slower because the file is loaded when G2 is started. Also, you must make sure that the path to the file named in gfr-file-location is always valid.

To load local text resources at G2 startup:

➔ Specify the file name in the gfr-file-location attribute and set the gfr-preload-resource attribute to true.

Loading a Local Text Resources on Demand

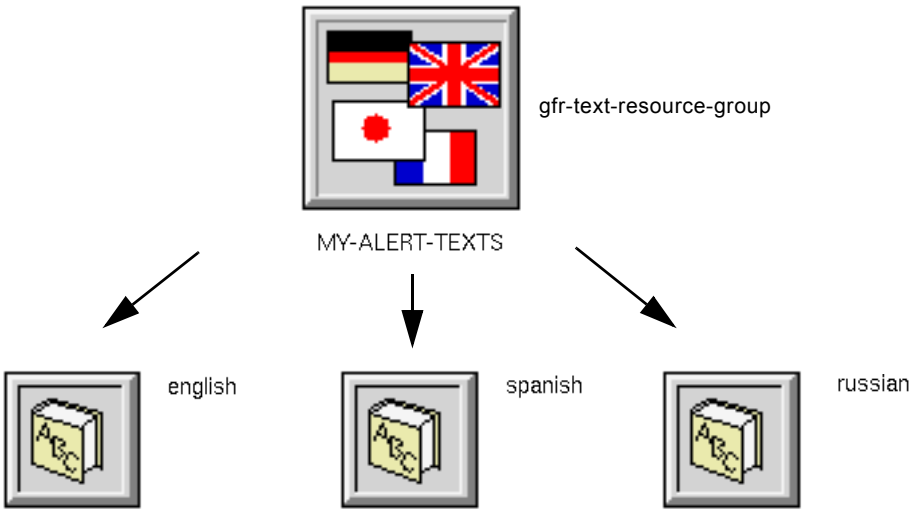
If you choose to load the resource only on demand, you save 100% of the memory for the local text resources that are not used during a G2 session. If you are supporting several languages, and only one is currently in use, then this option saves you from loading unused languages. However, the first time any resource is demanded, CPU time is allocated to loading the file.

To load local text resources on demand:

➔ Specify the file location and set gfr-preload-resource to false.

Using Text Resource Groups

Each local text resource is associated with exactly one gfr-resource-group, which serves to link resources in different languages. All local text resources associated with a resource group contain the same keys but different languages, as shown in the following figure:



Within a KB, you can have as many resource groups as you like. Usually, each module supplies its own resource group or groups.

To create a text resource group:

➔ Clone it from the palette workspace gfr-top-level, or create it, using the standard G2 create action, in a procedure, rule, or action button.

Gensym recommends that, within a module, you divide texts according to their purpose and have multiple resource groups, rather than putting all the texts in one resource group. This both improves access time and makes it easier to organize your texts. For example, you might make a resource group for error messages, another for dialog labels, and another for texts appearing on menus.

Caution You must not have more than one local text resource with the same language in a given resource group.

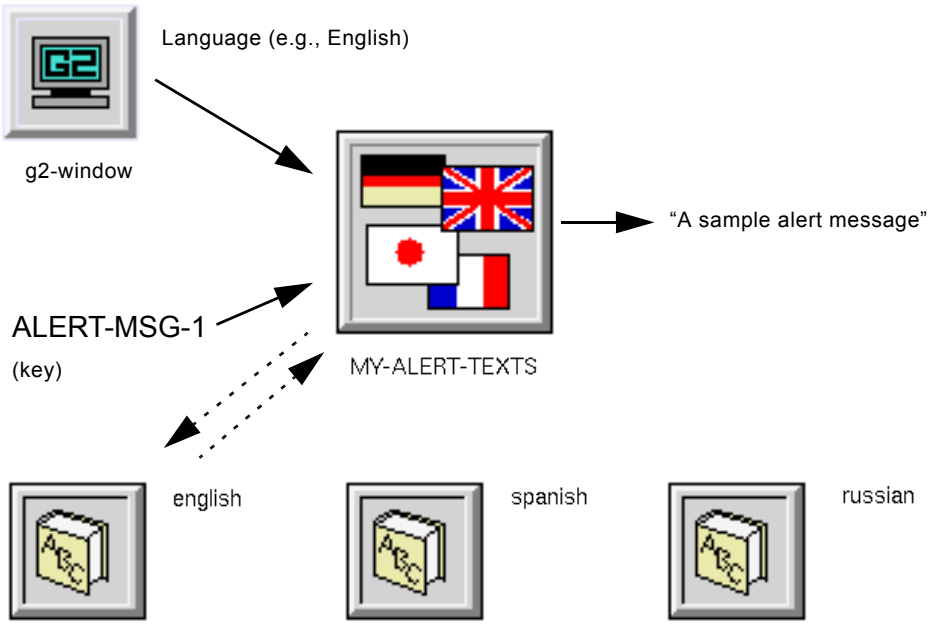
The attributes of a text resource group are as follows:

Attribute	Description
gfr-version	A text giving version information about the object.
Allowable values:	Any text
Default value:	The current version of GFR
gfr-default-language	Indicates which language to use when the requested language is not supported or unspecified, or if the key is not found in the requested language.
Allowable values:	Any symbol
Default value:	The symbol english
gfr-use-default-language	A flag indicating whether the default language is to be used.
Allowable values:	true or false
Default value:	true

For a description of the use of default languages, see Using Default Languages.

Accessing Localized Texts

At run time, use the key, the resource group, and the window-specific language to retrieve a localized text. The following figure shows the flow of information:



Typically, you do not interact directly with local text resources when retrieving localized texts. Instead, you interact with the resource group, and the resource group locates the appropriate local resource and retrieves the text corresponding to the given key.

The following line of code shows how you would access a localized text in the English language, corresponding to the example above:

```
LocalText = call gfr-localize-message(my-alert-texts, the symbol alert-msg-1,
    gfr-language(Win), Win);
```

In this example, Win is the window where the call originates, and the text is to be rendered in the language of that window. The function gfr-language retrieves the window-specific language. This function returns the current default language when the window-specific language is unspecified.

For more information on gfr-localize-message and gfr-language, see Application Programmer's Interface.

Using Text Substitutions

GFR provides for text substitutions within localized texts. Text substitution increases the flexibility of the text facility. You can substitute text when the text you want to display contains run-time information, such as the value of a variable, the name of an item, and so on.

GFR uses bracketed integers, such as [1], [2], [3], to indicate where substitutions should be made within a local text. For example, in ALERT-MSG-2, "Delete this [1]?", you can substitute the [1] with the class of the item being deleted, as follows:

```
LocalText = call gfr-localize-message(my-alert-texts, the symbol alert-msg-2,
    gfr-language(Win), the class of Foo, Win);
```

If the class of Foo is tank-with-one-input, then LocalText will have the value "Delete this tank-with-one-input?", assuming the language of the window is English.

Tip You can use the localization facility to "pretty print" or localize class or attribute names. For example, you could provide a local language version or a more readable English text representation for tank-with-one-input before substituting it into the alert message.

You can have as many as 10 different substitutions within a single message, and a given substitution can repeat more than once. Substitutions can be any value (text, symbol, quantity, or truth-value).

You use the procedure gfr-localize-message to generate the localized text, regardless of the number of substitutions. After the third argument to gfr-localize-message, you can type the substitution arguments as given. For example, the call to gfr-localize-message with three substitution arguments would have the form:

```
LocalText = call gfr-localize-message(group-name, key, language,
    substitution-1, substitution-2, substitution-3, window);
```

Note GFR does not signal errors if you provide too few or too many substitution arguments. Substitutions that are possible are made, and the remainder are ignored.

Using Text Proxies

In addition to providing access to texts, GFR provides:

- Object classes – To embed a localizable text into an object.
- Message classes – To display localizable messages on a workspace.

The object classes are called *text proxies*. You use a text proxy when an object has an attribute in which you would normally put a text, if you were not concerned about localization. Instead of hard-coding the text, you introduce the text proxy as subobject. GFR evaluates the text proxy at run-time to yield a localized text.

There are two classes of text proxy, gfr-simple-text-proxy and gfr-text-proxy, whose attributes are given below:

Attribute	Description
gfr-text-resource-group	Names the resource group that contains the key-text pair needed to evaluate this text proxy.
Allowable values:	The name of any gfr-text-resource-group
Default value:	The symbol g2
gfr-message-name	The key used to retrieve the message from the resource group.
Allowable values:	Any symbol
Default value:	The symbol english
gfr-substitutions (gfr-text-proxy only)	A list containing values to be substituted into the text.
Allowable values:	Any values
Default value:	none (an empty list)

For example, suppose you have a class of buttons with an attribute called label, containing the text used in the label of the button. To make the button localizable, instead of putting the text of the label directly in the attribute, define the label attribute to be an instance of a gfr-simple-text-proxy.

The text proxy names the resource group and gives the message name that provides the button label. At run time, you make the following call to get the localized text of the button, Button:

```
Label = call gfr-evaluate-text-proxy(the label of Button, gfr-language(Win), Win);
```

Typically, you would make this call just before the button is displayed on a window, when the language of the window is known.

Note Although text proxies provide a convenient interface, they introduce the memory overhead of an additional object. If memory is a concern, you may want to add two symbolic attributes to the object, one naming the resource group and the other giving the key, instead of using a text proxy.

The class gfr-simple-text-proxy does not support substitution arguments. If you want to use substitutions in an embedded object, you use the class gfr-text-proxy, and insert the substitutions into the list gfr-substitutions, which is an attribute of this class.

GFR provides the API procedure gfr-configure-text-proxy as a convenient way to set up a text proxy before it is evaluated. For a simple text proxy, the call to gfr-configure-text-proxy sets up the resource group and message name. For a gfr-text-proxy, the same call sets up the resource group, message name, and the substitutions. See Application Programmer's Interface for details.

Using Localizable Message Classes

GFR provides two classes of localizable messages that you can use where you want visible text on a workspace. The classes are gfr-localizable-message and gfr-

simple-localizable-message. The following table summarizes the attributes of these classes:

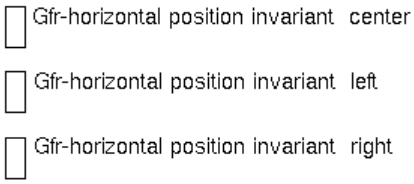
Attribute	Description
gfr-id	An identifier of the message.
Allowable values:	Any text
Default value:	" "(the empty string)
gfr-horizontal-position-invariant	The horizontal position of the message that is maintained if the width of the message changes when it is rendered in a specific language.
Allowable values:	One of the following symbols: left, right or center
Default value:	The symbol center
gfr-vertical-invariant	The vertical position of the message that is maintained if the height of the message changes when it is rendered in a specific language.
Allowable values:	One of the following symbols: top, bottom or center
Default value:	The symbol center
gfr-text-proxy	A text proxy used to produce the text of the message.
Allowable values:	An instance of a gfr-text-proxy (for a gfr-localizable-message) or a gfr-simple-text-proxy (for a gfr-simple-localizable-message)
Default value:	An instance of a gfr-text-proxy (for a gfr-localizable-message) or a gfr-simple-text-proxy (for a gfr-simple-localizable-message)

Example

This is an example of how to use localizable messages. On the workspace shown in the following figure, you see three localizable messages with different horizontal invariants. In each case, the vertical invariant is center. The border color of messages, normally transparent, has been set to black to see more easily.

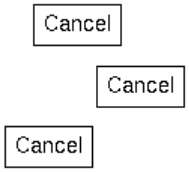
For the purposes of this example, the text proxies of the messages have been configured to use one of the messages in gfr-test-messages, on the examples workspace in GFR.

The workspace initially has three empty messages, as shown in the following figure:



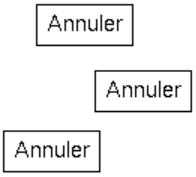
If the language of the window is English, and we make the following call. The results appear in the following figure:

```
call gfr-localize-messages-on-workspace(ws1, gfr-language(Win), Win);
```



Hint The default font size of GFR's localized message classes is large. To set the font size and magnification of a message to another value, use the G2 system procedure g2-set-font-of-text-box.

To translate the messages, change the language of the window to French and repeat the gfr-localize-messages-on-workspace call. The results appear in the following figure:



Tip If you support window-by-window localization in your application, design your application so that user interface workspaces containing text are never simultaneously shown on more than one window. This is not necessary if every window connected to a given G2 is in the same language.

Using Default Languages

When the window-specific language of the g2-window passed to the function gfr-language is not specified, GFR uses G2's language parameters system table as its global default language.

To set the global default language:

➔ Edit the current-language attribute of the language parameters system table.

You can also specify a default language for each gfr-text-resource-group by setting the attribute gfr-default-language. You use this language when no local text resource exists for the language requested in a call to gfr-localize-message. If the local text resource for the default language is not found, an empty string is returned.

You can also prevent the use of a default language by setting gfr-use-default-language of a resource group to false. If you do not use a default language, the empty string is returned if the local text resource for the requested language is not found.

5

Managing Palettes

Discusses how to use the palette creation utilities provided by GFR.

Introduction

Palettes are a common user interface device in G2 applications. A palette is a workspace from which users can clone items to use in a KB they are developing.

As a method of creating instances, palettes have several advantages over the New Object command on the KB Workspace menu:

- Objects on palettes can be initialized in any way required by the application.

  When you use the New Object menu, the object you obtain might not be initialized properly. For example, an object created in this manner cannot have a subworkspace.

- Palettes immediately present the objects that the user is meant to see.

  Use of the New Object menu could require complex navigation through many menu levels to find the desired class. The names of the classes that must be traversed might be unfamiliar to the user. Additionally, navigation through the class hierarchy might require traversal of private classes, which should be hidden from the end user.

- Palettes allow users to identify visually the object they want to create, even if they cannot remember the class name.

  Users recognize classes by their icons more easily than by their class name.

Because of these advantages, palettes have become a standard way to present items to the user, and they are found in many KB modules.

Standardizing Palette Creation and Management

You have several reasons to standardize on a single palette creation and management system, aside from the basic inefficiency of having different software development groups building and maintaining their own palette management systems. Some of the reasons for adopting a standard include:

- It is not as straightforward as it seems to create a good palette management system.

  Although the item configuration selecting any X implies clone implements a basic click-to-clone behavior, you cannot use this configuration statement on proprietary workspaces.

- When you create a palette using item configurations, no clean way exists to prevent the user from transferring the cloned item back to the palette, intentionally or accidentally.

  Many palette systems involve whenever any item is moved rules to handle this contingency, but such rules impose an efficiency drag on the entire KB because they trigger on *all* item moves, not just the moves to palette workspaces.

- To enhance usability, a single standard set of mouse gestures should exist for cloning items from palettes.

GFR provides a standard, easy-to-use system for creating palettes. Because it is based on mouse tracking rather than item configurations, it improves upon most existing G2 palette systems in several ways:

- You can make GFR's palettes proprietary to prevent a user from altering the palette.
- The GFR palette system does not require the user to write complex item configuration statements.
- There are no whenever rules dealing with object movement in the GFR palette system, which might interfere with the efficiency of the KB.
- You can disable items on GFR palettes to make them invisible to rules and procedures within G2, so they do not interfere with the functioning of "real" items in your KB.
- GFR provides optional bubble help on palette items.

Implementing Palette Behavior for Items

Although palettes seem like a special type of workspace, GFR implements palette behavior on individual items, not entire workspaces. Not all items on a palette workspace, such as labels, titles, or hide buttons, should have click-to-clone behavior.

Caution Palette behavior is implemented by placing a transparent object of the class gfr-palette-window over the palette item. You must never create, clone, delete, or otherwise manipulate palette windows except through GFR menu choices or API procedures.

If a mouse gesture can clone an item, the item has **palette behavior**. When you use GFR to add palette behavior to an item, you get the following behavior:

- When you press a mouse button over a palette item, bubble help (if defined) appears.

  If you move the mouse to another palette item without releasing the mouse button, GFR dismisses the bubble help for the first item and displays the bubble help for the second item.

- When you click on a palette item (or release the mouse over the original item after viewing bubble help), a new instance of the object is created and attached to the mouse.

  For details on how this object is created, Understanding How Items are Created from a Palette.

- When you click the mouse again, the instance is deposited wherever the mouse happens to be at the time.

  If you are over the background or a proprietary workspace, the instance is deleted.

- If you deposit the item back onto the original workspace, the new instance is deleted, and the workspace is shrinkwrapped.

- If you have successfully placed the object instance on a workspace, GFR calls the method gfr-initialize if the item is a gfr-item-with-uuid.

  You can use the gfr-initialize method for any purpose appropriate to your application.

These behaviors occur when the workspace that contains the palette item is proprietary, or when you are not in Administrator mode and the workspace containing the palette item is not proprietary.

When you are in Administrator mode and the workspace containing the palette item is not proprietary, selecting the palette item displays the menu of the palette window.

If you drag the palette window, the palette item "snaps" to the new location of the palette window after the move. For more information on user modes and making workspaces proprietary, see the *G2 Reference Manual*.

Palette windows cover the item, its stubs, and its attribute displays.

Note When you are in Administrator mode on a non-proprietary palette, use the clone palette item menu choice to create a new instance of the palette item. To display the clone palette item menu choice, select the palette window covering the item of interest.

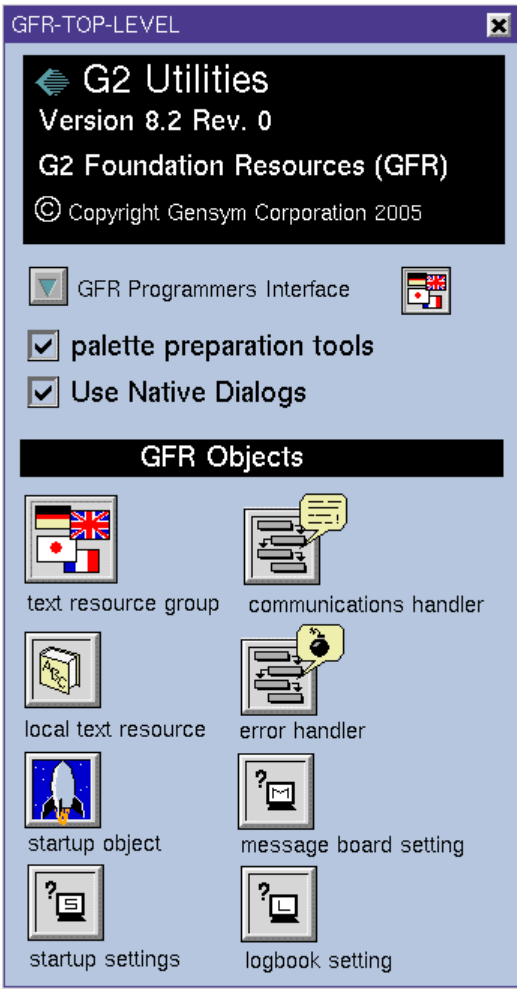
Adding Palette Behavior to an Item

Using GFR's palette preparation tools, you can add palette behavior to G2 items.

Note G2 must not be paused or reset when you perform these actions.

To activate menu choices related to palette preparation:

1 Get the workspace gfr-top-level and click the check box labelled palette preparation tools, as shown in the following figure:



2 Alternatively, you can turn on palette preparation programmatically by concluding that the logical parameter named gfr-palette-tools-on is true.

The following user menu choices appear on G2 items when GFR palette preparation is active:

This menu choice...	Appears on...	And does this...
add palette behavior	All G2 items except workspaces and connections, if the item does not already have palette behavior	Adds palette behavior to the item
remove palette behavior	Any G2 item that has palette behavior	Removes palette behavior and returns the normal behavior to an item
show palette windows	Workspaces that contain any item with palette behavior	Outlines all palette windows on the workspace
hide palette windows	Workspaces that contain a visible palette window	Makes the palette windows on the workspace invisible

To add palette behavior to an item:

➔ Choose add palette behavior from the item menu.

If you are not in Administrator mode, you immediately get the palette behavior described in the table. You can add palette behavior to items that are disabled. You cannot add palette behavior to items on proprietary workspaces.

To remove palette behavior from an item on a non-proprietary workspace:

1 Switch into Administrator mode.

2 Choose remove palette behavior from the menu of the palette window over the item.

Caution You must never delete an item with palette behavior without first using the remove palette behavior menu choice to return the item completely to normal G2 object behavior. Otherwise, there may be invisible connection stubs left on the item. You can also use the API call, gfr-remove-palette-behavior-from-item.

To view the items on a workspace that have palette behavior:

➔ Choose the show palette windows workspace menu choice.

This menu choice appears only if there are palette items on the workspace. The palette window covers the item, its stubs, and its attribute displays. Be sure to hide the palette windows, using the hide palette windows menu choice when you are done.

Note You cannot manually resize a palette window or an item with a palette window, without causing a misalignment between the palette window and the item it is covering. If a misalignment occurs, you must remove the palette behavior and add it again using the menu choices provided.

Understanding How Items are Created from a Palette

When the user selects a palette item, GFR creates an item and transfers it to the mouse. The item is instantiated in one of two ways:

- If the palette item does not have a subworkspace, GFR creates a new instance of the object by calling the method gfr-create-instance-from-palette-item, which you can optionally provide.

  If you do not provide an overriding method, GFR uses the default method of creating an instance from the object definition of the class, using the native G2 create action.

- If the palette item has a subworkspace and an item of the same class as the palette item exists on this subworkspace, GFR creates the new item by cloning the item on the subworkspace.

Note The item created from the palette is never a clone of the palette object itself. It is either an object created via the gfr-create-instance-from-palette-item method or a clone of the item on the subworkspace of the palette item. This allows the palette to be proprietary and the palette item to be disabled, as discussed in the following sections.

To create an instance programmatically:

➔ gfr-create-instance-from-palette-item
(*item*: class item, *window*: class g2-window
-> *instance*: class item)

where:

item: The palette item.

window: The g2-window of the client.

In your method, you create, configure, and return a new object that is attached to the user's mouse when you press the button over the palette item. When you write this method, you must include the class of the palette item in the declaration, not class item.

If the palette item is disabled, your gfr-create-instance-from-palette-item method must be able to refer to inactive items. To allow your method to refer to inactive items, set this property of your method using the following syntax:

```
conclude that the may-refer-to-inactive-items of the evaluation-attributes
    of your-method = true
```

Cloning the Palette Item

The second technique of creating an item allows you to control the creation of the palette item without writing methods. For example, if you want the cloned palette item scaled differently than its defined icon size, follow these steps:

1. Create two instances of the item you want to appear on the palette and change their sizes to the desired dimensions.
2. Create a subworkspace under one of the items.
3. Transfer the other item to the subworkspace.
4. Optionally, disable the superior item.
5. Add palette behavior to the superior item.

When the user presses the mouse on the palette item, GFR clones the item found on the subworkspace of the palette item and transfers it to the mouse. Do not disable the subworkspace items.

Handling Complex Initialization Requirements

To accommodate more complex initialization requirements, such as rotating or connecting the new object when it arrives on the destination workspace, you can provide an initialization method called gfr-initialize.

GFR calls the gfr-initialize method immediately after the user deposits a new ID-bearing item (either gfr-object-with-uuid or gfr-message-with-uuid) onto a workspace.

The signature for this method is:

gfr-initialize
(*item*: class item, *client*: class object)

where:

item: The item to be initialized. When you write this method, you must include the class of the item you want to initialize in the declaration, not class item.

client: An object representing the source of the call, typically a g2-window.

Caution If you place an item created from a palette directly onto a disabled or inactive workspace, GFR does not call the gfr-initialize method.

Configuring Palette Workspaces

After you have added palette behavior items on a workspace, you can specify the item configuration of the workspace. For example, you may not want the user to move and edit free text labels or to add and remove palette behavior from items.

The following are typical item configurations for a palette workspace:

```
configure the user interface as follows:
    unless in administrator mode:
    menu choices for workspace include hide-workspace;
    menu choices for borderless-free-text include: nothing;
    non-menu choices for item exclude additionally:
    move-object, click-to-edit
```

These restrictions prevent the user from moving and editing borderless free texts and they also suppress menu choices, other than hide-workspace, from the workspace. Of course, you may add any item configurations you wish.

Special Considerations for Proprietary Palettes

GFR gives you the ability to create proprietary palettes. When you make a workspace containing proprietary palette items, you can make the palette impervious to accidental damage by the user, even when the user is in Administrator mode.

If you are using the create-by-cloning technique, and palette items have subworkspaces, these subworkspaces must be explicitly marked as not proprietary during package preparation.

When you create a proprietary palette, you should include restrictions on the workspace similar to the previous example plus the following restrictions on these proprietary items:

```
restrict proprietary items as follows:
    menu choices for workspace include:hide-workspace;
    menu choices for borderless-free-text include: nothing;
    non-menu choices for item exclude additionally:
    move-object, click-to-edit
```

Adding Bubble Help to Palette Items

For palette items, GFR provides optional bubble help, also known as a tooltips.

To add bubble help to a palette item on a non-proprietary workspace:

1. Add palette behavior to the item, as described in Adding Palette Behavior to an Item.
2. In Administrator mode, click on the palette item to display the palette window menu.
3. From the palette window menu, select Table.
4. Change the value of gfr-help-message-name to the key for the bubble help text.
5. Change the gfr-help-text-resource to the name of the gfr-text-resource-group that provides the text of the bubble help.

Note You must use the GFR text translation facility when you use bubble help.

The attributes in the table for the palette window menu are:

Attribute	Description
gfr-help-message-name	The key of the help message in the local text resource.
Allowable values:	Any symbol
Default value:	unspecified

Attribute	Description
gfr-help-text-resource	The name of the text resource that specifies the text of bubble help.
Allowable values:	Any symbol naming a text resource for bubble help
Default value:	unspecified

6

The Universal Unique ID System

Discusses how GFR generates and manages universal unique identifiers (UUIDs).

Introduction

It is often desirable to tag items in a KB with a permanent, unique identifier (ID). Like a pointer, a unique ID provides a persistent handle to an item that can be stored in other objects that reference the identified item. A unique ID provides a way for the KB developer to build arbitrarily complex data structures involving multiple items.

GFR's unique ID management system also provides a mechanism for detecting and managing creation of ID-bearing objects by the user. When you create or clone an ID-bearing item, the ID management system calls the methods gfr-initialize and/or gfr-copy. These methods enable you to initialize the object created by the user.

Unique ID Format

GFR offers a standard ID generator that generates a unique ID. The ID generated by GFR is unique not only to the KB, but also is *universally unique*: The ID is not duplicated in any KB, anywhere in the world, at any time, past or future, unless the KB file is copied or the same KB is loaded into another G2. Therefore, Gensym calls it a Universal Unique ID (UUID).

UUIDs 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.

Note In addition to GFR's unique ID management system, G2 offers its own unique-identification mixin class that adds a universal unique identifier (UUID) attribute to new or existing classes. You can add a UUID to the instances of a subclass by including the unique-identification mixin class in the subclass's inheritance. G2 implements UUIDs as indexed attributes. Both the G2 UUID and the GFR UUID have the same format, as described in this section. For more information on the G2 UUID facility, see the *G2 Reference Manual*.

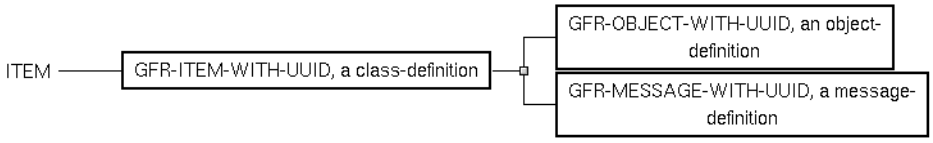
To generate a unique ID:

➔ gfr-universal-unique-id
( )
-> *uuid*: text

Even if you use no other parts of GFR's ID mechanism, you can use gfr-universal-unique-id to generate IDs.

Inheriting Classes with Universal Unique IDs

GFR provides two classes, gfr-object-with-uuid and gfr-message-with-uuid, that have a universal unique ID attribute. The classes inherit from the gfr-item-with-uuid class, as shown in the following figure:



You can use multiple inheritance from these classes into your classes that require IDs. The two classes are:

- gfr-object-with-uuid, which any *object* class can inherit.
- gfr-message-with-uuid, which any *message* class can inherit.

Note GFR does not provide a connection class with ID.

The universal unique ID attribute of gfr-object-with-uuid and gfr-message-with-uuid is summarized in the following table:

Attribute	Description
gfr-uuid	An indexed attribute containing the universal unique ID of the item.
Allowable values:	Any UUID generated by gfr-universal-unique-id
Default value:	"" (the empty string)

Referencing an Item through its UUID

Because the gfr-uuid is an indexed attribute, you can use index-attribute lookups to efficiently locate an item using its UUID as a handle. For complete information about indexed attributes, see the *G2 Reference Manual*.

For example, if the UUID of the object we are trying to find is MyID, you can use the following statement to find it:

```
if there exists a gfr-object-with-uuid Obj such that
    (the gfr-uuid of Obj = MyID) then
begin

    {Insert statements dealing with Obj here}

end
```

To find gfr-messages-with-uuids, use an analogous statement. You can also add other criteria into the existence check by extending the such that clause.

Using the ID Management System

The job of the ID management system is to assure that new instances of the classes gfr-object-with-uuid and gfr-message-with-uuid are assigned IDs whenever they are created. As long as G2 is not reset, GFR detects manual actions such as creating and cloning and maintains unique IDs for all ID-bearing objects and messages. The actions detected by GFR include:

- Creation of objects by selecting New Object or New Free Text from the KB Workspace menu, or selecting the create-instance menu choice of a class definition.
- Creation of items by manual cloning, including items cloned from a palette, and items created by cloning with the operate on area tool.
- Creation of objects by workspace cloning, including objects created by cloning an object with a subworkspace or cloning a workspace containing objects with subworkspaces.

The ID management system does not update IDs of items that are attributes of other items. If you define a class that has an instance of an ID-bearing item as an attribute, you must provide a gfr-initialize method for the class that assigns an ID to the subobjects. For more information on the gfr-initialize method, see Using the gfr-initialize Method.

Creating ID-Bearing Items Programmatically

When you programmatically create an instance of classes gfr-object-with-uuid or gfr-message-with-uuid, this action is not detected by GFR. You must make sure that the object you create is assigned a new ID. You do this by calling the method gfr-initialize on the object you have created.

Two fundamentally different create actions exist in G2:

- Simple Create
- Create by Cloning

Simple Create

If foobar is a subclass of either ID-bearing class, the first case can be handled straightforwardly, as follows:

```
create a foobar Foo;
call gfr-initialize(Foo, Client);
```

where Client is any client object, usually a g2-window.

If the class has an attribute that is an instance of an ID-bearing class, the author of the class must provide a gfr-initialize method to assign an ID to the subobject.

Create by Cloning

The programmatic clone action is more involved. When you create an item by programmatically cloning another item, the possibility exists that the item you are cloning might have a subworkspace containing other items. If the items on the subworkspace bear IDs, they also must be assigned new IDs.

GFR provides a procedure that checks the subworkspace hierarchy of the cloned item and appropriately initializes the IDs by dispatching calls to gfr-initialize and

gfr-copy, as described in the following sections. The name of this procedure is gfr-check-uuids-on-cloned-item.

For example, if Bar is an item that has a subworkspace that could contain ID-bearing objects, use the following sequence for cloning the item:

```
create a item Bar2 by cloning Bar;
call gfr-check-uuids-on-cloned-item(Bar2);
```

Bar itself may or may not be an ID-bearing object.

You must also use this cloning procedure when you are programmatically cloning a workspace, as in the following example:

```
create a kb-workspace WS2 by cloning WS1;
call gfr-check-uuids-on-cloned-item(WS2);
```

The call to gfr-check-uuids-on-cloned-item assures that all ID-bearing items upon WS2 and in the workspace hierarchy of objects upon WS2 are correctly initialized with new IDs.

Using the gfr-initialize Method

The GFR ID-management system uses the method gfr-initialize to give new instances unique IDs. You can specialize the gfr-initialize method to give initialization behavior to your classes whenever an instance of your class is created.

Here are some of the ways you could use the gfr-initialize method:

- To validate whether the object has been placed on the right type of workspace, or in the correct module.
- To establish relations that must apply to each item of a given class.
- To create additional objects needed in a structure, such as populating a matrix object with arrays representing the matrix's rows, when the item-array is created.
- To launch a procedure invocation associated with the object that could monitor or animate the object.

The signature of the gfr-initialize method is as follows:

gfr-initialize
(*item*: class item, *client*: class object)

where:

item: The item to be initialized, which is an ID-bearing object.

client: The g2-window of the client requesting the initialization, or gfr-default-window if no appropriate window is available.

Caution When you create your own gfr-initialize method, you must use a call next method statement somewhere in your method.

As long as GFR is running, gfr-initialize is called automatically on each new ID-bearing object you manually create. The calls are automatically made when you are performing manual actions such as creating objects using G2's New Object menu or new instance menu choice, and cloning objects or workspaces.

These calls to gfr-initialize are asynchronous and lag the actual creation of the object by some small amount of time. By default, IDs are updated as a priority 3 task. When you create or clone objects programmatically, GFR does not call the gfr-initialize method automatically; you must call gfr-initialize or gfr-check-uuids-on-cloned-item, as described in this and the previous section.

Using the gfr-copy Method

When the user clones a gfr-object-with-uuid or gfr-message-with-uuid and the object is initialized, the ID management system immediately calls a second method called gfr-copy. When the default G2 clone does not perform the clone action required by your data structures, the gfr-copy method can be used for specializing the native G2 clone action.

For example, you want to clone a matrix (an item array whose elements are float arrays) and copy the contents of the float arrays into the cloned matrix. You also want the contents of the cloned matrix and the source matrix to be identical. The G2 clone action does not assure that the contents are identical.

Therefore, when you define your matrix class, you should inherit from gfr-object-with-uuid and appropriately define gfr-initialize and gfr-copy methods.

The signature of gfr-copy is as follows:

```
gfr-copy
    (source-item: class gfr-item-with-uuid,
     cloned-item: class gfr-item-with-uuid)
```

where:

source-item: The item that was cloned.

cloned-item: The item created by cloning.

When you create a gfr-copy method for your class, you must replace the class name in the declaration of the method to your class name.

Validating UUIDs

The GFR ID management system only works when G2 is not reset. If the user creates objects by instantiation or cloning while G2 is reset, the objects do not automatically receive unique IDs, which is a corruption of the ID management system. Because of this possibility, whenever G2 is started, GFR validates all ID-bearing objects and messages.

During validation, the ID management system checks for duplicate IDs. By default, the ID management system alerts you if it detects items with duplicate IDs. The alert message contains the ID of the duplicate objects, allowing you to use the Inspect facility to locate them.

When duplicate IDs are detected, it is usually an indication that:

- An ID-bearing object was cloned while G2 was reset.
- A programmatic create or clone action was performed without calling:
  - gfr-initialize (see Using the gfr-initialize Method).
  - gfr-check-uuids-on-cloned-item (see Create by Cloning).

Depending how IDs are used in the modules required by your KB, it may be unsafe to have items in your KB with duplicate IDs. Therefore, the message posted by GFR recommends that the user delete the objects with duplicate IDs before continuing.

You can override the warning issued by GFR and handle invalid IDs yourself by providing a method named gfr-handle-invalid-uuid. The signature of this method is as follows.

gfr-handle-invalid-uuid
(*obj*: class gfr-item-with-uuid, *client*: class object)

where:

obj: The object with duplicate IDs.

client: The g2-window of the client or gfr-default-window if no appropriate window is available.

Normally, the method you provide should change the ID of the item in question or delete the item.

7

Additional GFR Utilities

Discusses file parsing and other GFR utilities.

Introduction

GFR provides a small number of additional utilities dealing with parsing, file operations, and the coordinates of item edges. This chapter summarizes these utilities.

File Parsing

GFR provides two utility that help you load data from files into G2. While G2 already provides a set of system procedures for reading and writing files, these utilities work only with text strings.

Putting together a line of text for writing to a file is usually straightforward. Do this using G2's text concatenation operations. However, reversing the process by parsing lines of text that have been read from a file is a difficult problem many developers face.

While GFR does not solve the parsing problem in its most general form, it does provide a utility that converts the contents of a file into a value list containing G2 value types, namely floats, integers, symbols, texts, and truth-values. Once a file has been converted into a value list, further text parsing is usually unnecessary.

Thus, GFR removes some of the difficulty of loading file data into G2. GFR uses this utility to read text resource files.

The two procedures GFR uses to read file data are gfr-load-file-into-list and gfr-parse-string-into-value-list. These procedures are described in File Parsing and Miscellaneous Functions and Procedures.

To convert a list of values into a string appropriate for writing to a file in a format compatible with these utilities, use gfr-convert-value-list-to-string.

Item Edge Position Functions

GFR provides functions that return the workspace coordinates of the left, right, top and bottom of an item. Although G2 provides the width or height of an item and its workspace position, it does not have a built-in function to determine the edge positions of an item.

The correct calculation of the edges is a little trickier than it seems, since you must account correctly for fractional pixels if the height or width of the item is an odd number. The functions gfr-left, gfr-right, gfr-top and gfr-bottom account correctly for the rounding.

Part III

API Procedures and Functions

Chapter 8: Application Programmer's Interface

Describes the Application Programmer's Interface (API) to the GFR module.

8

Application Programmer's Interface

Describes the Application Programmer's Interface (API) to the GFR module.

Introduction

This chapter presents all the procedures and functions in GFR's API in the following functional categories:

- Module management
- Communications and error handling
- Localization
- Palette management
- Unique ID facility
- File parsing and other utilities

Specifying the Client Object Argument

Most API procedures require a client object as their last argument. In general, this is the g2-window where the call originated, but the client can be any object that represents the source of the call, such as another G2, an external program, etc. When communicating with the user, GFR uses the client object to determine the user mode and language, when needed.

When you begin a thread of processing from an action button or user menu choice, the client should be the window associated with the button or menu choice and it should be accessed with the this window syntax. When you start a procedure from an external client, you should create your own client object representing the external client. If the thread of processing is started by a rule or other scheduled activity in G2 not associated with a specific window, you may use the permanent g2-window object named gfr-default-window as the client argument.

Module Management Utilities

GFR provides the following procedures and functions for managing modules:

gfr-deposit-item-in-public-bin

gfr-disable-error-handling

gfr-disable-version-checking

gfr-enable-error-handling

gfr-enable-version-checking

gfr-get-active-setting

gfr-get-directly-required-modules

gfr-get-directly-requiring-modules

gfr-get-g2-version

gfr-get-handler-hierarchy

gfr-get-linearized-module-hierarchy

gfr-get-module-of-item

gfr-get-public-bin-for-module

gfr-get-required-modules

gfr-get-requiring-modules

gfr-get-supporting-version-information

gfr-get-top-level-module

gfr-get-version

gfr-install-module-settings

gfr-invalidate-module-information

gfr-startup-module

gfr-startup-modules

gfr-deposit-item-in-public-bin

Places an item in a module's public bin.

Synopsis

gfr-deposit-item-in-public-bin
(*item*: class item, *module*: symbol)

Argument	Description
item	The item to be deposited.
module	The module where the item is to be stored.

Description

You use this procedure when your module wants to store an item in another module. This procedure first finds the target module's public bin or creates one if the bin does not exist. Then, the item is transferred to the subworkspace of the bin at a random location.

The item passed to this procedure must be transient, otherwise an error is signalled. The item deposited in the bin is not made permanent by this procedure.

Example

The following call deposits an instance of a gfr-module-setting in the module module-1 and makes it permanent:

```
create a gfr-module-setting S;
call gfr-deposit-item-in-public-bin(S, the symbol module-1);
make S permanent;
```

gfr-disable-error-handling

Disables GFR's error handling facility until the next G2 start or until GFR's error handling facility is enabled by gfr-enable-error-handling.

Synopsis

gfr-disable-error-handling ()

Description

You use this procedure if you want to disable GFR's error handling facility. The effect of this procedure lasts only through the next G2 start. To control whether GFR initially enables error handling, use a gfr-settings-object.

Example

The following call turns off error handling:

```
call gfr-disable-error-handling();
```

gfr-disable-version-checking

Disables version checking and module upgrades until the next G2 start or until gfr-enable-version-checking is called.

Synopsis

gfr-disable-version-checking ()

Description

You use this procedure if you want to prevent GFR from checking version consistency when modules are merged into a running G2. This procedure should only be used in special situations, because it may result in an inconsistent set of modules being loaded.

The effect of this procedure lasts only through the next G2 start. To control whether GFR initially checks versions, use a gfr-startup-settings object.

Example

The following call turns off version checking:

```
call gfr-disable-version-checking();
```

gfr-enable-error-handling

Enables GFR's error handling facility by replacing the current default error handler with GFR's error handling dispatcher.

Synopsis

gfr-enable-error-handling ()

Description

You use this procedure if you want to enable GFR's error handling facility.

Example

The following call turns on error handling:

```
call gfr-enable-error-handling();
```

gfr-enable-version-checking

Enables version checking and module upgrades until the next G2 start.

Synopsis

gfr-enable-version-checking ()

Description

You use this procedure if you want GFR to check version consistency when modules are merged into a running G2.

The effect of this procedure lasts only through the next G2 start. To control whether GFR initially checks version, use a gfr-startup-settings object.

Example

The following call turns on version checking:

```
call gfr-enable-version-checking();
```

gfr-get-active-setting

Finds the active module setting of a given class.

Synopsis

gfr-get-active-setting
(*classname*: symbol, *client*: class object)
-> *setting*: class gfr-module-setting

Argument	Description
classname	The class of the module setting to be retrieved.
client	The client originating this call.

Return Value	Description
setting	The active module setting matching the *classname*.

Description

This procedure returns the active module setting object of a given class. The active module setting is the instance of the given class that is highest in the module hierarchy. If there is more than one instance of the class in the highest module, one instance is chosen arbitrarily. The class name of the returned item matches the *classname* argument exactly (i.e., the returned object is not a subclass of the target class).

If there is no instance of the given *classname*, an error is signalled.

Example

The following call returns the active module-x-color-setting, which is a subclass of gfr-module-setting:

```
Setting = call gfr-get-active-setting(the symbol module-x-color-setting, Win);
```

gfr-get-directly-required-modules

Returns the modules directly required by another module.

Synopsis

gfr-get-directly-required-modules
(*module*: symbol, *module-list*: class symbol-list, *client*: class object)

Argument	Description
module	The name of the module that is the subject of this call.
module-list	The required modules appended to this list.
client	The client originating this call.

Description

This procedure returns the names of the modules directly required by a given module. This list of module names is simply the contents of the attribute directly-required-modules of the module-information object for the given module.

This procedure does not clear the *module-list* before appending the required modules.

Example

The following call gets the directly required modules of module GFR:

```
call gfr-get-directly-required-modules(the symbol GFR, module-list, Win);
```

The symbol sys-mod is appended to module-list as a result of this call.

gfr-get-directly-requiring-modules

Returns the names of the modules directly requiring a given module.

Synopsis

gfr-get-directly-requiring-modules
(*module*: symbol, *module-list*: class symbol-list, *client*: class object)

Argument	Description
module	The name of the module that is the subject of this call.
module-list	The requiring modules appended to this list.
client	The client originating this call.

Description

This procedure returns the names of the modules directly requiring a given module, that is, the name of each module that includes the target module as one of its directly-required-modules, as given by its module-information object.

This procedure does not clear the *module-list* before appending the requiring modules.

Example

The following call gets a list of modules directly requiring the module sys-mod:

```
call gfr-get-directly-requiring-modules(the symbol sys-mod, module-list, Win);
```

Assuming GFR is loaded and is the only module requiring sys-mod, the symbol GFR is appended to module-list as a result of this call.

gfr-get-g2-version

Returns the current G2 version in quantitative form.

Synopsis

gfr-get-g2-version ()
-> *release*: float, *revision*: integer, *type*: symbol

Value	Description
release	The major/minor release number of G2.
revision	The revision number of G2.
type	The release type of G2, which is one of ALPHA, BETA or RELEASE.

Description

G2 versions are expressed as a major release, a minor release, and a revision number. This procedure returns the major-minor release number as in dd.d format, and the revision number as an integer.

The type is a symbol representing the type of G2 release (ALPHA, BETA or RELEASE).

Example

If G2 Version 2015 Rev. 0 is being used, then the following procedure returns 7.0 and 0 as the values of release, revision, and type, respectively:

```
example-proc (Win: class g2-window)
release: float;
revision: integer;
type: symbol;
begin
    release, revision, type = call gfr-get-g2-version();
    inform the operator that "G2 version info is [release] Rev. [revision] {type]";
end
```

The version information appears on the Message Board, as shown in the following figure:


MESSAGE-BOARD
#16 1:23:00 p.m. G2 version info is 5.0 Rev. 1
RELEASE

gfr-get-handler-hierarchy

Determines the hierarchy of handlers for a specified item.

Synopsis

gfr-get-handler-hierarchy
(*item*: class item, *handlers*: class item-list, *client*: class object)

Arguments	Description
item	The item whose handlers you want to determine.
handlers	A list of handlers for this item.
client	The client originating this call.

Description

This procedure enables you to determine the hierarchy of handlers for a given item. The item must be either an instance of class error or a gfr-communication. The *handlers* argument is usually an empty list on input. On output, it contains the handlers for the given item in order from highest to lowest precedence.

gfr-get-linearized-module-hierarchy

Returns a sorted list of the modules in a KB.

Synopsis

gfr-get-linearized-module-hierarchy
(*module-list*: class symbol-list, *client*: class object)

Argument	Description
module-list	The names of modules present in the KB are appended in sorted order to this list.
client	The client originating this call.

Description

This procedure returns the module hierarchy flattened into a list. The list contains all modules sorted so that if a module X is required by another module Y, module X always appears after module Y in the list. The linearization follows the same rules as the linearized class inheritance path for object definitions with multiple inheritance (see the *G2 Reference Manual* for details).

This procedure does not clear the *module-list* before appending to it.

Example

Consider the following module hierarchy involving modules U, V, W, X, Y and Z:

- U requires V and W.
- V requires X and Y.
- Y requires Z.
- W requires Z.

The following call gets the linearization of this module hierarchy:

```
call gfr-get-linearized-module-hierarchy(module-list, Win);
```

On return, the module-list contains the modules in this order: U, V, X, Y, W, Z.

gfr-get-module-of-item

A function that returns the current module assignment of the item.

Synopsis

gfr-get-module-of-item
    (*item*: class item)
    -> *module*: symbol

Argument	Description
item	The item whose module assignment is to be determined.

Return Value	Description
module	The module of the item or the symbol unspecified.

Description

The gfr-get-module-of-item function call returns the name of the module to which an item is assigned. If the item is not assigned to any module, gfr-get-module-of-item returns the symbol unspecified.

Example

The following example determines the module assignment of an object Obj and its definition Def, and determines if proper modularity is satisfied:

```
InstanceModule = gfr-get-module-of-item(Obj);
DefinitionModule = gfr-get-module-of-item(Def);
if InstanceModule /= DefinitionModule then begin
    create a symbol-list module-list;
    call gfr-get-requiring-modules(DefinitionModule, module-list, Win);
    if InstanceModule is a member of module-list then inform the operator that
        “modularity is OK” else inform the operator that “KB is not correctly
        modularized”;
    delete module-list;
end else inform the operator that “modularity is OK”
```

gfr-get-public-bin-for-module

Returns the public bin for a module.

Synopsis

gfr-get-public-bin-for-module
(*module*: symbol, *client*: class object
-> *module-bin*: class gfr-public-bin)

Arguments	Description
module	The module that owns the public bin you want to use
client	The client originating this call.

Return Value	Description
module-bin	The public bin of the specified module.

Description

This procedure returns the public bin for the specified module. If a bin does not exist when this call is made, this procedure creates one. The objects deposited in the bin are stored on the subworkspace of the bin.

gfr-get-required-modules

Returns the modules required by another module, including indirectly required modules.

Synopsis

gfr-get-required-modules
(*module*: symbol, *module-list*: class symbol-list, *client*: class object)

Argument	Description
module	The name of the module that is the subject of this call.
module-list	The required modules are appended to this list.
client	The client originating this call.

Description

This procedure returns the names of the modules required by a given module. These are all the modules that are below the given module in the module hierarchy.

This procedure does not clear the *module-list* before appending the required modules.

Example

The following call returns the required modules of module GFR:

```
call gfr-get-directly-required-modules(the symbol GFR, module-list, Win);
```

The symbols sys-mod and uilroot are appended to module-list as a result of this call.

gfr-get-requiring-modules

Returns all modules requiring a given module, directly or indirectly.

Synopsis

gfr-get-requiring-modules
(*module*: symbol, *module-list*: class symbol-list, *client*: class object)

Argument	Description
module	The name of the module that is the subject of this call.
module-list	The requiring modules are appended to this list.
client	The client originating this call.

Description

This procedure returns the names of the modules that require the given module. These are all the modules that require, directly or indirectly, the given module.

This procedure does not clear the *module-list* before appending the requiring modules.

Example

The following call gets a list of modules directly requiring the module uilroot:

```
call gfr-get-requiring-modules(the symbol uilroot, module-list, Win);
```

Assuming GFR is the top level module, the symbols GFR and sys-mod are appended to module-list as a result of this call.

gfr-get-supporting-version-information

Returns information about the modules supporting another module from that module's information object.

Synopsis

gfr-get-supporting-version-information
(*module*: symbol, *client*: class object)
-> *version-information*: sequence

Argument	Description
module	The name of the module for which supporting module version information is sought
client	The client originating this call.

Return Value	Description
version-information	A sequence of structures describing the supporting modules, where each structure has the form: (module-name: symbol, version-string: text, version-number: quantity, compatible-version-number: quantity)

Description

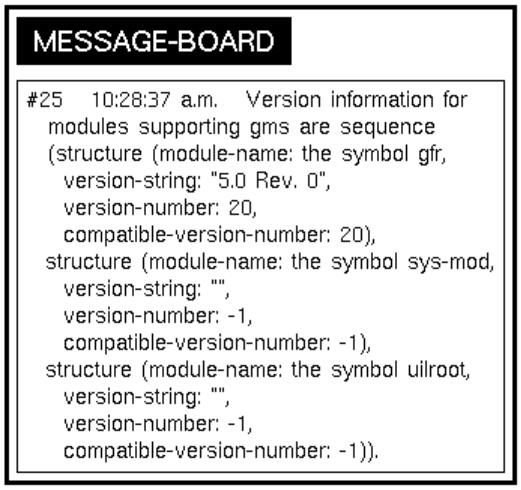
This procedure provides version information about all the modules that support the given module. The version information is returned in the form of a sequence, with one element for each supporting module. Each structure in the sequence includes the supporting module name, the current version as a text, the current version number as an quantity, and the oldest compatible version number as a quantity.

Example

The following example procedure determines the supporting module version information for the GMS module, where Version Info is a sequence:

```
example-proc (Win: class g2-window)
VersionInfo: sequence;
begin
    VersionInfo = call gfr-get-supporting-version-information (the symbol
      gms,  Win);
    inform the operator that "Version information for modules supporting gms are [VersionInfo].";
end
```

The following message is printed to the Message Board:



gfr-get-top-level-module

A function that returns the name of the top-level module.

Synopsis

gfr-get-top-level-module ()
-> *name*: symbol

Return Value	Description
name	The name of the top-level module.

Description

The gfr-get-top-level-module function call returns the name of the top-level module.

Example

Assuming GFR is the top-level module, the following function call returns the symbol GFR:

```
TopModule = gfr-get-top-level-module();
```

gfr-get-version

Returns the version of the gfr module.

Synopsis

gfr-get-version ()
-> *version*: text, *revision*: quantity

Return Value	Description
version	A text describing the version number of the module. The version number is found in the gfr-version-description attribute in the gfr-version-information-object.
revision	An integer which increments on every revision of the module. The revision is the gfr-version-number.

Description

This procedure allows you to find out the version of the gfr module that is currently loaded.

The version information is found in the gfr-version-description attribute in the gfr-version-information-object. It is returned as a text in the format *major.minor type revision*, where:

major: An integer representing the major release number.

minor: An integer representing the minor release number.

type: A word describing the type of release, such as "Rev.," "Alpha," or "Beta."

revision: An integer representing the revision number.

For example, the current release of this software is "7.0 Rev. 0". The format of this string is subject to change.

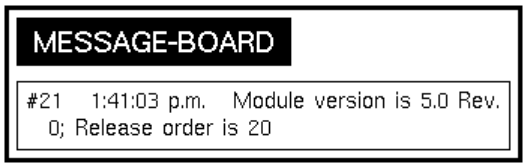
The second return argument can be used to establish the release order of different versions of the module. This number increases with each revision of the software.

Example

The following procedure returns version information about the gfr module:

```
example-proc2 (Win: class g2-window)
versino: text;
revision: quantity;
begin
     version, revision = call gfr-get-version();
     inform the operator that "Module version is [version];
          Release order is [revision]";
end
```

The version information for the module appears in the Message Board:



gfr-install-module-settings

Installs the module settings for a given module.

Synopsis

gfr-install-module-settings
(*module*: symbol, *client*: class object)

Argument	Description
module	The name of the module that is the subject of this call.
client	The client originating this call.

Description

This procedure installs the module settings for a given module. GFR determines which module settings to install in a given module by looking at the subclasses of gfr-module-setting defined in that module. For each setting class defined in the target module, GFR determines which instance of that class should be active, using the module hierarchy precedence described in the Managing User-Settable Parameters for Modules.

Caution You should always provide a default instance of every class of module setting you define, so that it is guaranteed that GFR will find a suitable module setting to install.

When GFR locates the instance, it calls the method gfr-propagate-module-setting-information. This method takes the actions necessary to implement the settings contained in the active setting object. You may, for example, copy information in the setting object into private data structures within the target module.

This procedure is called automatically by GFR before it starts up a module.

Example

See Managing User-Settable Parameters for Modules.

gfr-invalidate-module-information

Forces GFR to read the module hierarchy and update its cached module information.

Synopsis

gfr-invalidate-module-information
(*client*: class object)

Argument	Description
client	The client originating this call.

Description

GFR internally caches the module hierarchy in a form that allows rapid access to this information. G2 sets up this cache when it starts and revises it when the module hierarchy is changed. When a call is made to one of the GFR procedures that provides module hierarchy information, GFR uses the cached information, not the actual module hierarchy.

Revising the module cache depends on firing event-detector rules. Therefore, if you make a programmatic change to the module hierarchy and *immediately* want module hierarchy information, you must call this procedure to force GFR to cache the current module structure.

Example

The following call invalidates GFR's cached module information:

```
call gfr-invalidate-module-information(Win);
```

gfr-startup-module

Performs initialization activities for a module.

Synopsis

gfr-startup-module
(*module*: symbol, *client*: class object)

Argument	Description
module	The module to start.
client	The client originating this call.

Description

This procedure performs the startup activities for a module. These activities include, in the following order:

- For each module required by this module, validate the version of this module and, if necessary, perform upgrade activities on this module.
- Validate the G2 version required by this module.
- Validate the versions of and, if necessary, upgrade all modules that require this module.
- Install the module settings by calling gfr-install-module-settings.
- Load text resources, if the gfr-preload-resource attribute of the text resource is true.
- If an instance of a gfr-startup-object exists in the module, call the procedure named by the startup-procedure of the object.

Example

The following call starts up the module module-1:

```
call gfr-startup-module(the symbol module-1, Win);
```

gfr-startup-modules

Performs initialization activities for a list of modules.

Synopsis

gfr-startup-modules
(*modules*: class symbol-list, *client*: class object)

Argument	Description
modules	The list of modules to perform startup activities on.
client	The client originating this call.

Description

This procedure performs the startup activities for a list of modules. The startup activities for the modules in the list are performed in bottom-up order, with the modules lower in the module hierarchy starting before those higher in the hierarchy. The startup activities are the same as described in Using Module Startup Objects.

Example

The following call starts up the modules module-1 and module-2:

```
create a symbol-list Modules;
insert the symbol module-1 at the end of Modules;
insert the symbol module-2 at the end of Modules;
call gfr-startup-modules(Modules, Win);
delete Modules;
```

Communications Operations

GFR provides the following procedures for structured communications in a KB:

gfr-call-next-communication-handler

gfr-call-next-error-handler

gfr-dispatch-communication

gfr-call-next-communication-handler

Dispatches a call to the next communication handler appropriate to the given communication.

Synopsis

gfr-call-next-communication-handler
(*communcation*: class gfr-communcation, *initiatingitem*: class item,
client: class object)
-> *return*: structure

Argument	Description
communication	The gfr-communication that is currently being handled
initiatingitem	The initiating item associated with the communication, passed to the handler
client	The client originating this call.

Return Value	Description
return	Any attributes returned from the handler.

Description

This procedure is used to pass control to the next communication handler capable of handling a specific type of communication. The precedence order for communication handlers is described in Communication and Error Handlers. You use this procedure in a manner similar to G2's "call next method" statement.

This procedure can only be called from a communication handler, otherwise an error is signalled.

The returned structure contains any attributes returned from the handler, if any.

gfr-call-next-error-handler

Dispatches a call to the next error handler appropriate to the given error.

Synopsis

gfr-call-next-error-handler
(*error*: class error)

Argument	Description
error	The error that is being processed.

Description

This procedure is used to pass control to the next error handler capable of handling a specific type of error. The precedence order for error handlers is described in Communication and Error Handlers. You use this procedure in a manner similar to G2's "call next method" statement.

This procedure can only be called from an error handler; otherwise an error is signalled.

If an appropriate next error handler is not found, calling this procedure has no effect.

gfr-dispatch-communication

Dispatches a gfr-communication to GFR's communication handling system.

Synopsis

gfr-dispatch-communication
(*communcation*: class gfr-communication, *initiating-item*: class item,
client: class object)
-> *return*: structure

Argument	Description
communication	The communication to be handled.
initiating-item	Any item associated with the communication.
Client	The client originating this call.

Return Value	Description
return	Any attributes returned from the handler.

Description

This procedure is called to dispatch a *Communication* to GFR's communication handling facility. Typically, you create and configure a communication object, then call this procedure to begin the processing of the *Communication*. GFR will find the handler with the highest precedence for that type of communication and call it.

If the *Communication* defines one or more return arguments, they will be returned from gfr-dispatch-communication in the returned structure.

Localization Operations

GFR provides the following procedures and functions for localizing text in a KB:

gfr-add-to-local-text-resource

gfr-clear-local-text-resource

gfr-configure-text-proxy

gfr-do-single-text-substitution

gfr-evaluate-text-proxy

gfr-get-all-unsubstituted-messages

gfr-get-local-text-resource

gfr-get-unsubstituted-message

gfr-language

gfr-load-local-text-resource-from-file

gfr-localize-message

gfr-localize-messages-on-workspace

gfr-make-local-text-resource-permanent

gfr-modify-message-in-local-text-resource

gfr-remove-from-local-text-resource

gfr-write-local-text-resource-to-file

gfr-add-to-local-text-resource

Adds one key-text pair to a local text resource.

Synopsis

gfr-add-to-local-text-resource
(*resource*: class gfr-local-text-resource, *key*: symbol, *msg*: text,
client: class object)

Argument	Description
resource	The local text resource that is the target of this operation.
key	The symbolic key for the message to be added to the resource.
msg	The text of the message to be added to the resource.
client	The client originating this call.

Description

This procedure adds a single key-text pair to a given local text resource. You provide the symbolic key and text of the message you want to add. This procedure does not affect the permanent values of the local text resource. If you want your changes to be permanent, you must call gfr-make-local-text-resource-permanent.

Example

The following call adds the message "Help" under the key gfr-help to the local text resource named text-resource-1:

```
call gfr-add-to-local-text-resource(text-resource-1, the symbol gfr-help,
    "Help", win);
```

gfr-clear-local-text-resource

A function that clears the key-text pairs stored in a local text resource.

Synopsis

gfr-clear-local-text-resource
(*resource*: class gfr-local-text-resource, *client*: class object)

Argument	Description
resource	The local text resource that is the target of this operation.
client	The client originating this call.

Description

This function clears the localized texts and the symbolic keys that are the contents of a local text resource. Other attributes of the local text resource are not affected by this call. This call does not affect permanent values of the resources, if any.

Example

The following function calls the local text resource named text-resource-1:

```
call gfr-clear-local-text-resource(text-resource-1, win);
```

gfr-configure-text-proxy

Sets the symbol key, resource group, and substitution arguments of a text proxy.

Synopsis

gfr-configure-text-proxy
(*proxy*: class gfr-text-proxy, *resource-group*: symbol, *key*: symbol, *optional-arguments*)

Argument	Description
proxy	The text proxy to be configured.
resource-group	The name of the resource group containing the *key*.
key	A symbol that is the key for retrieving the localized text.
optional-arguments	Up to 10 values that will be the substitution values in the localized text when the proxy is evaluated.

Description

This procedure changes the attributes of gfr-text-proxy or gfr-simple-text-proxy objects. Calling this procedure is equivalent to:

- Concluding that the gfr-text-resource-group attribute of *proxy* is *resource-group*.
- Concluding that the gfr-message-name attribute of *proxy* is *key*.
- Inserting the values in optional arguments at the end of the gfr-substitutions list of the proxy.

It is up to the caller to clear the substitutions list of the text proxy, if desired, before making this call.

Example

The following call configures the text proxy proxy-1 to refer to the resource group named my-error-messages and the localized text keyed by the symbol error-message-1, with the substitutions 123 and the symbol foo:

```
call gfr-configure-text-proxy(proxy-1, the symbol my-error-messages,
    the symbol error-message-1, 123, the symbol foo);
```

gfr-do-single-text-substitution

Substitutes a given value for a substitution marker.

Synopsis

gfr-do-single-text-substitution
(*input-text*: text, *substitution-number*: integer, *substitution*: value)
-> *text*: text

Argument	Description
input-text	The text into which substitutions are to be made.
substitution-number	The integer marker that is to be substituted for.
substitution	The value to be converted into text and substituted into the *input-text*.

Return Value	Description
text	The text resulting from the substitutions.

Description

This procedure substitutes the substitution value at every position identified by the bracketed integer [*substitution-number*]. If there is no match, the returned text is the same as *input-text*.

Example

The following call substitutes the symbol foo wherever [2] occurs in the input text:

```
output-text = call gfr-do-single-text-substitution("[2] is a meaningless
    [1] symbol and so is [2]bar", 2, the symbol foo);
```

The result is "foo is a meaningless [1] symbol and so is foobar".

gfr-evaluate-text-proxy

Returns the text represented by a text proxy localized to a given language.

Synopsis

gfr-evaluate-text-proxy
(*proxy*: class gfr-text-proxy, *language:* symbol, *client*: class object)
-> *text*: text

Argument	Description
proxy	A text proxy whose gfr-message-name, gfr-text-resource-group, and gfr-substitutions have been configured.
language	The target language.
client	The client originating this call.

Return Value	Description
text	The text resulting from evaluating the proxy.

Description

This procedure takes a configured text proxy object and returns the text represented by the text proxy in the given language.

Example

Suppose the symbol error-message-1 in the resource group named my-error-messages is keyed to the English text "Error [1]: Illegal object type [2]". The following code fragment configures the text proxy proxy-1 and then evaluates it:

```
call gfr-configure-text-proxy(proxy-1, my-error-messages,
    the symbol error-message-1, 123, the symbol foo);
ErrorMessage = call gfr-evaluate-text-proxy(proxy-1, the symbol English,
    Win);
```

As a result of these calls, ErrorMessage will be "Error 123: Illegal object type foo".

gfr-get-all-unsubstituted-messages

Returns the full contents of a local text resource.

Synopsis

gfr-get-all-unsubstituted-messages
(*resource*: class gfr-local-text-resource, *keys*: class symbol-list,
msgs: class text-list, *client*: class object)

Argument	Description
resource	The local text resource that is the target of this operation.
keys	A symbol list that will receive the keys stored in the resource.
msgs	A text list that will receive the message texts stored in the resource.
client	The client originating this call.

Description

This procedure returns the full contents of a local text resource in two lists you provide. This procedure does not clear the provided lists before appending the keys and texts in the local text resource.

Example

The following call retrieves the contents of text-resource-1 into symbol-list-1 and text-list-1:

```
call gfr-get-all-unsubstituted-messages(text-resource-1, symbol-list-1,
    text-list-1, win);
```

gfr-get-local-text-resource

Returns the local text resource object belonging to a resource group, given a language.

Synopsis

gfr-get-local-text-resource
(*resource-group*: class gfr-text-resource-group, *language:* symbol,
client: class object)
-> *resource*: class gfr-local-text-resource

Argument	Description
resource-group	The resource group that is the target of this call.
language	The target language.
client	The client originating this call.

Return Value	Description
resource	The local text resource located by this call.

Description

This procedure returns a local text resource associated with a given resource group in the specified language. All local text resource objects whose gfr-resource-group attribute names the given resource group are considered potential matches.

If the resource group is configured to allow the use of a default language and the local text resource for the specified language is not found, the local text resource corresponding to the default language is returned.

As a side effect of calling this procedure, if the local text resource is empty, a call is made to gfr-load-local-text-resource-from-file, so the local text resource contains its proper key-value pairs on return.

If a suitable local text resource cannot be found, the error gfr-language-not-found is signalled.

Example

The following code fragment gets the English-language local text resource from the resource group named my-error-messages:

```
LocalResource = call gfr-get-local-text-resource(my-error-messages,
    the symbol English, Win);
```

gfr-get-unsubstituted-message

Returns a text from a local text resource based on a key, without substitutions.

Synopsis

gfr-get-unsubstituted-message
(*local-resource*: class gfr-local-text-resource, *key:* symbol, *client*: class object)
-> *text*: text

Argument	Description
local-resource	The resource group that is the target of this call.
key	The symbolic key to the text being retrieved.
client	The client originating this call.

Return Value	Description
text	The text retrieved by this call.

Description

This is a low-level procedure for returning a text from a local resource, given a key. When you use this procedure, substitutions are not made in the text. An empty string is returned if the key is not matched.

To retrieve a local text resource, use gfr-get-local-text-resource.

Example

The following code fragment gets a text keyed by the symbol error-message-1 in English from a local resource associated with the resource group named my-error-messages:

```
LocalResource = call gfr-get-local-text-resource(my-error-messages,
    the symbol English, Win);
ErrorMessage = call gfr-get-unsubstituted-message(LocalResource,
    the symbol error-message-1, Win);
```

gfr-language

A function that returns the language of a window or the default language.

Synopsis

gfr-language
(*client*: class object)
-> *language*: symbol

Argument	Description
client	The client object whose language is to be determined.

Return Value	Description
language	The language of the window or the global default.

Description

Use the gfr-language function call when you are determining the language of a g2-window or other client object.

- If the client is a window and the window-specific language of the window exists, gfr-language returns that language.
- If not, this function returns the value of the current-language attribute of the Language Parameters system table.

Example

The following function call gets the language of the window Win:

```
Language = gfr-language(Win);
```

gfr-load-local-text-resource-from-file

Loads a local text resource with the key-text value pairs found in a file.

Synopsis

gfr-load-local-text-resource-from-file
(*local-resource*: class gfr-local-text-resource, *client*: class object)

Argument	Description
local-resource	The resource group that is the target of this call.
client	The client originating this call.

Description

This procedure loads a local text resource, using the contents of the file identified by the gfr-file-location attribute of *local-resource*. Only the key-text value pairs are updated when the file is loaded; the remaining attributes of the local resource are unaffected. The resource is cleared before loading the data contained in the file. Calling this procedure does not affect the permanent values of the text resource.

For a description of the required file format, see Localizing KBs.

- If the file format is incorrect, the error gfr-invalid-file-format is signalled.
- If the file names a resource group other than that named by the gfr-resource-group attribute of *local-resource*, the error gfr-wrong-resource-group is signalled.
- If the file cannot be opened for read, the error gfr-open-file-failure is signalled.

Example

The following call loads the local text resource named local-resource-1, using the file path specified in its gfr-file-location attribute:

```
call gfr-load-local-text-resource-from-file(local-resource-1, Win);
```

gfr-localize-message

Gets the localized version of a text based on a resource group, key, and language, and performs substitutions, if any.

Synopsis

gfr-localize-message
(*resource-group*: class gfr-text-resource-group, *key*: symbol, *language*: symbol, *optional-arguments*, *client*: class object)
-> *text*: text

Argument	Description
resource-group	The resource group containing the *key*.
key	A symbol that is the key for retrieving the localized text.
language	The language for the returned text.
optional-arguments	Up to 10 values to be substituted into the returned text.
client	The client originating this call.

Return Value	Description
text	The localized text produced by this call.

Description

Calling this procedure is the standard way to obtain localized texts. Given a resource group, key, and language, this procedure finds the local text resource, retrieves the text corresponding to the key, and then performs substitutions on the text for each bracketed integer.

This procedure may return the text in the default language of the resource group, if the gfr-use-default-language of the resource group is true. If the key cannot be found in either the specified language or the default language (if used), the empty string (" ") is returned.

Example

The following call retrieves an English-language text from the resource group named my-error-messages. keyed by the symbol error-message-1, with the substitutions 123 and the symbol foo:

```
ErrorMessage = call gfr-localize-message(my-error-messages,
    the symbol error-message-1, the symbol English, 123, the symbol foo,
    Win);
```

If the unsubstituted text is "Error [1]: Illegal object type [2]", then ErrorMessage will have the value "Error 123: Illegal object type foo" as a result of this call.

gfr-localize-messages-on-workspace

Localizes into a given language all items of the class gfr-localizable-message on a given workspace.

Synopsis

gfr-localize-messages-on-workspace
(*workspace*: class kb-workspace, *language*: symbol, *client*: class object)

Argument	Description
workspace	The workspace that contains messages to be localized.
language	The language for the localized messages.
client	The client originating this call.

Description

This procedure localizes all localizable messages on a workspace. Each instance of a gfr-localizable-message upon the workspace receives the following treatment:

- The text of the message is changed to the result obtained by evaluating the text proxy of the message.
- The message is repositioned upon the workspace at the position indicated by the position invariants of the message.

You must make all gfr-localizable-messages on the target workspace transient before calling this procedure. You may make them permanent after the completion of this procedure.

Example

Please refer to the example described in Using Localizable Message Classes.

gfr-make-local-text-resource-permanent

Makes the current key-text pairs in a local text resource a permanent part of the KB.

Synopsis

gfr-make-local-text-resource-permanent
(*local-resource*: class gfr-local-text-resource, *client*: class object)

Argument	Description
local-resource	The local resource that will be made permanent by this call.
client	The client originating this call.

Description

This procedure makes the transient key-text value pairs stored in a local text resource the initial values of the local resource. The initial values are the values that will be stored when the KB is saved in a file and the values that the local resource will have when G2 is initially started. See Storing Local Text Resources for a full discussion of the alternative methods of persistent storage of the information in a local text resource.

Example

The following call sets the initial values of the local text resource named local-resource-1:

```
call gfr-make-local-text-resource-permanent(local-resource-1, Win);
```

gfr-modify-message-in-local-text-resource

Changes the text of one key-text pair in a local text resource.

Synopsis

gfr-modify-message-in-local-text-resource
(*resource*: class gfr-local-text-resource, *key*: symbol, *msg*: text,
client: class object)

Argument	Description
resource	The local text resource that is the target of this operation.
key	The symbolic key for the message to be modified in the resource.
msg	The text that will replace the existing text of the message referenced by *key*.
client	The client originating this call.

Description

This procedure changes the text associated with an existing key in a local text resource and does not affect the permanent values of the local text resource.

- If you want your changes to be permanent, you must call gfr-make-local-text-resource-permanent.
- To add a new key-text pair, see gfr-add-to-local-text-resource.
- If *key* does not exist, this procedure signals an error.

Example

The following call changes the text associated with the key gfr-sample to "New sample text" in the local text resource named text-resource-1:

```
call gfr-modify-message-in-local-text-resource(text-resource-1,
    the symbol gfr-sample, "New sample text", win);
```

gfr-remove-from-local-text-resource

Removes one key-text pair from a local text resource.

Synopsis

gfr-remove-from-local-text-resource
(*resource*: class gfr-local-text-resource, *key*: symbol, *client*: class object)

Argument	Description
resource	The local text resource that is the target of this operation.
key	The symbolic key for the message to be removed from the resource.
client	The client originating this call.

Description

This procedure removes a key-text pair from a given local text resource based on its key. This procedure does not affect the permanent values of the local text resource.

- If you want your changes to be permanent, you must call gfr-make-local-text-resource-permanent.
- If *key* does not exist, this procedure signals an error.

Example

The following call removes the key-text pair associated with the key gfr-sample in the local text resource named text-resource-1:

```
call gfr-remove-from-local-text-resource(text-resource-1,
    the symbol gfr-sample, win);
```

gfr-write-local-text-resource-to-file

Writes the current contents of a local text resource to a file.

Synopsis

gfr-write-local-text-resource-to-file
(*local-resource*: class gfr-local-text-resource, *client*: class object)

Argument	Description
local-resource	The resource group that is the target of this call.
client	The client originating this call.

Description

This procedure writes the name of the resource group, version information, language, and the current key-text value pairs stored in a local text resource to the file identified by the gfr-file-location attribute of the local resource. For a description of the resulting file format, see Localizing KBs. If the file cannot be opened for read, the error gfr-open-file-failure is signalled.

Example

The following call writes the information in the local text resource local-resource-1 to a file:

```
call gfr-write-local-text-resource-to-file(local-resource-1, Win);
```

Procedures Dealing with Palette Management

GFR provides the following procedures and functions for managing palettes:

gfr-add-palette-behavior-to-item

gfr-create-instance-using-palette-method

gfr-item-is-palette-object

gfr-remove-palette-behavior-from-item

gfr-show-bubble-help

gfr-add-palette-behavior-to-item

Adds a palette window to the item, giving it palette behavior.

Synopsis

gfr-add-palette-behavior-to-item
(*item*: class item, *client*: class object)
-> *palette-window*: class gfr-palette-window

Argument	Description
item	The item that is to be converted into a palette item.
client	The client originating this call.

Return Value	Description
palette-window	The new palette window created by this procedure.

Description

This procedure adds a palette window to the given item, giving it the palette behavior defined in Managing Palettes.

This procedure can refer to inactive items.

Example

The following call gives item-1 palette behavior:

```
call gfr-add-palette-behavior-to-item(item-1, win);
```

gfr-create-instance-using-palette-method

Programmatically creates an instance, using the same method used in creating from a palette.

Synopsis

gfr-create-instance-using-palette-method
(*class*: symbol, *client*: class object)
-> *instance*: class item

Argument	Description
class	The class of the item that is to be created.
client	The client originating this call.

Return Value	Description
instance	The new instance created by this procedure.

Description

This procedure creates an item, using the palette creation method. The item returned from this procedure is the same as it would be if you obtained it by manual cloning from a palette. Since palette items can have a subworkspace, non-default size or other non-default attributes depending on the palette configuration and the nature of the gfr-create-instance-from-palette-item method, the item returned from this procedure might be different from an item of the same class created using the native G2 create action.

Example

The following call creates an instance of the class foobar:

```
Bar = call gfr-create-instance-using-palette-method(the symbol foobar, win);
```

gfr-item-is-palette-object

A function that determines if an item has palette behavior.

Synopsis

gfr-item-is-palette-object
(*item*: class item)
-> *outcome*: truth-value

Argument	Description
item	The item in question.

Return Value	Description
outcome	A truth-value indicating if the item has palette behavior.

Description

This function is used to determine if an item has palette behavior. True is returned if the item has a palette window or is a subobject of a palette item and therefore has palette behavior; otherwise false is returned.

Example

The following function call determines if item-1 has palette behavior:

```
IsPaletteItem = gfr-item-is-palette-object(item-1);
```

gfr-remove-palette-behavior-from-item

Removes the palette window from an item, removing its palette behavior.

Synopsis

gfr-remove-palette-behavior-from-item
(*item*: class item, *client*: class object)

Argument	Description
item	The item whose palette behavior is to be removed.
client	The client originating this call.

Description

This procedure removes the palette window from the given item, if it contains one. Removing palette behavior is defined in Managing Palettes.

Item can be an inactive item.

Example

The following call removes palette behavior from item-1:

```
call gfr-remove-palette-behavior-from-item(item-1, win);
```

gfr-show-bubble-help

Pops up a workspace in the vicinity of the mouse.

Synopsis

gfr-show-bubble-help
(*mouse-ws*: class kb-workspace, *bubble-ws*: class kb-workspace,
mouse-x: integer, *mouse-y*: integer, *win*: class g2-window)

Argument	Description
mouse-ws	The workspace that the mouse is currently tracking over.
bubble-ws	The workspace that is to be shown.
mouse-x	The current mouse x-position upon *mouse-ws, as a workspace coordinate.*
mouse-y	The current mouse y-position upon *mouse-ws*, as a workspace coordinate.
win	The window where the bubble workspace is to be shown.

Description

This procedure shows a workspace (called the bubble workspace) in the vicinity of the mouse. If possible, the entire bubble workspace is shown on the screen, without positioning the bubble workspace directly under the mouse.

You must provide the current mouse coordinates to this procedure, which are usually provided from a mouse tracking procedure.

The magnification of the bubble workspace is automatically set to the same magnification as the workspace that the mouse is currently tracking over.

Procedures Dealing with Unique IDs

GFR provides the following procedures for generating universal unique IDs:

gfr-check-uuids-on-cloned-item

gfr-universal-unique-id

gfr-check-uuids-on-cloned-item

Provides dispatch to the gfr-initialize and gfr-copy methods after programmatic cloning.

Synopsis

gfr-check-uuids-on-cloned-item
(*item*: class item)

Argument	Description
item	An item that has just been created by programmatic cloning.

Description

You must call this procedure after creating an item, using a programmatic clone action, if the item is a gfr-object-with-uuid or a gfr-message-with-uuid, or if it contains an item of these classes in its workspace hierarchy.

Internally, this procedure has an efficient way to check for the existence of ID-bearing items in the workspace hierarchy. If the item you cloned has a subworkspace and you are unsure about the possible presence of ID-bearing items in this hierarchy, do not bother to check for the presence of such an item before calling this procedure; simply call it, and the checking will be done for you.

Example

The following code fragments illustrate the correct use of this procedure:

```
create a item Bar2 by cloning Bar;
call gfr-check-uuids-on-cloned-item(Bar2);
```

```
create a kb-workspace WS2 by cloning WS1;
call gfr-check-uuids-on-cloned-item(WS2);
```

gfr-universal-unique-id

Generates a universal unique identifier.

Synopsis

```
gfr-universal-unique-id ()
    ->  uuid: text
```

Return Value	Description
uuid	A universal unique ID.

Description

This procedure generates a UUID with the format discussed in Unique ID Format.

Example

The following generates a UUID:

```
UUID = call gfr-universal-unique-id();
```

File Parsing and Miscellaneous Functions and Procedures

GFR provides the following procedures and functions for parsing the contents of files and for returning the coordinates of item edges:

gfr-bottom

gfr-convert-value-list-to-string

gfr-left

gfr-load-file-into-list

gfr-parse-string-into-value-list

gfr-right

gfr-top

gfr-bottom

A function that returns the location in workspace coordinates of the bottom edge of an item on a workspace.

Synopsis

gfr-top
(*item*: class item)
-> *coordinate*: integer

Argument	Description
item	The item whose bottom edge you want to locate.

Return Value	Description
coordinate	An integer representing the coordinate for the bottom edge of the item.

Description

This function returns the integer coordinate for the bottom edge of an item on a workspace. The coordinate is measured from the origin (0, 0) of the workspace.

gfr-convert-value-list-to-string

Converts a value list into a string suitable for saving to a file, using the system procedure g2-write-line-in-gensym-charset.

Synopsis

gfr-convert-value-list-to-string
(*values*: class value-list, *separator*: text, *clear-list*: truth-value,
client: class object)
-> *string*: text

Argument	Description
values	A list of values that are to be converted to a string.
separator	A character used to separate the values in the list.
clear-list	A truth-value indicating whether the value list is to returned empty or with its elements intact.
client	The client originating this call.

Return Value	Description
string	The text string that is generated.

Description

You use this procedure to generate a text string by concatenating the elements of a value list, or any subclass of value list, using the given separator between elements. This procedure is the inverse of gfr-parse-string-into-value-list, and follows the same rules for handling G2's value types. The rules are as follows:

- Truth values, integer, and symbols are written out as is.
- Floating point numbers are expressed to the full available precision.
- Texts are delineated with quotation marks, and quotes that are part of the text are replaced by two consecutive quotation marks.

Because of these conversions, the resulting string is not necessarily equivalent to simple text concatenation using G2's text concatenation operators.

For correct handling of multiple-line texts or texts with special characters, you must use g2-write-line-in-gensym-charset, rather than g2-write-line, if you write the resulting string to a file.

Often the separator is a comma or a comma followed by a space, but you can use any G2 text. Set the argument *clear-list* to true if you want this procedure to return your value list empty, which can be useful when you are using this procedure in an iteration.

Example

Suppose we have a value-list containing the following elements:

Element	Value	Type
0	true	truth-value
1	false	truth-value
2	!nv	text
3	1.0e32	float
4	ab cde	text
5	abc	symbol
6	Bob says “hi”	text
7	1	integer
8	!nv	text

The following text fragment opens the file foo.text, generates a string, and writes the string to the file:

```
Stream = call g2-open-file-for-write("foo.text");
String = call gfr-convert-value-list-to-string(values, ", ", false, win);
call g2-write-line-in-gensym-charset(Stream, String);
call g2-close-file(Stream);
```

The file foo.text contains the following line of text:

```
true, false, , 1.0e32, "ab\ SScde", ABC, "Bob says ""hi""", 1, ,
```

gfr-left

A function that returns the location in workspace coordinates of the left edge of an item on a workspace.

Synopsis

gfr-top
    (*item*: class item)
    -> *coordinate*: integer

Argument	Description
item	The item whose left edge you want to locate.

Return Value	Description
coordinate	An integer representing the coordinate for the left edge of the item.

Description

This function returns the integer coordinate for the left edge of an item on a workspace. The coordinate is measured from the origin (0, 0) of the workspace.

gfr-load-file-into-list

Loads the contents of a comma-separated-value file into a value list.

Synopsis

gfr-load-file-into-list
(*stream*: class g2-stream, *data:* class value-list,
values-per-line: class integer-list, *progress:* class integer-parameter,
allow-other-processing-interval: integer, *client*: class object)
-> *outcome*: truth-value

Argument	Description
stream	A g2-stream that was generated by calling g2-open-file-for-write or g2-open-file-for-read-and-write.
data	A value list that receives the values read from the file.
values-per-line	An integer list that receives the number of values per line in the file.
progress	An integer parameter that is incremented as the save proceeds, which can also be used to interrupt this procedure.
allow-other-processing-interval	An integer indicating how many lines should be read before allowing other processing.
client	The client originating this call.

Return Value	Description
outcome	A truth-value indicated whether the load completed.

Description

This procedure loads the contents of a file into a value list. The file that is written must be opened prior to this call, and is provided as a g2-stream object. For more information on streams, refer to the *G2 System Procedures Reference Manual*.

The format of the file must be as follows:

- The file can have any number of lines.
- The values on each line are separated by a comma.
- The number of values on a line may vary.
- Texts are explicitly quoted, embedded quotes within a text are doubled (each quote is replaced by a two quotes).
- Line breaks embedded within texts are represented as the character sequence "@L" or "\ SS."
- Special characters appearing in texts are represented in their encoded form, as given in the *G2 Reference Manual*, for example, the trademark symbol is represented as ~:

The texts read from the file are converted into the G2 value types according to the rules explained in the description of gfr-parse-string-into-value-list. The values are appended to the value-list *data*, which you provide. As each line of the file is read, the number of values on the line is appended to *values-per-line*. Neither *data* nor *values-per-line* are cleared when this procedure is called.

This procedure periodically allows other processing (see the *G2 Reference Manual* for discussion of allow other processing statements). Just before allowing other processing, the *progress* parameter is updated to indicate the percentage complete of the load. *Progress* has a final value of 100 when the load is finished. If you start (rather than call) this procedure, you can set up a monitor that receives control when *progress* is updated, by using a wait until *progress* receives a value statement.

The other use of the *progress* parameter is to abort a load in progress. If you delete *progress* before the completion of the load, the load will be aborted at the next allow other processing break. If the load is aborted, this procedure returns false. Otherwise, it returns true.

If you want G2 to continue other scheduled activities while the file load is in progress, set the *allow-other-processing-interval* argument to a small number of lines. You may set this parameter to a large number if you want the procedure to run without interruption.

Example

Suppose we have a file that contains the following lines:

```
true, false, , 1.0e32
"ab\ SScde", abc, "Bob says ""hi"""
1,
```

The following code fragment reads this file, assuming it is already open for read on *stream-1*:

```
create a value-list Data;
create an integer-list ValsPerRow;
create an integer-parameter Progress;
call gfr-load-file-into-list(stream-1, Data, ValsPerRow, Progress, 20, Win);
```

As a result of this call, ValsPerRow = (4, 3, 2) and Data contains the following elements:

Element	Value	Type
0	true	truth-value
1	false	truth-value
2	!nv	text
3	1.0e32	float
4	ab cde	text
5	abc	symbol
6	Bob says "hi"	text
7	1	integer
8	!nv	text

Note that, in this case, the processing inside gfr-load-file-into-list occurred as a single uninterrupted activity because the allow-other-processing-interval (20) was greater than the number of lines in the file (3).

gfr-parse-string-into-value-list

Converts a line of text into a value list according to parsing rules based on G2's value types.

Synopsis

gfr-parse-string-into-value-list
(*input-text*: text, *data:* class value-list, *separator:* text, *client*: class object)

Argument	Description
input-text	The text that is to be parsed.
data	A value list that receives the values parsed from the *input-text*.
separator	The text that separates the values in the *input-text*.
client	The client originating this call.

Description

This procedure converts a text into one or more G2 values, which are appended to the value list *data*. When this procedure is called, the input text is first separated into substrings, or tokens, delimited by the specified separator, ignoring separators embedded in text strings. Each token is converted to a value, using the following rules:

- If the token is the empty string or consists only of white space, the text *!nv* is inserted into *data*.
- If there are opening and closing quotes, the token is converted to a text by removing the surrounding quotes, and replacing any embedded double quotes with one quotation mark, and replacing the line break sequence <lb> with an actual line break.
- If the token begins with a quantity, it is converted to a quantity using G2's quantity operator, which can return either an integer or a float.
- If the token is true or false, it is converted into a truth-value.
- Otherwise, the token is converted to a symbol using G2's symbol function.

If the last step fails to yield a valid token, the text !nv is inserted at the end of *data*.

Example

See gfr-load-file-into-list for examples of the text conversion in this procedure.

gfr-right

A function that returns the location in workspace coordinates of the right edge of an item on a workspace.

Synopsis

gfr-right
(*item*: class item)
-> *coordinate*: integer

Argument	Description
item	The item whose right edge you want to locate.

Return Value	Description
coordinate	An integer representing the coordinate for the right edge of the item.

Description

This function returns the integer coordinate for the right edge of an item on a workspace. The coordinate is measured from the origin (0, 0) of the workspace.

gfr-top

A function that returns the location in workspace coordinates of the top edge of an item on a workspace.

Synopsis

gfr-top
(*item*: class item)
-> *coordinate*: integer

Argument	Description
item	The item whose top edge you want to locate.

Return Value	Description
coordinate	An integer representing the coordinate for the top edge of the item.

Description

This function returns the integer coordinate for the top edge of an item on a workspace. The coordinate is measured from the origin (0, 0) of the workspace.

Index

G

H

I

K

L

W