G2 Menu System

User's Guide

Version 2015

G2 Menu System User's Guide, Version 2015

December 2015

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

Part Number: DOC024-1200

Contents Summary

Contents

Preface

Describes this user's guide and the conventions that it uses.

About this Guide

This guide contains complete information about the G2 Menu System (GMS) and shows you how to use GMS at any supported level. This guide:

- Introduces the G2 Menu System and describes the menus and associated capabilities that it provides.
- Describes the GMS user interface and shows you how to use it to define menus graphically.
- Describes the GMS Application Programmer's Interface (API) and shows you how to use it to define and manage menus programmatically.
- Lists all GMS API functions and their signatures in a reference dictionary.
- Includes a glossary of all GMS terms and concepts.

Audience

This guide assumes that 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 that you have a general familiarity with menu systems as seen from the user's viewpoint. It does not assume an understanding of menu system internals on any platform.

Organization

This guide contains sixteen chapters, two appendixes, a glossary, and an index, in five parts:

	Title	Description
Part I	**Introduction**	
1	Overview of the G2 Menu System	Introduces the G2 Menu System (GMS) and defines GMS terms and concepts.
2	Getting Started	Describes the requirements for running GMS, how to install GMS, the module structure of GMS, and the GMS Demo.
Part II	**Creating a Menu**	
3	Defining a Simple Menu	Shows how to define a GMS menu by cloning GMS template objects, linking them into a hierarchy that defines the structure of the menu, setting the attributes of the template objects, and defining the actions to be taken when each menu choice is made.
4	Compiling the Menu Specification	Shows how to compile and display menus and how to reference the parts of a menu.
5	Using Specialized Templates	Describes how to use the two predefined templates provided by GMS to change the user mode and to display a named workspace.

	Title	Description
14	Customizing the GMS Interface to the User	Shows how to set GMS properties for individual G2 users. These properties include: font and separator size, help display, text and background colors, navigation modes, menu blinking, initial menu, and menu language.
15	Specifying the Interface between GMS and G2	Shows how to configure GMS to work with and extend G2 capabilities such as accessing a compiled resource from G2, implementing keyboard accelerators, and handling mouse events for popup menus.
16	Managing GMS Programmatically	Shows how to manage programmatically the internal operations of GMS. You can use the techniques described to manage menu building and compilation, change global settings, manage user preferences, and reset GMS without resetting G2.
Part IV	**Appendixes**	
A	GMS API Reference	Describes all supported GMS API calls in alphabetical order.
B	GMS Common Names and Formal Names	Gives the common name and the formal name of every GMS entity.

A Note About the API

The GMS 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 GMS release that includes them.

Therefore, it is essential that you use GMS 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 GMS may change, or in the present, because the code may not correctly manage the internal operations of GMS.

If GMS 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
<u>*text-string*</u>	Return values of G2 procedures and methods in syntax descriptions
File Name, OK, Apply, Cancel, General, Edit Scroll Area	GUIDE and native dialog fields, button labels, tabs, and titles
File > Save Properties	GMS and native menu choices
workspace	Glossary terms

Convention Examples	Description
`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 <u>*underlined*</u>. 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)
-> <u>*transferred-items*</u>: 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`

Introduction

Overview of the G2 Menu System

Introduces the G2 Menu System (GMS) and defines GMS terms and concepts.

Introduction

This chapter introduces the G2 Menu System (GMS) and defines GMS terms and concepts. Be sure you are familiar with the information in this chapter before you read the rest of this guide.

What is GMS?

GMS is a G2 utility that provides the tools to enable you to create menus similar to the ones you see on PCs. You design these menus using **menu specifications**.

You can create GMS menus dynamically or save them as permanent parts of your G2 application. These menus can accept user input from the mouse or keyboard, and you can update them in real time when data changes in your application.

GMS is composed of:

- A graphical user interface for organizing the structure of your menus.
- An Application Programmer's Interface (API) that enables you to display, configure, and manipulate menu entries programmatically.

If you want your menu text to appear in a language other than English, you can use Gensym Foundation Resources (GFR) to internationalize GMS menus.

When you click on an object in G2, the object normally displays a G2 menu. When you use GMS, you can configure classes of objects to use GMS menus rather than G2 menus. Classes so configured will display GMS popup menus instead of G2 menus. Classes that are not so configured continue to use G2 menus.

GMS Menu Types

The types of GMS menus include:

- **Pulldown menus** that are arranged in a **menu bar** at the top of a G2 window.
- **Popup menus** that are associated with a G2 item and display when needed.

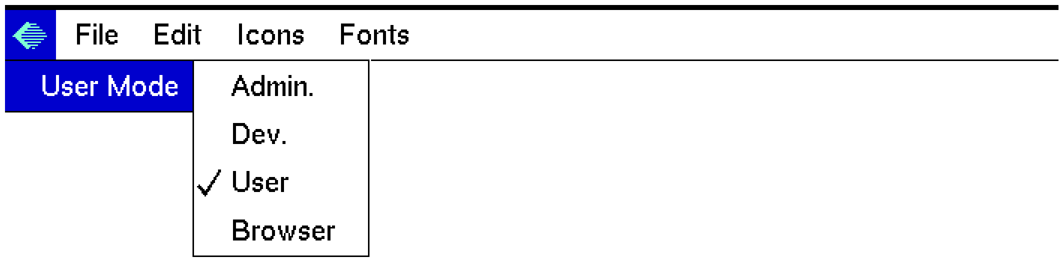
Both pulldown menus and popup menus can contain **cascade menus** or "walking menus" that display choices for a menu entry.

When this guide refers to a **GMS menu**, the reference applies equally to pulldown menus, popup menus, and cascade menus.

Popup and cascade menus both appear only when needed. Because they come and go, this guide refers to popup and cascade menus as **transient menus**.

Pulldown Menus in Menu Bars

A pulldown menu appears in a horizontal menu bar that extends along the top of a G2 window, as shown in the following figure.



GMS automatically positions a menu bar at the top of the G2 window. If the window is not wide enough to show all menu entries in one row, GMS automatically wraps the menu onto additional rows as needed.

GMS automatically positions a transient menu to be near the site of the mouse click that activated it and to be completely visible within the G2 window. If GMS cannot display the complete menu at an acceptable location, it automatically displays the menu with scroll indicators. If you hold the mouse pressed over a scroll indicator, the menu scrolls to display additional entries.

Menu bars typically remain on display over time. You can use API calls to make them appear, change, and disappear as needed, but they do not change merely because you have chosen an entry. A G2 window can contain at most one menu bar at a time.

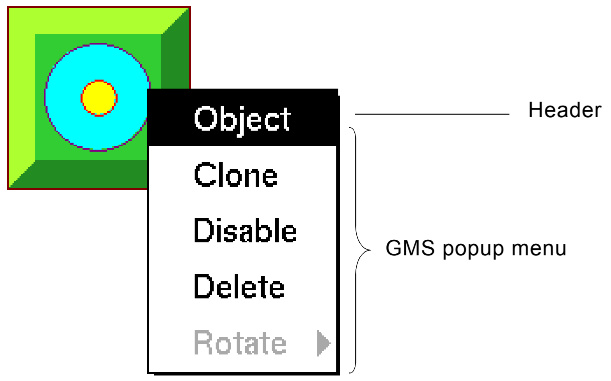
By convention, a menu bar:

- Displays additional menu entries that perform actions.
- Contains either a label or an icon, but not both.
- Contains entries that are always enabled.

GMS does not enforce these conventions, However, we recommend that you follow them for consistency with Windows standards.

Popup Menus

A popup menu is a freestanding menu that can appear anywhere in a G2 window in response to a mouse click. A popup menu is associated with an item in a G2 window. Any G2 item can be configured to display a GMS popup menu instead of its G2 menu. An example is shown in the following figure:

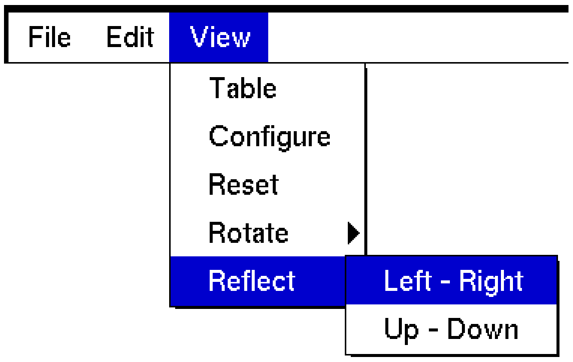


Popup menus appear only when they are needed. A popup menu typically appears in response to a mouse click, and disappears as soon as you have chosen an entry from the menu or from a cascade menu that is subsidiary to it.

A popup menu can have a title block called a **header** that describes the purpose of the menu. Menu bars and cascade menus do not have headers. In the example of a popup menu, the term “object” is a header for the popup menu.

Cascade Menus

A cascade menu is a subsidiary menu that appears when you choose a higher-level menu entry. The presence of a cascade menu in one of the pulldown menu entries is indicated by a right-pointing arrow. To display the cascade menu, click on the arrow.



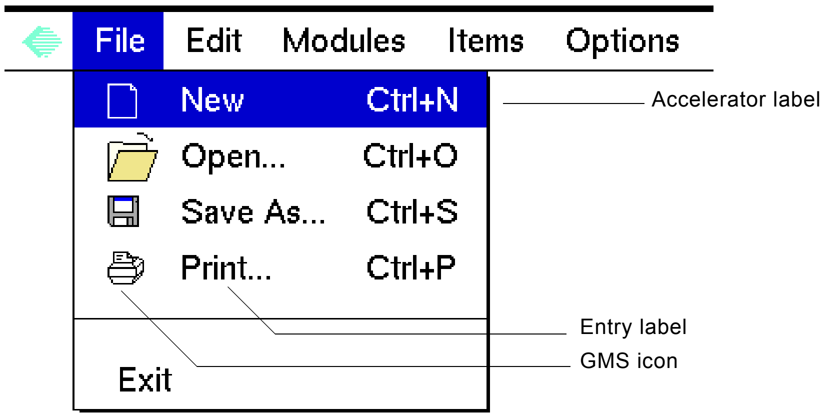
Cascade menus are similar in appearance to popup menus, except that a cascade menu cannot have a header. Cascade and popup menus follow the same conventions for marking entries.

A cascade menu appears when you choose a cascading entry from a higher-level menu. The higher-level menu remains on display after the cascade menu appears. Cascade menus can contain cascading entries, to a maximum of 256 levels.

When you choose an entry from a cascade menu, the menu and any higher-level transient menus immediately disappear. If the highest-level menu is a menu bar, it remains on display unless an API call explicitly removes it.

GMS Menu Entries

Every GMS menu includes one or more menu entries. A **menu entry** is a rectangular cell that can contain an entry label, an icon, and/or an accelerator label, as shown in the following figure:



Components of a Menu Entry

Each menu entry contains one or more of these components:

- **Entry label**: Text that describes what will happen if you choose the entry. Examples:
  - Open
  - Save As
- **GMS icon**: An icon that denotes the entry’s effect graphically.
- **Accelerator label**: Indicates a keystroke that has the same effect as choosing the menu entry. Examples:
  - Ctrl+O
  - Ctrl+S

Types of Menu Entries

The three types of menu entries are:

- **Cascading entry:** GMS automatically displays a cascade menu when you select a cascading entry. GMS automatically marks every cascading entry in a popup menu with a triangular pointer on the right side.
- **Leaf entry:** Choosing a leaf entry executes a user-defined procedure, known as a **callback procedure**. Such a procedure can either display a dialog that requests additional information or take an action immediately.

Note Only a leaf entry can specify an accelerator label.

- **Dialog entry:** A leaf entry whose callback procedure displays a dialog is known as a dialog entry. By convention, the menu designer marks a dialog entry in a popup menu by suffixing an ellipsis (...) to the entry's label. Examples:
  - Open...
  - Save As...

  This guide does not include the ellipsis when referring to the entry label of a dialog entry.

Selecting and Choosing Menu Entries

To use GMS menus, you use the mouse to select and choose menu entries. When you **select** a menu entry, you press the mouse button while the mouse pointer is over the menu entry or move the mouse pointer over it with the mouse button already pressed. GMS indicates a selected entry by changing its color.

After you select a menu entry, you can **choose** it by raising the mouse button with the mouse pointer still over it. You can also choose a menu entry by clicking the mouse button while the mouse pointer is over the menu choice.

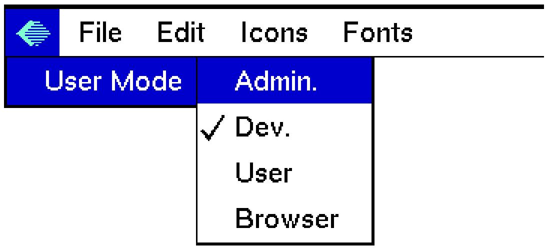
Enabled and Disabled Menu Entries

A menu entry can be enabled or disabled. For example, you might want to disable a menu entry if it does not apply to the current situation, such as no object selected for a rotate menu entry. An **enabled menu entry** is fully functional.

A **disabled menu entry** appears but is not selectable and cannot perform its task. If you select a disabled cascading menu entry, it does *not* display its cascade menu. If you choose a disabled leaf entry, it does *not* execute its callback procedure.

Adding a Check to a Leaf Entry

You can set any leaf entry in a transient menu to be checked or unchecked. A **checked menu entry** has a check mark to the left of its entry label; an **unchecked menu entry** has none. You can use check to indicate anything that is useful, such as the toggling of a state. You can inform the user that a particular state is selected, such as a particular user mode, as this figure shows:



Associating Help with a Menu Entry

You can associate a **Help label** with any menu entry. A Help label provides information about the entry that can help the user to understand what it does. When GMS is configured to display Help information, and a menu entry has a Help label, GMS displays the Help label in a **Help bar** whenever the menu entry is selected. The Help bar appears at the bottom of the G2 window, and is visible only when it is needed.

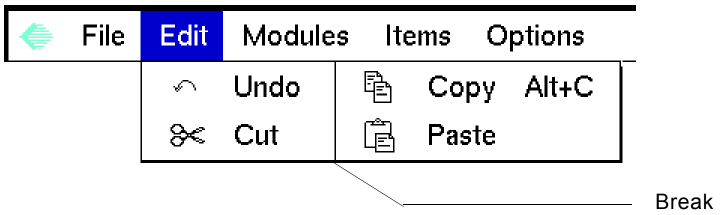
GMS Menu Customizations

GMS offers several ways to customize menu appearance and behavior.

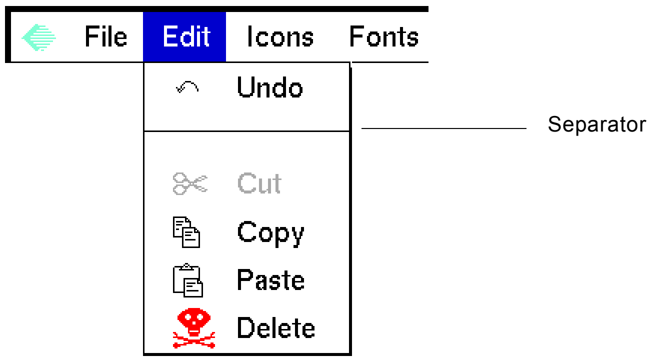
Using Dividers in Menu Panels

The entries in a GMS menu constitute the menu's **panel**. You can separate parts of a menu's panel by using menu **dividers**. There are three types of menu dividers:

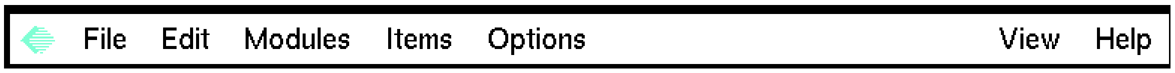
- **Breaks:** By default, a menu bar panel occupies a single row, and a transient menu panel occupies a single column. You can divide a menu bar into multiple rows and a transient menu into multiple columns by inserting a break into the menu before each entry that is to begin a new row or column.



- **Separators:** By default, the panel of a transient menu has no subdivisions. You can subdivide a transient menu panel into groups of entries by inserting a separator between any adjacent entries. GMS draws a horizontal line across the panel between the two entries.



- **Justifiers:** By default, the entries in a menu bar are left-justified. You can cause any or all entries to be right-justified by inserting a **j**ustifier into the menu before the first entry that is to be right justified. In the example which follows, the justifier is placed between the templates for Options and View in the menu specification. When a break follows a justifier in a menu bar, the new row is also right-justified.



Choosing a Menu Entry

GMS menus are structured hierarchically. The top of the hierarchy is *always* a menu bar or a popup menu. Top-level menu entries might display cascade menus whose entries, when selected, might display lower-level cascade menus, and so on. The lowest level of the hierarchy is the leaf entry, which executes the menu choice when chosen.

GMS supports two standard menu navigation configurations for menus:

- **Walking menus:** allow you to drag your mouse over a top-level menu entry, and over any cascade menus to a leaf entry. When you raise the mouse button over a leaf entry, GMS executes the menu choice.
- **Sticky menus:** allow you to click on menu entries rather than dragging over them. Clicking a cascading entry displays its cascade menu. Clicking the entry again hides the menu. Clicking a leaf entry executes the menu choice.

Walking menus are always available; sticky menus are optional. When sticky menus are enabled, you can mix the two techniques as desired.

Defining GMS Menus Statically or Dynamically

You can define a GMS menu statically or dynamically. A **static menu specification** completely defines a menu in advance. A **dynamic menu specification** defines a menu only partially: the definition includes templates that programmatically complete the menu definition just before the menu is displayed.

A static menu definition is appropriate for menus that are always the same. Dynamic menus provide more flexibility but are slower because GMS must construct some or all of each menu immediately before displaying it.

You can use the GMS API to display any statically or dynamically defined menu in any G2 window at any time. You cannot associate a GMS menu with a particular workspace but only with a G2 window as a whole.

When you display a menu defined by a dynamic menu specification, GMS by default compiles and builds the menu immediately before displaying it.

Customizing GMS Menus Globally or for Each User

You can configure GMS to automatically display a menu bar when a user logs in to G2, either directly or via Telewindows. The menu bar displayed can depend on the current G2 application and/or the identity of the user.

You can customize the behavior, language, and appearance of GMS menus and change this customization at any time while running the KB. You can customize:

- The language in which labels appear to the user.
- The font size used in the menus.
- The colors used to indicate menu entries that are selected, disabled, or neither.
- The availability of sticky menus.
- The display of Help bar information.

You can configure appearance per-application, per-user, or both. Thus a G2 application can use GMS to display menus in different languages to different users in different countries.

GMS provides all standard G2 methods for customizing the user environment. For example, you can restrict menus based on the user mode of each G2 window.

Displaying GMS Menus in Telewindows

G2 provides tools for creating standard end-user interfaces for G2 applications when viewed through Telewindows. The **Native Menu System (NMS)** allows you to create custom pulldown menus and popup menus by:

- Rendering menus created using the GMS as Windows menus, when viewed through standard Telewindows on a Windows platform.
- Providing an API for creating and manipulating Windows menus, using G2 system procedures.

To create custom menus, the G2 developer can choose to use GMS, which provides a graphical interface, or G2 system procedures, which provides a programmatic interface. Both techniques support standard menu features such as menu bars, submenus, and popup menus.

While you can use the NMS API to implement almost everything that you can implement using GMS, GMS provides a more intuitive, graphical environment for defining menus. In addition, it provides built-in tools that the NMS API requires specific programming to accomplish. Which approach you use depends on your preference.

To use native GMS menus, you simply load a KB that defines GMS menus; the menus automatically render as standard Windows menus when viewed through Telewindows. You can also choose to display GMS menus in Telewindows, using their classic G2 interface.

For examples of native GMS menus and other information on how to use native GMS menus, see Windows Menus in the *G2 Reference Manual*.

Getting Started

Describes the requirements for running GMS, how to install GMS, the module structure of GMS, and the GMS Demo.

Introduction

GMS is a G2 utility that provides the tools to enable you to create menus similar to the ones you see on PCs. This chapter describes the requirements for running GMS and describes the GMS Demo.

Installing GMS

GMS is provided as the module *gms.kb* in the *utils* subdirectory of the *kbs* directory under the *g2* directory.

To install GMS:

➔ Merge *gms.kb* into an existing KB manually by choosing Merge KB from the Main Menu.

or

➔ Merge *gms.kb* into another KB automatically by specifying it as a directly required module in the KB's Module Information Table.

You can merge GMS into any modularized knowledge base that does not duplicate a name used by GMS. All GMS names start with the prefix gms-.

GMS works only when G2 is running. After merging GMS, restart G2.

Requirements for Running GMS

GMS contains a table of version information accessible from the GMS Top Level workspace. To view this information, click on the copyright symbol (©) in the title section of the GMS Top Level workspace. The table contains information including the minimum version of G2 in which the current version of GMS will run.

To view the version information:

1 Display the GMS Top Level workspace.

2 Click the copyright symbol (©) in the title section.

or

```
➔ gms-get-version
    ( )
    -> current-version: text
```

Returns the current version of GMS, for example, 8.0 Rev. 0.

Starting GMS

GMS works only when G2 is running. After merging GMS, restart G2.

To start GMS:

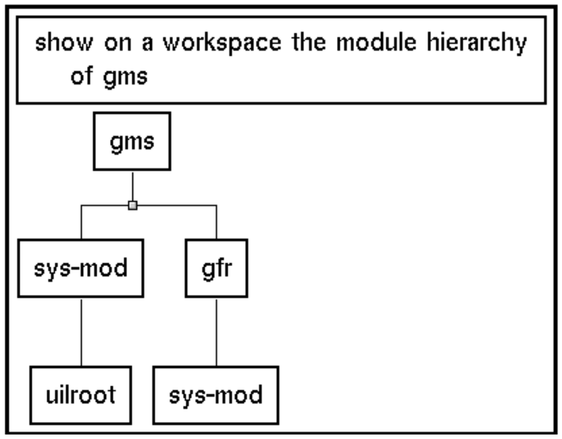
➔ Restart G2.

GMS Modules

The following table describes the GMS modules.

Module	File Name	Contents
gms	*gms.kb*	Definitions and API support to enable you to create GMS menus
sys-mod	*sys-mod.kb*	System procedure library
gfr	*gfr.kb*	Tools to internationalize your GMS menus
uilroot	*uilroot.kb*	Definitions and API support for navigation buttons

The following Inspect workspace illustrates the GMS module hierarchy:



The GMS Demo KB

GMS provides a demo KB in a file called *gmsdemo.kb*, also found in the *kbs* directory. The demo KB loads GMS as a required module and contains examples of all GMS capabilities.

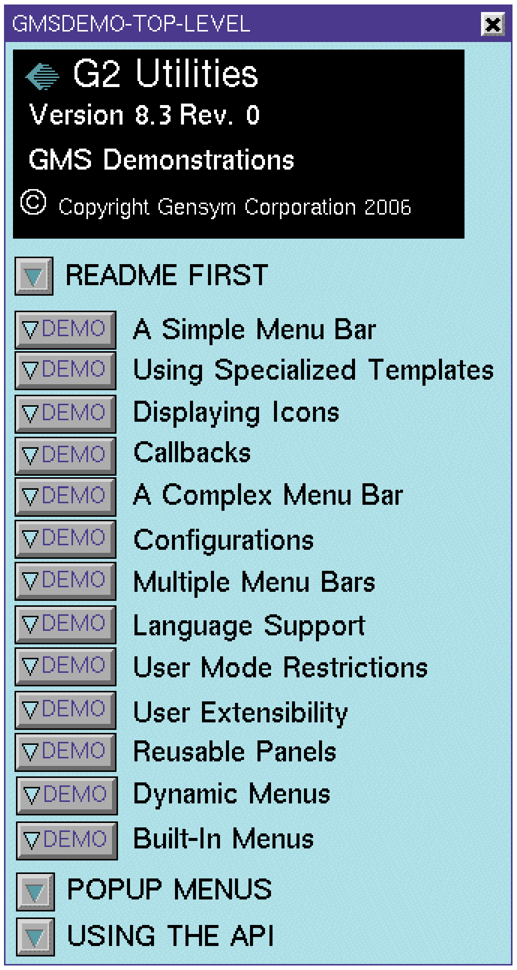
On the GMS demo KB you can view online various GMS capabilities as you read about them in the documentation. As you browse the demo KB, you can see how to construct menus and how to define their behaviors by setting attributes and using API calls.

The chapters of this guide are arranged to cover the material in approximately the same order as the demo. The introduction to each chapter mentions the name of the demo where examples of the current topic can be found.

Note Be sure you are in **developer** mode when you use the GMS demo KB. If you are not in developer mode, clicking buttons will display tables rather than access the demos.

When you load the demo KB and start G2, the GMS Demo workspace appears. The workspace name is gmsdemo-top-level and it is labelled "GMS DEMO." It contains a list of the demos, plus two submenus for popup menus and the API. There is a readme section explaining that the demos are arranged by increasing difficulty.

The following figure shows the GMS demo top level workspace:



GMSDEMO-TOP-LEVEL

G2 Utilities

Version 8.3 Rev. 0

GMS Demonstrations

© Copyright Gensym Corporation 2006

README FIRST

DEMO A Simple Menu Bar

DEMO Using Specialized Templates

DEMO Displaying Icons

DEMO Callbacks

DEMO A Complex Menu Bar

DEMO Configurations

DEMO Multiple Menu Bars

DEMO Language Support

DEMO User Mode Restrictions

DEMO User Extensibility

DEMO Reusable Panels

DEMO Dynamic Menus

DEMO Built-In Menus

POPUP MENUS

USING THE API

Using the Demo KB

To activate a GMS demo:

➔ Click the appropriate DEMO button.

Any workspaces relating to other demos disappear.

To dismiss the GMS Demo workspace or any demo subworkspace:

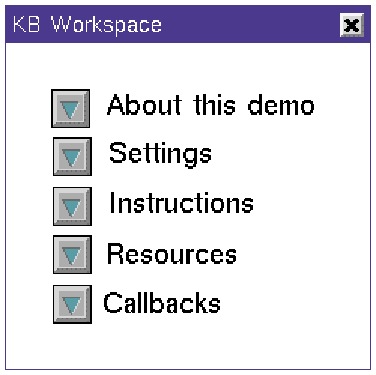
➔ Choose Hide Workspace from the workspace's G2 menu.

To redisplay the GMS Demo workspace:

➔ Choose Main Menu > Get Workspace > gmsdemo-top-level.

Navigating a Demo

When you click a DEMO button in the GMS DEMO workspace, the following workspace appears showing the five menu buttons that are common to all the demos:



Clicking each of the five buttons displays a subworkspace that relates to the demo:

- **About this demo**: General information about the purpose of the demo.
- **Settings**: The global settings and user preferences in effect for the demo.
- **Instructions**: Specific instructions for executing the demo.
- **Resources**: The GMS menu specification(s) of the menu(s) that the demo displays.
- **Callbacks**: The procedure(s) that GMS calls when you choose menu entries while running the demo.

Part II

Creating a Menu

Chapter 8: Dynamically Defining and Changing Menus

Shows how to create and modify GMS menus in real time.

Chapter 9: Including Additional Features in a Menu

Shows how to include some additional features of GMS in your menu. These features include menu dividers, menu accelerators, menu help information, distributing menu specifications over several workspaces, and creating reusable menu definitions.

Defining a Simple Menu

Shows how to define a GMS menu by cloning GMS template objects, linking them into a hierarchy that defines the structure of the menu, setting the attributes of the template objects, and defining the actions to be taken when each menu choice is made.

Introduction

This chapter shows you how to build a GMS menu, using a GMS menu specification. The process has five stages:

- First you create a GMS menu specification by cloning template objects from the GMS palette.
- Then you then use specialized connection stubs to connect the templates into a hierarchy that defines the structure of the menu.
- You fill in the attributes for the template objects to define the behavior of the menu.
- You write callback procedures that define the action to be taken when each menu choice is selected.
- You add special GMS features to the menu.

As you work on your menu specification, you can move between the stages. Building a menu specification is iterative.

The principles described in this chapter apply to all GMS menu specifications. For simplicity, the chapter focuses on static menu specifications. Dynamically Defining and Changing Menus, describes techniques for specifying GMS menus dynamically.

To see online examples of the concepts covered in this chapter:

1. Load the *gmsdemo.kb* from the *kbs* directory.
2. Choose the demo called A Simple Menu Bar.

Components of a Menu Specification

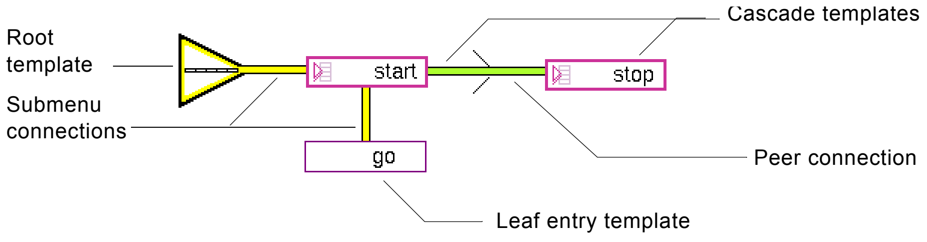
A GMS menu specification is a tree of objects called **template objects** linked together with GMS connections. GMS template objects are usually called **templates**.

- One of these templates, called the **Root template**, forms the root of the tree and contains attributes that affect the menu as a whole.
- The root template has one or more child templates, called **Cascade Menu templates** representing top-level menu entries.
- Each Cascade Menu template has one or more child templates, each representing a lower level menu entry. These child templates define the cascade menu.
- Templates that represent leaf entries have no child templates.

Two types of connections link these templates together:

- A **submenu connection** links a parent template to its *first* child template.
- A **peer connection** links two sibling templates.

A GMS menu specification begins with a root template which can be a a Menu Bar template or a Popup Menu template. This root template is linked by a submenu connection to a Cascade Menu template that defines the first entry in the menu bar or popup.



The only difference between the specification of a menu bar and the specification of a popup menu is the type of root template used. For a menu bar use a **Menu Bar template** and for a Popup menu use a **Popup Menu template**.

The remaining entries in the top level of a menu specification are Cascade Menu Templates. These templates are connected by peer menu connections in the same order in which the entries appear in the top level of the menu.

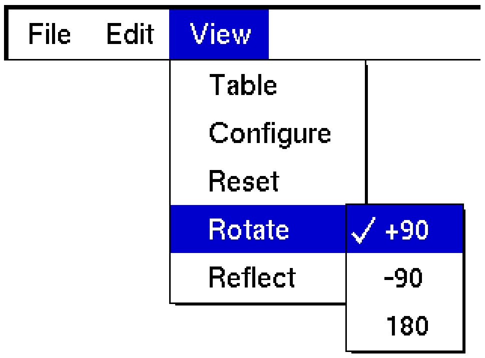
Each Cascade Menu template in the menu is connected by a submenu connection to the first entry in its cascade menu. This entry is connected by a peer menu connection to one or more entries linked by additional peer menu connections. This structure can continue recursively to a maximum of 256 levels of nested menus.

The graphical part of a menu specification defines the menu's structure, but does not specify the contents and behavior of the individual menu entries. All such information is supplied in the same way, by setting attribute values of the template objects that constitute the menu definition. These attributes, and the required values for each, are described in Specifying Menu Properties.

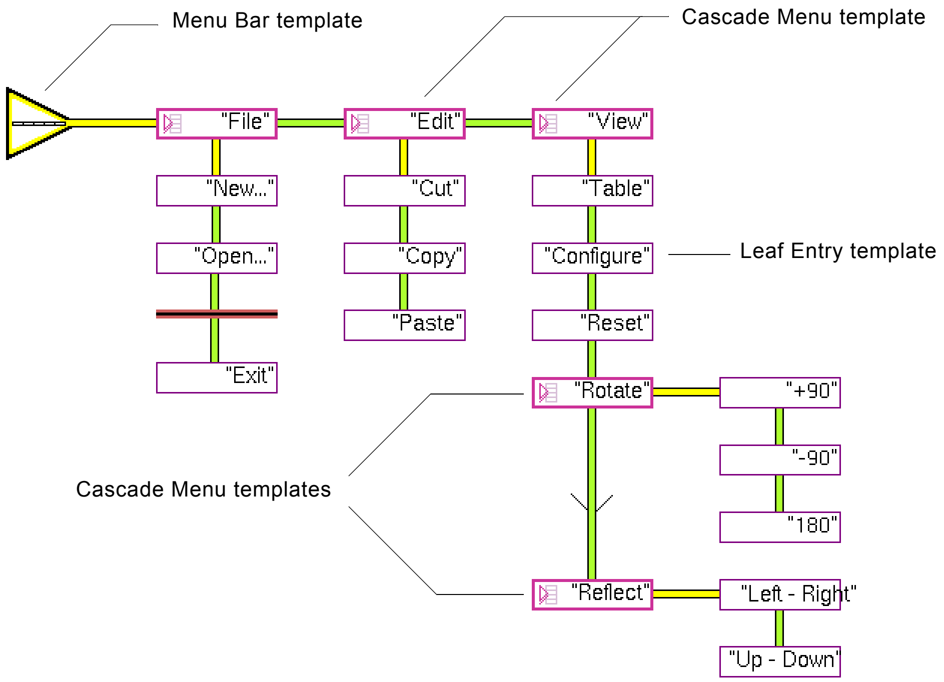
Example of a GMS Menu Specification

The following two figures show the structure of a typical menu bar and all of its associated cascade menus.

The first figure shows the menu bar with a pulldown menu selected along with a cascade menu that further defines the selections for the Rotate menu entry.



The second figure is a GMS menu specification showing the complete structure of the pulldown menus in this menu bar example. Observe the mapping between the menu structure and the graphical display of it in the GMS menu specification.



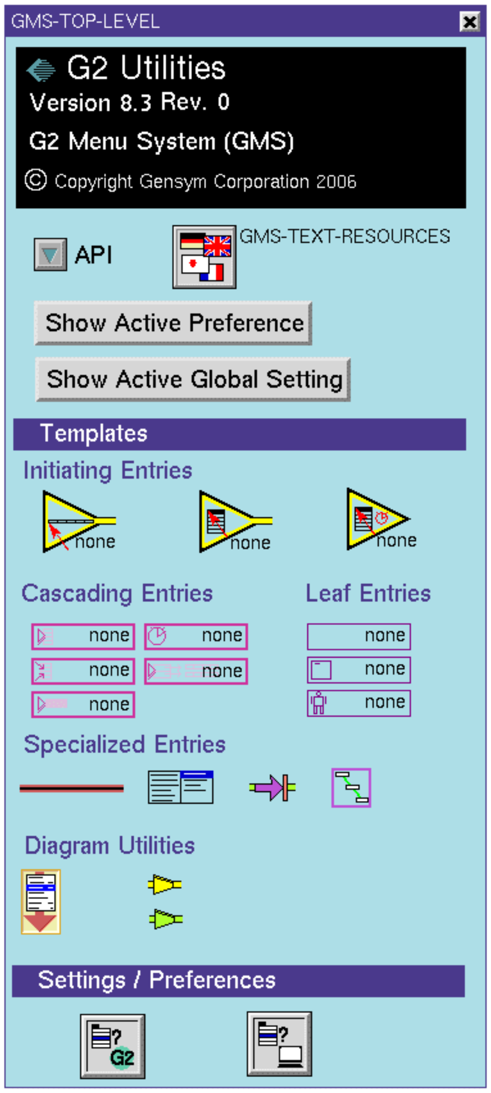
Displaying the GMS Palette

GMS provides all template objects on a palette called the **GMS Palette**.

To access the GMS Palette:

➔ Choose Maine Menu > Get Workspace > gms-top-level.

The following figure shows the GMS Palette. You can display a help label for each template object by holding the mouse button down while the mouse cursor is over the object.



The following table contains an illustration and a brief description of each template on the GMS palette.

GMS Template Objects

Icon	Template	Description
GMS Text Resources		
	Text resource group	The text resource group contains the menu translations for your menus. The default text resource is English. For more information see Internationalizing GMS Menus.
Initiating Entries		
none	Menu Bar template	The root template for a menu bar of pulldown menus. The template displays a menu bar. Clone this template first in the upper-left of your graphical menu specification for a menu bar. For more information see Components of a Menu Specification.
none	Popup Menu template	The root template for a popup menu. The template displays a popup menu. Clone this template first in the upper left of your graphical menu specification for a popup menu. For more information see Components of a Menu Specification.
none	Dynamic Popup template	The root template for a dynamic popup menu. The clock on the template indicates that this is a dynamic template. Clone this template to create a dynamic popup menu. For more information see Dynamically Defining and Changing Menus.

GMS Template Objects

Icon	Template	Description
Cascading Entries		
▷ none	Cascade Menu template	The template for top-level menu entries in a static menu. For more information see Components of a Menu Specification.
none	Dynamic Cascade template	The template for top-level menu entries in a dynamic menu. For more information see Defining and Displaying a Dynamic Popup Menu.
none	Reusable Panel template	The template for cascade menu entries that you intend to use for more than one menu entry. For example, you might want to use up and down options for two menu entries. For more information see Creating Reusable Cascade Menus.
▷ none	Switch Menu Bar template	A special case of dynamic cascade menu that enables you to switch between all compiled GMS menu bars. For more information see Switching Menu Bars.
▷ none	Built-in G2 Menu template	Allows you to create one of the built-in G2 menus. This menu entry is only supported in Telewindows.

Leaf Entries

none	Leaf Entry template	A final menu choice that executes a callback procedure.

GMS Template Objects

Icon	Template	Description
none	Show Workspace template	A menu entry that displays a named target workspace. For more information see Displaying a Workspace.
none	Change User Mode template	A menu entry that changes user mode (administrator, developer, browser, user, and others). For more information see Changing the User Mode.
Non Text Entries		
	Separator template	Displays a divider line between menu entries in a column.
	Break template	Splits a single column into two columns.
	Right Justifier template	When placed in a menu bar, aligns all entries after it to the right side of the menu bar.
	Place Holder	Specifies a connection between two menu entries in a panel that spans *two modules*. For more information see Extending Menu Specifications across Modules.
Diagram Utilities		
	SubPanel Container	Indicates that part of a menu specification resides on a subworkspace. For more information see Using SubPanel Containers to Distribute Menu Specifications.

GMS Template Objects

Icon	Template	Description
	Submenu Connection Post	Enables you to extend entries joined by a submenu connection across workspaces. For more information see Using SubPanel Containers to Distribute Menu Specifications.
	Peer Menu Connection Post	Enables you to extend entries joined by a peer menu connection across workspaces *or modules*. For more informations see Using Connection Posts to Distribute Menu Specifications.
Settings / Preferences		
	Global Setting	Specifies the GMS behavior for the entire G2 window or application. For more informations see Configuring Global GMS Characteristics.
	User Preference	Enables you to specify the GMS settings for individual users. For more informations see Customizing the GMS Interface to the User.

Creating Menu Templates

To create a GMS menu specification, you clone template objects from the GMS Palette, connect them to specify the menu's structure, and set attributes of the cloned templates as needed to define the properties of the constituent menus and menu entries. You don't have to completely define the structure before you set template attributes; you can go through many iterations of these tasks as you work on the menu specification.

You can create a GMS menu specification on any workspace. An application can contain any number of menu specifications on any number of workspaces.

You can plan your menu(s) in advance, or you can clone, connect, reconnect, and delete template objects at any time, and change their attribute values as desired. You can ask GMS to recompile a menu specification at any time.

Thus, you can develop menu specifications iteratively, building them, testing them, modifying them, and immediately testing them again. GMS signals any compilation and execution errors. By default, G2 reports any signalled error in the Operator Logbook.

To clone template objects from the palette to a workspace:

1 Click the mouse button over the template.

A copy of the template appears, attached to the mouse pointer.

2 Move the mouse cursor to the desired location on the workspace.

3 Click the mouse button to place the template on the workspace.

The template transfers to the workspace at the location you chose or at the nearest grid position if the snap grid is on.

Any workspace that contains one or more GMS template objects is called a **GMS workspace**. You can set GMS to position template objects on a GMS workspace to appear only at vertices of a regular grid. Such a grid is called a **snap grid**.

Note Before enabling a snap grid, place a template object on a GMS workspace.

To set a snap grid on a GMS workspace:

➔ Choose KB Workspace > GMS Grid On.

The snap grid is not visible on the workspace, but it affects the positioning of all GMS template objects. When you clone a template to the workspace, or move any template on the workspace, GMS adjusts the template to appear at the nearest point on the snap grid. Existing templates are unaffected unless you move them.

Once you set a snap grid on a workspace, the grid remains on until the next time you reset G2.

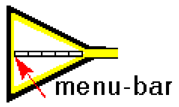
Note You can use the snap grid in any mode except administrator mode.

Using Root Templates

A GMS menu specification begins with a unique object called the **root template**. This template defines the menu as a whole; it does not correspond to any visible menu entry. GMS provides two types of root templates that are used in *static* menu specifications and a specialized root template used in *dynamic* menu specifications. For information on dynamic menus see Dynamically Defining and Changing Menus.

The Menu Bar Template

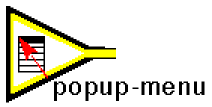
Use a Menu Bar template to initiate a menu bar specification for a pulldown menu. This template contains the attributes necessary to define a menu bar.



The Popup Menu Template

Use a Popup Menu template to initiate a menu specification for a popup menu.

For information on how to create a popup menu, see Defining Popup Menus.



Using Entry Templates

A menu specification contains an **entry template** for every entry in every menu. GMS provides several types of entry templates. The two that are most useful in static menu specifications are:

Cascade Menu Template

The cascade menu template defines an entry that displays a cascade menu when selected (a cascading entry).

Leaf Entry Template

The leaf entry template defines an entry that executes a callback procedure when chosen (a leaf entry).

Using Connections

GMS provides two types of connection for linking template objects into a menu specification: the submenu connection and the peer menu connection. The two types of connections have different colors; peer connections are green, while submenu connections are yellow.

Submenu Connection

A submenu connection connects two adjacent entries that represent a parent-child or superior-subordinate relationship in a menu structure, specifically connecting:

- A root template to the first menu template in its menu specification.
- A cascade menu template to the first entry template in its cascading menu.

Peer Menu Connection

Connects two adjacent entries in a sibling relationship at the same level in a menu specification. You use a peer menu connection to connect two adjacent cascade menu templates or two adjacent leaf entry templates.

Managing Connection Stubs

Every root template (except dynamic popup) on the GMS Palette has one connection stub, an output stub of class submenu-connection. However, entry and divider templates on the palette do not have connection stubs, because GMS cannot anticipate which stubs you will need in a given menu specification.

Instead, you create stubs when you need them and remove any that you do not use by choosing commands from templates' G2 menus. You can use these commands in any order that is convenient to provide the connection stubs you need and remove any that you do not use.

To add submenu connection stubs to a cloned template object:

➔ Choose add submenu stubs from the template's menu.

A submenu connection stub appears centered on each side of the template. The stubs are directed:

- The stubs on the top and left are input stubs.
- The stubs on the right and bottom are output stubs.

You can use these stubs as needed and ignore any that you do not need.

To add peer menu connection stubs to a cloned template object:

➔ Choose add peer stubs from the template's menu.

A peer menu connection stub appears centered on each side of the template. The stubs are directed:

- The stubs on the top and left are input stubs.
- The stubs on the right and bottom are output stubs.

You can use these stubs as needed and ignore any that you do not need.

To remove unused connection stubs from a cloned template object:

➔ Choose remove stubs from the template's menu.

All unused connection stubs disappear from the template object. Stubs on other templates are unaffected.

To remove unused connection stubs from all templates in a menu specification:

➔ Choose remove stubs from the menu of the specification's root template.

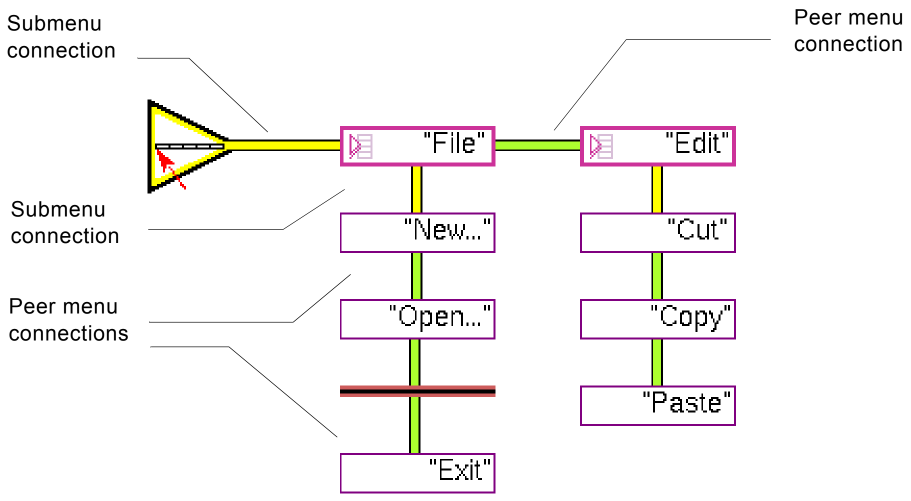
All unused connection stubs disappear from every template object in the specification. Stubs on other specifications are unaffected.

To connect the templates into a menu specification:

1. Use stub management commands to make connection stubs appear on templates as needed.
2. Connect the templates as needed to establish the desired structure. You can attach a connection to a template anywhere, whether or not a connection stub exists there.
3. Remove any unused connection stubs.

Note GMS does not let you mix peer and submenu connections, so you cannot create an illegal structure.

The following diagram identifies the connections in a menu specification. The root entry connects to the first cascade entry template, using a submenu connection. The first cascade entry template connects horizontally to the second cascade entry template, using a peer menu connection. These cascading entry templates connect to the first entries in their menus, using submenu connections. Leaf entries further down the menu structure connect to each other using peer menu connections.



Specifying Menu Properties

To define a menu's properties, you set attributes as needed in the template objects that define the menu. When you do not want to define some component of a menu entry, such as its label or its icon, specify the relevant attribute value as none.

Attributes of a Root Template

The attribute settings of the root template determine the behavior and appearance of the entire menu. The following attribute table describes the attributes that are common to the Menu bar template and the Popup menu template, the two root templates used for static menus.

Using attribute settings you can restrict a menu's use, define the menu's appearance, set access keys, provide help information, enable or disable the menu on display, and set default behavior for menu choices.

Attribute	Description
gms-restricted-modes	Lists the user modes in which this menu is displayed or lists modes in which this menu is not displayed.
Allowable values:	One or more symbols
Default value:	A symbol-array with an initial value of G2.
Notes:	To restrict the menu when a user is in a given mode, specify the mode as a symbol. To restrict an entry when the user *is not* in a not in a given mode, specify the mode as a symbol prefixed by a tilde (~). For information on using the tilde see .
gms-index	Unique index number for this template object.
Allowable values:	Not user settable.
Default value:	0
Notes:	You may access the gms-index but do not change its value. For more information see Referencing Menus and Menu Entries.
gms-user-key	Holds any value assigned by the user and can be used to access the template.
Allowable values:	Any symbol or value
Default value:	none

Attribute	Description
gms-text-resource-group	Specifies the text resource group that GMS will use for translations.
Allowable values:	The name of any gms-text-resource-group
Default value:	none
Notes:	For more information see Specifying the Translation Dictionary.
gms-label	Specifies the menu label for the entire menu specification.
Allowable values:	A quoted string or any GFR symbol that will be used to translate the text.
Default value:	none
Notes:	Can be used to refer to the specification as a whole. Is seen in the header of a popup menu. For information on using GFR symbols see Internationalizing GMS Menus.
gms-help-label	Specifies the help information which will appear in the Help bar at the bottom of the G2 window.
Allowable values:	A quoted string or any GFR symbol that will be used to translate the text.
Default value:	none
Notes:	Help information will appear when the menu entry is selected and GMS is configured to display Help. Controlling Help Information.

Attribute	Description
gms-activation-callback	Specifies the default callback procedure which runs when a user chooses a menu entry which does not have a value for gms-activation-callback.
Allowable values:	A symbol naming a gms-activation-callback procedure.
Default value:	none
Notes:	For more information see Specifying a Default Activation Callback.
gms-initially-enabled	Specifies whether the menu is initially enabled or disabled.
Allowable values:	true, false
Default value:	true
Notes:	Disabling the root template disables the entire menu.
gms-posting-callback	Specifies the procedure which runs when a menu is displayed or hidden. Can be used to customize a menu's appearance and to restore it to the default state.
Allowable values:	A symbol naming a gms-posting-callback procedure.
Default value:	none
Notes:	For more information see Invoking a Procedure When a Menu is Displayed or Hidden.

Additional Attribute for Popup Menus

In addition to the attributes shown above, the popup-menu-template has the following attribute:

Attribute	Description
gms-show-header	Specifies if a popup menu displays a header above the entries in its panel. Contains the label given by the gms-label attribute. By default, no header is displayed.
Allowable values:	true, false
Default value:	false

Notes on the Root Template's Properties

The Root Template's Menu Label

A GMS menu specification has a label called a **menu label**. A menu specification's label is an attribute of the specification's root template.

A root template has an attribute display that shows the template's label. On the palette, the value shown is none. When you clone a root template, you can change its label by editing the attribute display or by editing the template's gms-label attribute.

The Root Template's Key

Some situations require a menu to have an arbitrary associated value of some type. To provide for these cases, a root template has an attribute called a **key**, to which you can assign any value. When you use the GMS API, you can use this value to reference the root template.

To assign a key to the menu specification:

➔ Edit the root template's gms-user-key attribute to specify the desired key.

Attributes of a Menu Entry Template

The following attribute table describes the attributes that are common to the Cascade Menu template and the Leaf entry template, two entry templates commonly used in menus.

Using attribute settings you can restrict a menu item's use, define its appearance, set access keys, set its language, provide help information, enable or disable the menu on display, and set the action which occurs when this choice is made.

Attribute	Description
gms-restricted-modes	Lists the user modes in which this menu entry is displayed or lists modes in which this menu entry is not displayed.
Allowable values:	One or more symbols
Default value:	A symbol-array with an initial value of G2.
Notes:	To restrict the menu entry when a user is in a given mode, specify the mode as a symbol. To restrict an entry when the user *is not* in a not in a given mode, specify the mode as a symbol prefixed by a tilde (~). For information on using the tilde see .
gms-index	Unique index number for this template object.
Allowable values:	Not user settable.
Default value:	0
Notes:	You may access the gms-index but do not change its value. For more information see Referencing Menus and Menu Entries.
gms-user-key	Holds any value assigned by the user and can be used to access the menu entry.
Allowable values:	Any symbol or value
Default value:	none

Attribute	Description
gms-text-resource-group	Specifies the text resource group that GMS will use to translate this menu entry.
Allowable values:	The name of any gms-text-resource-group
Default value:	none
Notes:	For more information see Specifying the Translation Dictionary.
gms-label	Specifies the menu label for this menu entry.
Allowable values:	A quoted string or any GFR symbol that will be used to translate the text.
Default value:	none
Notes:	Can be used to refer to the menu entry.
gms-help-label	The help information which will appear in the Help bar at the bottom of the G2 window when a user selects this menu entry.
Allowable values:	A quoted string or any GFR symbol that will be used to translate the text.
Default value:	none
Notes:	Help information will appear when the menu entry is selected and GMS is configured to display Help. For more information on displaying Help text see Controlling Help Information.
gms-activation-callback	For a Cascade entry, specifies the default callback procedure which runs when a user chooses a menu entry which does not have a value for gms-activation-callback. For a leaf entry, specifies the callback procedure which runs when a user chooses the menu entry.

Attribute	Description
Allowable values:	A symbol naming a gms-activation-callback procedure.
Default value:	none
Notes:	For more information see Specifying a Default Activation Callback.
gms-inline-icon-class	Specifies the class of the menu entry icon.
Allowable values:	A symbol naming any subclass of gms-icon
Default value:	none
Notes:	The icon of the specified class will appear to the left of the menu entry label whenever the entry is visible to the user. For more information see Specifying a Menu Entry Icon.
gms-inline-icon-description	Contains an instance of the class which holds the attributes necessary for configuring an icon's appearance.
Allowable values:	class gms-icon-specification
Default value:	none
Notes:	Use this attribute to change the appearance of the entry's icon from the default. For more information seeConfiguring a GMS Icon.
gms-initially-enabled	Specifies that the menu is initially enabled.
Allowable values:	true, false
Default value:	true

Attribute	Description
gms-selection-callback	Specifies the procedure which runs when a menu item is selected. Can be used to customize a menu's appearance and to restore it to the default state.
Allowable values:	A symbol naming a gms-selection-callback procedure
Default value:	none
Notes:	For more information see Invoking a Procedure on Selection and Unselection.

Additional Attribute of a Cascade Menu Template

In addition to the attributes shown in the table of attributes for Entry templates, a Cascade Entry Menu template has the following attribute:

Attribute	Description
gms-posting-callback	Specifies the procedure which runs when a menu is displayed or hidden. Can be used to customize a menu's appearance and to restore it to the default state.
Allowable values:	A symbol naming a gms-posting-callback procedure.
Default value:	none
Notes:	For more information see Invoking a Procedure When a Menu is Displayed or Hidden.

Additional Attributes of a Leaf Entry Template

In addition to the attributes shown in the table of attributes for Entry templates, a Leaf Entry template has the following attributes:

Attribute	Description
gms-accelerator-label	Specifies an accelerator label for the entry.
Allowable values:	A quoted string or any GFR symbol that will be used to translate the text.
Default value:	none
Notes:	The accelerator label that you specify will appear to the right of the menu entry label whenever the entry is visible to the user. For more information see Specifying a Menu Entry Accelerator Label.
gms-initially-checked	Specifies if the menu entry has a check mark to its right when first displayed.
Allowable values:	true, false
Default value:	false
Notes:	For more information see Specifying Menu Entry Initial States.
gms-lock-during-callback	Specifies if the menu entry is locked against user input when a callback procedure is executing.
Allowable values:	true, false
Default value:	false
Notes:	For more information see Locking All Menus Against User Input.

Accessing a Menu Entry's Properties Programmatically

Any property which can be accessed through the attribute table can also be accessed programmatically either through ordinary G2 procedures or through the GMS API.

Accessing a Menu Entry's Label

Like root templates, every entry template has an attribute display of its gms-label attribute that shows the entry's label. On the palette, the value shown is none. When you clone an entry template, you can change its label by editing the attribute display or by accessing the entry template's table and editing the attribute there. In addition, a menu entry's label can be changed programmatically using the API.

To change the label of a menu entry:

➔ gms-set-label
(*handle*: integer, *menu-index*: integer, *label*: text)

Sets the label of the entry referenced by *menu-index* in the window referenced by *handle* to be *label*. The *label* must be a quoted string. To specify no label, give an empty string.

To obtain the label of a menu entry:

➔ gms-get-label
(*handle*: integer, *menu-index*: integer)
-> *label*: text

Returns the label of the entry referenced by *menu-index* in the window referenced by *handle*. If the entry has no label, the call returns an empty string.

Note that these functions manipulate labels as text, *not* as GFR symbols. If your menu uses GFR to provide internationalization, use GFR calls as needed to translate labels. See Internationalizing GMS Menus, for further information, and the *G2 Foundation Resources User's Guide* for complete information.

Specifying Menu Entry Initial States

When GMS displays a menu, each entry in the menu can be enabled or disabled. When GMS displays a transient menu, each leaf entry in the menu can be checked or unchecked. You can:

- Toggle these states as needed using API calls, as described under Disabling and Enabling Menu Entries and below.

- Set attributes in each menu entry template to define the enablement and checking that exists initially, prior to any change by an API call.

To specify the initial enabled/disabled state of a menu entry:

➔ Edit the entry template's gms-initially-enabled attribute to contain true (the default) if the entry is initially enabled, or false if it is initially disabled.

To specify the initial checked/unchecked state of a menu entry:

➔ Edit the entry template's gms-initially-checked attribute to contain true if the entry is initially checked, or false (the default) if it is initially unchecked.

Checking and Unchecking Menu Entries

You can programmatically place or remove a check in front of an entry in a transient menu in any window at any time using an API call. Checking a menu entry has no functional effect on the entry, it is purely a convenience for the user.

To check a menu entry:

➔ gms-check-entry
(*handle*: integer, *menu-index*: integer)

Checks the menu entry referenced by *menu-index* in the window referenced by *handle*. If the entry is already checked, the call has no effect.

To uncheck a menu entry:

➔ gms-uncheck-entry
(*handle*: integer, *menu-index*: integer)

Unchecks the menu entry referenced by *menu-index* in the window referenced by *handle*. If the entry is already unchecked, the call has no effect.

To determine if an entry is checked:

➔ gms-entry-is-checked
(*handle*: integer, *menu-index*: integer)
-> *status*: truth-value

Returns true if the menu entry referenced by *menu-index* in the window referenced by *handle* is checked, and false otherwise.

In some groups of menu entries only one entry can be checked at a given time. This type of entry is called a radio entry. It is also possible to check any one entry of a specified group of entries and at the same time uncheck any previously checked entry in the book.

To check one radio entry in a group and uncheck another:

➔ gms-check-radio-entry
(*handle*: integer, *check-index*: integer, *radio-start-index*: integer, *radio-end-index*: integer)

where:

check-index is the index of the entry to be checked.

radio-start-index is the index of the first entry in the group.

radio-end-index is the index of the last entry in the group.

check-index, *radio-start-index*, and *radio-end-index* must reference menu entries on the same panel. The menu entry specified by *check-index* will be checked and any previously checked entry between the last two will be unchecked.

Specifying the Effect of Choosing a Leaf Entry

To have an effect when chosen, a leaf entry invokes a user-defined procedure known as a callback procedure. This procedure must take three arguments that GMS passes to it when it is called, as described under Invoking a Procedure Upon Menu Selection.

To specify the callback procedure for a leaf entry:

➔ Edit the entry template's gms-activation-callback attribute to contain the name of the procedure to call.

GMS calls the specified procedure whenever the menu entry is chosen. If you do not specify a callback procedure for a leaf entry, GMS searches for a default callback procedure when the entry is chosen, as described under Specifying a Default Activation Callback.

Compiling the Menu Specification

Shows how to compile and display menus and how to reference the parts of a menu.

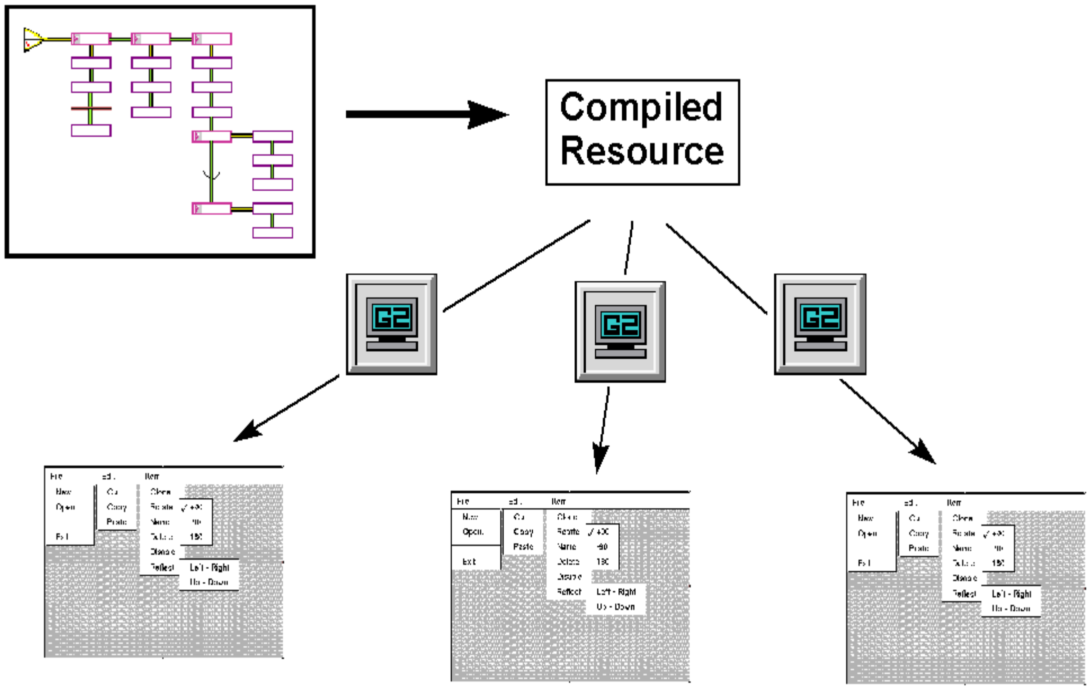
Introduction

After you have defined a menu using the techniques described in the previous chapter, you compile the menu in order to use it. This chapter describes what happens when you compile a GMS Menu. It also covers how to compile and display menus and how to reference the parts of a menu.

How GMS Compiles Menu Specifications

GMS does not display menus by reading information directly from their specifications. To improve performance, GMS compiles all menu specifications. The GMS compiler is native to GMS. It is not the same as the G2 compiler.

The following diagram summarizes what happens when you compile a menu. It is explained in the sections that follow.



The Menu Translation

The result of compiling a menu specification is called a **menu translation**. GMS keeps all menu translations in a repository called the c**ompiled resource**. At most one compiled resource exists, which holds every existing menu translation.

By default, GMS automatically:

- Deletes the compiled resource whenever G2 resets.
- Compiles all menu specifications whenever G2 starts and no compiled resource exists.

Caution The compiled resource is for internal use only by GMS. You never need to examine it, and you must never modify it in any way.

The Menu Instance

A menu translation in the compiled resource is not a displayable menu, but a representation that GMS can use to build a displayable menu. In order to display a menu in a window, GMS must build the menu itself. A menu that GMS builds from data in the compiled resource is called a **menu instance**.

Every menu instance is bound to a particular G2 window and can be displayed only in that window. Thus, a separate menu instance exists for every menu translation in every G2 window.

The Handle

When GMS builds a menu for a window, it first assigns the window an identifying integer called a **handle**. GMS uses a window's handle to manage the relationship of the window to any GMS menu(s) displayed within it.

A menu instance does not duplicate the information in the underlying menu translation in the compiled resource. The information remains in the compiled resource, and the menu instance makes it available in the context of a particular window.

By default, GMS automatically:

- Invalidates all handles and deletes all menu instances whenever G2 resets or menus are compiled.
- Assigns every G2 window a handle, and builds an instance of every defined menu for every window, whenever menus are compiled.
- Assigns a handle and builds a complete set of menus for any G2 window that is subsequently connected via Telewindows.

Compiling One or All Menus

You can ask GMS to compile or recompile any or all menu specifications at any time. When GMS compiles one or more menu specifications, it:

- Deletes any existing compiled resource.
- Invalidates any existing window handles and deletes any existing menu instances.
- Creates a new compiled resource that contains compilations of the menu specifications.

By default, GMS then assigns every G2 window a handle and builds an instance of every menu translation for every window. You can change this default using techniques described under Compiling and Building Menus.

To compile a menu specification:

➔ Choose compile tree from the menu of the menu specification's root template.

To compile all menu specifications:

➔ Choose compile all from the menu of any menu specification's root template.

GMS describes any compilation errors by signalling an error and cancelling compilation. For more information about compiling GMS menus see Compiling Menus.

Displaying Compiled Menus

When GMS displays a menu, it obtains the necessary information from the compiled resource for the menu instance, *not* the menu specification. After GMS has compiled a menu specification, you can delete the specification without affecting the display or use of the menu.

You cannot display a menu that has not been compiled. If you change a compiled menu specification but do not recompile it, GMS will continue to use the existing menu instance, and the change(s) will have no effect.

GMS displays a cascade menu automatically when its cascade entry is selected, and undisplays any transient menus automatically when the user has chosen a leaf entry. No programmer action is necessary in these cases.

The procedures described in this chapter apply to the display of all GMS menus. For simplicity, the chapter focuses on displaying static menus. Instructions for displaying dynamically defined menus appear in Dynamically Defining and Changing Menus.

GMS Handles for G2 Windows

To display a menu in a G2 window, GMS needs more information than the g2-window class provides. To provide this information without redefining the g2-window class, GMS maintains a separate data structure called a gms-handle for every G2 window.

When you program GMS, you never need to access the data in a gms-handle. Such data is only for internal use by GMS. Each G2 window is assigned a handle. When you need to refer to the window in a GMS API call, you pass the procedure the window's handle. GMS uses the handle to retrieve the appropriate gms-window.

GMS maps transparently between gms-handles and the windows they represent. Thus, you can ignore gms-handles when you program GMS and think in terms of windows only.

You can use any standard G2 technique to obtain the G2 window in which you want to display menus. GMS provides API calls for mapping between windows and their handles:

To get the GMS handle for a G2 window:

➔ gms-get-handle-for-window
(*window*: class g2-window)
-> *handle*: integer

Returns the handle associated with a G2 window or -1 if the window has no handle.

To get the G2 window for a GMS handle:

➔ gms-get-window-for-handle
(*handle*: integer)
-> *window*: class g2-window

Returns the G2 window associated with a handle. If *handle* is not a valid handle, GMS signals an error.

Referencing Menus and Menu Entries

When you operate on a menu as a whole or on an individual menu entry, you must indicate to GMS the menu or menu entry of interest by supplying an argument to an API call.

This argument cannot be the relevant template object itself, because GMS uses template objects only as inputs to the GMS compiler. Rather, you must identify the menu or entry of interest by giving a value that GMS can use to identify the menu or entry within the compiled resource.

Menu Indexes

When GMS compiles a menu specification, it assigns a positive integer called a **menu index** to every root and entry template in the specification. Every menu index is unique, since every template in an application has its own index number.

The compiler sets the gms-index attribute of every root and entry template to be that template's menu index number. You can access that attribute to obtain the index, but you must not change its value. The compiler also stores a template's menu index in the compiled resource.

Mapping between User Keys and Menu Indexes

GMS provides menu templates with an attribute called a **key**, to which you can assign any value.You can use this key to reference the menu template through the API. Since menu indexes are compiler generated, you need some way to obtain them after the compiler has generated them. The standard technique is to give each template whose index you will need a unique key. You can then use the procedures provided by the GMS API to map between menu keys and indexes.

To get the menu index for a menu key:

➔ gms-get-index-for-key
(*handle*: integer, *key*: value)
-> *menu-index*: integer

Returns the menu index of the entry referenced by *key*, or -1 if no such entry exists. If *key* is not unique, the call returns the index of some entry having that key.

If you are pulling down dynamic menus, your menu index does not reference different menu entries on different windows.

To get the menu key for a menu index:

➔ gms-get-key-for-index
(*handle*: integer, *index*: integer)
-> *key*: value

Returns the key of the entry referenced by *index*, or false if the entry does not exist or has no key.

If you are pulling down dynamic menus, your menu index does not reference different menu entries on different windows.

These calls obtain information from the compiled resource, so they work even if you deleted the menu specification after compiling it. If the specification still exists, you can use any standard G2 technique to obtain a template's key from its gms-user-key attribute.

Displaying and Undisplaying Menu Bars

Menu bars appear and disappear only in response to API calls.

To display a menu bar:

➔ gms-display-menu-bar
(*handle*: integer, *menu-index*: integer)

Displays the menu bar referenced by *menu-index* at the top of the G2 window referenced by *handle*.

To undisplay a menu bar:

➔ gms-hide-menu-bar
(*handle*: integer)

Undisplays the menu bar currently displayed in the G2 window referenced by *handle*. If no menu bar is visible, the call has no effect.

To redisplay a menu bar:

➔ gms-redisplay-menu-bar
(*handle*: integer)

Redisplays the menu bar most recently displayed in the G2 window referenced by *handle*. If no menu bar was ever displayed in the window, the call has no effect.

To obtain the menu bar of a G2 window:

➔ gms-get-menu-bar-index
(*handle*: integer)
-> *menu-index*: integer

Returns the index of the menu bar currently on display in the G2 window referenced by handle, or -1 if no such menu bar exists.

To obtain the current height of the menu bar of a G2 window:

➔ gms-get-current-menu-bar-height
(*handle*: integer)

Returns the number of pixels in a menu bar on the G2 window. If no menu bar is visible, get-current-menu-bar-height returns 0. Use the current menu bar height to determine, for example, where to place a dialog just below the current menu bar without obscuring the menu bar.

5

Using Specialized Templates

Describes how to use the two predefined templates provided by GMS to change the user mode and to display a named workspace.

Introduction

GMS contains two predefined leaf entry templates for tasks that users frequently need to perform. One template is for changing the user mode, and the other is for displaying a named workspace. The code for the activation callbacks is already included in the templates so it is not necessary to write callback procedures when using these templates.

GMS also provides a template for displaying one of the built-in G2 menus in Telewindows on Windows platforms.

To see online examples of the concepts covered in this chapter:

1 Load the `gmsdemo.kb` from the `kbs` directory.

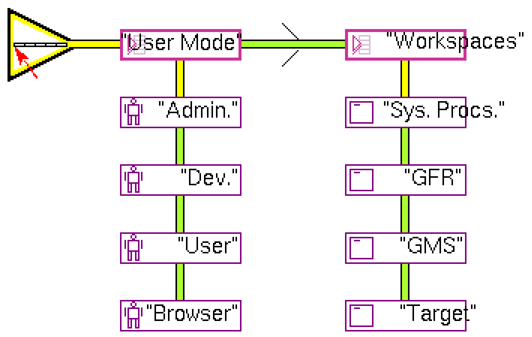
2 Choose the demo called Using Specialized Templates.

Predefined Leaf Templates

The two predefined templates provided by GMS are:

- **Change User Mode template**: Changes the user mode to a specified mode.
- **Show Workspace template**: Displays a workspace at a specified location.

The figure below shows a menu specification containing both Change User Mode Templates and Show Workspace Templates.



The Change User Mode template and the Show Workspace template are subclasses of the Leaf Entry template. Each has:

- Code for carrying out the operation characteristic of the template.
- Class-specific attributes that specify exactly what the operation should do.

Since the predefined leaf templates do not use callback procedures to perform their operations, you can customize them by defining a callback procedure, as with any Leaf Entry template. When you specify a callback procedure for a predefined leaf template, the template first performs its characteristic operation, then it invokes the callback procedure.

The GMS compiler includes the attributes of the two predefined templates in the compiled resource. Therefore, you can delete a predefined leaf template after you have compiled the menu specification that contains it.

Caution Do not create subclasses of GMS template objects that use class-specific attributes to hold information for use by callback procedures. For further information, see Providing Additional Data to an Activation Callback.

GMS also includes a predefined template called a **Switch Menu Bar template** for switching among currently available menu bars, as described under Switching Menu Bars.

Changing the User Mode

A Change User Mode template behaves just like a leaf entry template except it generates a leaf entry that changes the user mode to a mode specified in the template. When a Change User Mode entry exists in a menu and the user mode that it specifies is the current user mode, GMS automatically checks the menu entry.

none

Change User Mode Template Attributes

In addition to the attributes belonging to all Leaf Entry templates, as described in Attributes of a Menu Entry Template and Additional Attributes of a Leaf Entry Template, a Change User Mode template has the following attribute.

Attribute	Description
gms-target-user-mode	A symbol naming the mode to switch to.
Allowable values:	Any symbol representing a user mode
Default value:	administrator

To use a Change User Mode template in a menu specification:

1. Include the Change User Mode template in a menu specification exactly as you would any Leaf Entry template.
2. Edit the template's label attribute to specify the desired user mode.
3. Edit the template's gms-target-user-mode attribute to specify the desired user mode.

If the Change User Mode template specifies a callback procedure, choosing the entry first changes the user mode then invokes the procedure. If the template does not define a callback procedure, GMS does *not* search for a default callback procedure, as it would for an ordinary Leaf Entry template: it continues executing without notification or error.

Displaying a Workspace

A Show Workspace template is Leaf Entry template that:

- Defines additional attributes that specify the identity and desired appearance of a workspace that is to be displayed.
- Uses a predefined callback procedure to display the workspace.

A Show Workspace template is actually an interface to the G2 show workspace action. That action provides a variety of techniques for displaying a workspace at a given location and scale in a G2 window. You can use a Show Workspace template to achieve most of the effects of the show workspace action.

To use a Show Workspace template in a menu specification

1. Include the template in the specification exactly as you would a Leaf Entry template.
2. Edit the template's gms-display-target to specify the name of the workspace to display.
3. Edit the various display coordinates to designate the display scale and where the workspace should appear in the window.

If the Show Workspace template specifies a callback procedure, choosing the entry first displays the workspace, then invokes the procedure. If the template does not define a callback procedure, GMS does *not* search for a default callback procedure, as it would for an ordinary Leaf Entry template; instead, it continues executing without notification or error.

Show Workspace Template Attributes

In addition to the attributes belonging to all Leaf Entry templates, as described in Attributes of a Menu Entry Template and Additional Attributes of a Leaf Entry Template, a Show Workspace template has the following attributes.

Attribute	Description
gms-display-target	Specifies the workspace to be displayed.
Allowable values:	A symbol that names a workspace, the superior item of a subworkspace, or an item that exists on a workspace.
Default value:	none
gms-view-scale	Specifies the scale at which to display the workspace.
Allowable values:	A float value indicating a scale factor, or none
Default value:	none
gms-window-symbolic-location	A symbol that specifies a point in the window.
Allowable values:	top-left-corner, top-center, top-right-corner, right-center, bottom-right-corner, bottom center, bottom-left-corner, left-center, center
Default value:	none
gms-window-x-location **gms-window-y-location**	A coordinate pair, implemented as two integer attributes, that specifies a point in the window.
Allowable values:	Any two integers
Default values:	none

Attribute	Description
gms-window-x-offset **gms-window-y-offset**	A coordinate pair, implemented as two integer attributes, that specifies an offset from a window location.
Allowable values:	Any two integers
Default values:	none
gms-workspace-symbolic-location	A symbol that specifies a point on the workspace.
Allowable values:	top-left-corner, top-center, top-right-corner, right-center, bottom-right-corner, bottom center, bottom-left-corner, left-center, center
Default value:	none
gms-workspace-x-location **gms-workspace-y-location**	A coordinate pair, implemented as two integer attributes, that specifies a point on the workspace.
Allowable values:	Any two integers
Default values:	none
gms-workspace-x-offset **gms-workspace-y-offset**	A coordinate pair, implemented as two integer attributes, that specifies an offset from a workspace location.
Allowable values:	Any two integers
Default values:	none

Attribute	Description
gms-allow-for-menu-bar	A truth-value specifying whether to consider the presence of the menu bar displayed on the window. If set to true, all window location specifications relative to the top of the window (top-left, top-center, or top-right) are considered relative to the bottom of the menu bar.
Allowable values:	true or false
Default value:	false

The rest of this section shows you how to use these attributes to tell GMS what workspace to display and how to display it.

Specifying the Workspace to Display

To specify the workspace to display:

➔ Set gms-display-target to a symbol that names one of the following:

- A workspace: GMS displays the workspace.
- The superior item of a subworkspace: GMS displays the subworkspace.
- An item that exists on a workspace: GMS displays the workspace that contains the item.

To select the workspace to display, GMS first searches all named workspaces, then all items that have subworkspaces, then all items on workspaces, and displays the first matching workspace that it encounters. If GMS cannot find a matching workspace, it signals an error.

Specifying the Display Scale

To specify the scale at which to display the workspace:

➔ Set gms-view-scale to a float that gives the scale, or to none.

If you specify a float, G2 displays the workspace scaled by that value, up to a maximum of 4.0 times its normal size. Specifying 1.0 displays the workspace at normal size. Specifying none displays the workspace at the scale at which it was most recently displayed, or at normal size if the workspace has never been displayed.

Specifying a gms-view-scale value has the same effect as specifying the *scale* in a show *workspace* scaled by *scale* action.

Specifying the Workspace Location

Except for gms-display-target and gms-view-scale, all Show Workspace template attributes exist to tell G2 where to display the workspace. This section refers to such attributes as **location attributes**.

Despite the multiplicity of location attributes, the information G2 needs is actually simple: one point on the G2 window, and one point on the workspace. G2 displays the workspace so that the two specified points appear at the same location on the screen. This section refers to the two points as **display points**.

You can specify display points in two different ways:

- Symbolically, by giving a symbol that specifies a point on the edge or at the center of the window or workspace.
- Numerically, by giving a pair of integers that specify any point in the window or workspace.

The two specification techniques work in the same way for windows and for workspaces. You can specify one display point symbolically and the other numerically, or you can use the same technique for both.

Conflicting Display Point Specifications

The possibility of specifying a display point symbolically or numerically permits you to give conflicting specifications. To prevent ambiguity, a symbolic display point specification takes precedence over a numeric specification.

Missing Display Point Specifications

If you specify a window display point symbolically, and do not specify a workspace display point, the workspace display point defaults to the window display point.

In all other cases, if you fail to supply both a window display point and a workspace display point, the workspace appears where it did the last time it was displayed. If the workspace was never displayed, G2 provides default window and workspace display points.

Specifying a Display Point Symbolically

You can specify a window or workspace display point symbolically by giving one of the values top-left-corner, top-center, top-right-corner, right-center, bottom-right-corner, bottom center, bottom-left-corner, left-center, or center. Each of these describes a point along the edge, or at the center, of the window or workspace.

To specify a window display point symbolically:

➔ Set gms-window-symbolic-location to be one of the values: top-left-corner, top-center, top-right-corner, right-center, bottom-right-corner, bottom center, bottom-left-corner, left-center, or center.

To specify a workspace display point symbolically:

➔ Set gms-workspace-symbolic-location to be one of the values: top-left-corner, top-center, top-right-corner, right-center, bottom-right-corner, bottom center, bottom-left-corner, or left-center.

If you specify either display point as none (the default), GMS looks for a numeric specification of that display point.

Note To keep the current menu bar visible while displaying other workspaces relative to the top of the window, use a template's gms-allow-for-menu-bar attribute.

Specifying a Display Point Numerically

You can specify a window or workspace display point numerically by giving a pair of integers that designate the X and Y coordinates of the point. GMS looks for a numeric specification of a display point only when the symbolic specification for that point is none.

To specify a window display point numerically:

➔ Set gms-window-x-location and gms-window-y-location to be integers that specify the X and Y coordinates of the point.

To specify a workspace display point numerically:

➔ Set gms-workspace-x-location and gms-workspace-y-location to be integers that specify the X and Y coordinates of the point.

If you specify either coordinate as none, GMS treats both positions as being none, and obtains a default display point from G2.

Specifying a Location Offset

You can modify a display point specified symbolically or numerically by specifying a numeric offset. The offset consists of a pair of integers. One specifies an X offset, the other a Y offset.

When you specify an offset to a symbolic display point, GMS first converts the location to equivalent X and Y coordinates. When you specify an offset to a numeric display point, GMS already has the X and Y coordinates.

To implement an offset, GMS adds the X and/or Y offsets to the X and/or Y coordinates of the original display point. GMS uses the result as the display point.

To specify a window display point offset:

➔ Set gms-window-x-offset and gms-window-y-offset to be integers that specify the X and Y offsets of the point.

To specify a workspace display point offset:

➔ Set gms-workspace-x-offset and gms-workspace-y-offset to be integers that specify the X and Y offsets of the point.

You specify either or both offsets as none. The effect is the same as if you had specified the offset as 0.

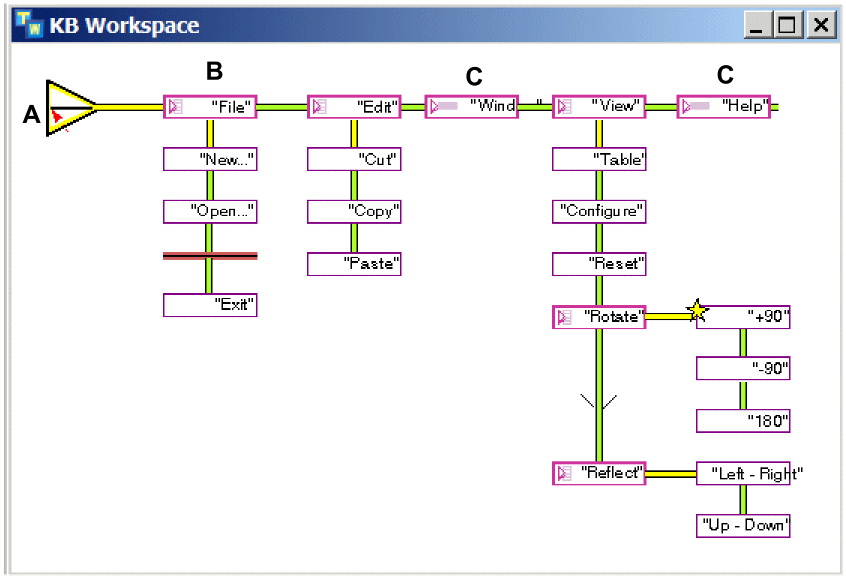
Creating Built-in G2 Menus

You can use a gms-builtin-template to create one of the built-in G2 menus. The built-in menu only works in Telewindows Next Generation (*twng.exe*); the menu choice is grayed out in classic G2 and Telewindows (*tw.exe*).

To specify which built-in menu to create, configure the gms-label attribute to be one of these text values: "file", "edit", "view", "toolbars", "run", "tools", "window", "run-options", "package-preparation", "system-tables", or "help".

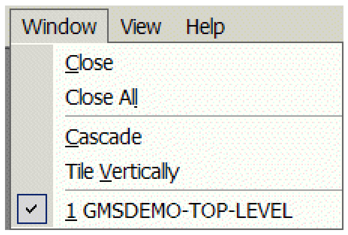
To localize the built-in menu text, configure the gms-builtin-designator attribute to be one of these symbols: file, edit, view, toolbars, run, tools, window, run-options, package-preparation, system-tables, or help. When gms-builtin-designator is specified as a symbol, specify the gms-label as a symbol, which is a GFR key for localizing the built-in menu text.

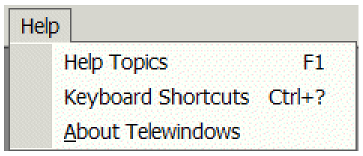
Here is a menu specification that includes two built-in G2 menu templates, the Window menu and the Help menu:



A: Menu Bar Template
B: Cascade Template
C: Built-in Template

Here are the two built-in G2 menus in Telewindows:





Defining Popup Menus

Describes how to create and display popup menus.

Introduction

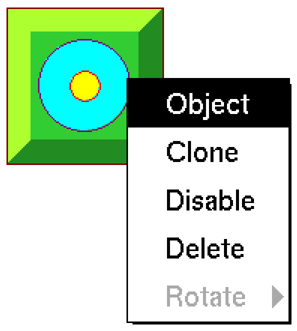
A popup menu is a freestanding menu that can appear anywhere in a G2 window in response to a mouse click. A popup menu is associated with an item in a G2 window. Any G2 item can be configured to display a GMS popup menu instead of its G2 menu.

A popup menu typically appears when the user clicks the mouse on an item and disappears as soon as the user selects a leaf entry from the menu or from a cascade menu that it contains. The item on which the user clicks is called the **initiating item**.

A popup menu can have a title block called a **header** that describes the purpose of the menu.

A popup menu can contain cascading entries like any other menu. You can display popup menus automatically or through the API. To display popup menus automatically, you use gms-popup-subscriber as a mixin class for the initiating object. GMS then handles all of the mouse gestures automatically. Otherwise, the program developer can handle the mouse gestures and use the API to post the menu at the appropriate location.

The figure below shows a popup menu:



To see online examples of the concepts covered in this chapter:

1 Load the *gmsdemo.kb* from the *kbs* directory.

2 Press the navigation button next to the label POPUP MENUS.

   This displays a list of the demos pertaining to popup menus. There are two demos in this section that illustrate the concepts covered in this chapter.

3 Choose the following demos:

   - Simple Popup Menus
   - Posting Menus for Items

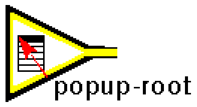
Defining a Popup Menu

You define a GMS popup menu in a manner similar to the way you define a GMS Menu Bar:

- Create a GMS menu specification for the popup menu.
- Set the attributes of the root template to specify if the popup menu will display a header.

Creating a Menu Specification for a Popup Menu

You create the menu specification for a popup menu the same way you create any menu specification except that you use the Popup Menu template for the root specification. The Popup Menu template in the next figure has been given the gms-label popup-root.



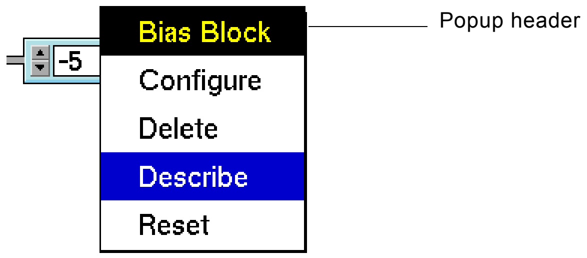
To create a GMS menu specification for a popup menu:

1. Clone a Popup Menu template and place it on a workspace.
2. Clone Cascade Menu templates and Leaf Entry templates as needed and attach them to the Root template and to each other, using the appropriate connections.
3. Give all of the Leaf Entry templates labels by editing the attribute display of gms-label on each template.

   The value of the gms-label attribute should be either a string, which appears on the menu, or a symbol which GFR will uses to translate the label.
4. On the table for the root template, specify the attribute gms-user-key as a symbol.

Specifying a Popup Menu Header

You can set a popup menu to display a header above the entries in its panel. The header contains the label of the menu, which is given by the gms-label attribute of its root template. The popup in the following figure has a header:



To specify the header of a popup menu:

➔ Edit the root template's gms-label attribute to specify the desired text as either:

- A quoted string that gives the text literally.
- A GFR symbol.

Specifying this text does not cause the popup menu to have a header. It only defines the text of the header.

To specify that a popup menu displays a header:

➔ Edit the root template's gms-show-header attribute to have the value true.

To specify that a popup menu does not display a header:

➔ Edit the root template's gms-show-header attribute to have the value false (the default).

Displaying Popup Menus

A GMS popup menu typically appears because the user clicked the mouse on an item. This item is called the **initiating item**. The popup menu disappears automatically when the user selects a leaf entry from the menu or a subsidiary cascade menu. If the user of the popup menu decides to take none of the menu choices, clicking on the header dismisses the menu.

G2 includes a predefined popup menu capability that operates below the level of GMS. When you use GMS, you can configure objects to display GMS popups rather than G2 popups. GMS offers three ways to associate a GMS popup menu with an item:

- Include the mixin gms-popup-subscriber in the item's class definition, as described in Using gms-popup-subscriber to Display a Popup Menu.
- Use G2 item configurations to invoke either the procedure gms-display-popup-menu or gms-display-popup-menu-at-last-event-location when the user clicks on the item, as described in Using G2 Configurations to Display a Popup Menu.
- Configure GMS to handle mouse events, as described under Handling Mouse Events for Popup Menus, This technique provides customized mouse-event handling, but requires some programming.

Using gms-popup-subscriber to Display a Popup Menu

GMS provides a mixin class called gms-popup-subscriber that you can include in a class definition. Clicking an instance of a class that inherits gms-popup-subscriber causes GMS to display a popup menu.

To use gms-popup-subscriber to associate a GMS popup menu with a class:

1. On the table for the class definition of the initiating object, give the attribute direct-superior-classes the additional class gms-popup-subscriber.

   This mixin adds the attribute gms-popup-pointer to the class definition.

2. Edit the attribute-initializations of the class to read:

   gms-popup-pointer initially is *popup-key*

   where *popup-key* is the key of the root template of a popup menu.

This makes an association between the gms-user-key on the root template of the popup menu and the class of the object. Whenever the user clicks the mouse on an instance of the class, GMS displays the popup menu whose root template specifies *popup-key* as its gms-user-key. If no such menu exists, GMS signals an error.

When the user clicks a leaf entry selected through a GMS popup displayed via gms-popup-subscriber, GMS automatically includes the initiating item in the arguments to the leaf entry's callback procedure.

Accessing G2 Tables and Modifying G2 Attributes

Since the gms-popup-subscriber mixin substitutes GMS menus for G2 menus, you cannot directly edit the table of an item that inherits gms-popup-subscriber. However, the table and its attributes remain accessible in other ways.

To edit the table of an item that inherits gms-popup-subscriber:

➔ Use Inspect to obtain the instance and display its table.

To modify programmatically an item that inherits gms-popup-subscriber:

➔ Use the conclude and change actions, as with any G2 item.

To provide G2 menu capabilities in a GMS menu:

➔ Define GMS menu entries and callback procedures as needed to provide the same effect as the G2 menu entries.

Using G2 Configurations to Display a Popup Menu

GMS provides two API calls that display a popup menu. You can use either of these calls in conjunction with G2 configurations to display a popup menu. The calls are:

gms-display-popup-menu
(*handle*: integer, *menu-index*: integer, *x*: integer, *y*: integer, *initiating-item*: item-or-value)

Displays the popup menu referenced by *menu-index* in the G2 window referenced by *handle* at window coordinates (*x*, *y*). GMS passes *initiating-item* to any callback procedure that is invoked via the popup menu.

You can set *initiating-item* to be the item that the user clicked to display the popup. If you don't need to pass this item to a callback procedure, specify a dummy value of false. Do not specify any other dummy value, or GMS signals an error.

gms-display-popup-menu-at-last-event-location
(*handle*: integer, *menu-index*: integer, *initiating-item*: item-or-value)

Displays the popup menu referenced by *menu-index* in the G2 window referenced by *handle* at the coordinates of the last event in the window. The gms-display-popup-menu-at-last-event-location determines the last event like the system procedures g2-last-input-event and g2-last-input-event-info. The gms-display-popup-menu-at-last-event-location procedure uses g2-last-input-event-info to determine the coordinates of the location for posting the menu.

GMS passes *initiating-item* to any callback procedure that is invoked via the popup menu.

You can set *initiating-item* to be the item that the user clicked to display the popup. If you do not need to pass this item to a callback procedure, specify a dummy value of false. Do not specify any other dummy value, or GMS signals an error.

Typically, you would use this procedure to post a menu in response to a mouse click by the user. To post a menu when the user selects an item, the item must have a configuration clause that specifies a procedure that calls gms-display-popup-menu-at-last-event-location, as in the following example:

```
selecting any item implies post-menu
```

where post-menu is a user-menu-choice that does the following:

```
start my-posting-procedure (the item, this window);

my-posting-procedure (InitiatingItm: class item, Win: class g2-window)
GMSHandle: integer;
PopupMenuIndex: integer;
begin
    GMSHandle = call gms-get-handle-for-window (Win);
    PopupMenuIndex = call gms-get-index-for-key
        (the symbol POPUP-MENU);
    call gms-display-popup-menu-at-last-event-location (GMSHandle,
    PopupMenuIndex, InitiatingItm);
end
```

To use G2 configurations to associate a GMS popup menu with a class:

1 Create a User Menu Choice that calls a procedure that calls either gms-display-popup-menu or gms-display-popup-menu-at-last-event-location and passes it appropriate arguments, as in the previous example.

2 Configure the class that is to use GMS popup menus to execute the User Menu Choice in response to a mouse-up event. Use either of the configuration clauses:

- releasing any mouse button on...
- selecting any item...

Caution Call gms-display-popup-menu and gms-display-popup-menu-at-last-event-location *only* in response to a mouse-up event, or while the mouse button is already raised. If you call gms-display-popup-menu or gms-display-popup-menu-at-last-event-location in response to a mouse-down event, or while the mouse button is already depressed, GMS will be unable to detect the mouse state and will be corrupted. To recover from this error, reset G2 or call gms-reset, as described under Resetting GMS without Resetting G2.

To display a popup menu in response to a mouse-down event, or while the mouse button is already depressed, use gms-manage-popup-menu as described under Handling Mouse Events for Popup Menus.

7

Writing and Using Callback Procedures

Shows how to write GMS callback procedures and supply them with the information that they need.

Introduction

The purpose of any menu system is to execute procedures when the user chooses leaf entries from menus. All other menu system features exist to make the choice as easy and as useful as possible under changing conditions and in different environments.

This chapter shows you how to write GMS callback procedures and supply them with the information that they need. For simplicity, the chapter focuses on callback procedures invoked from static menus. Dynamically Defining and Changing Menus, describes some additional considerations that apply with dynamically defined menus.

To see online examples of the concepts covered in this chapter:

1. Load the `gmsdemo.kb` from the `kbs` subdirectory in the `utils` directory under the *g2* directory.
2. Choose the demo called Callbacks.

Types of GMS Callback Procedures

- **Activation callback procedure:** A G2 procedure that is called when the user selects a GMS menu leaf entry. The activation callback procedure associated with a particular menu entry is specified by the gms-activation-callback attribute of the entry's template object, as described under Specifying the Effect of Choosing a Leaf Entry. The syntax of an activation callback procedure is described in Invoking a Procedure Upon Menu Selection.
- **Posting callback procedure:** Allows you to dynamically configure any statically defined menu bar or transient menu. GMS invokes this procedure immediately before it displays the menu, and again immediately after it hides the menu. The posting callback procedure associated with a particular menu is specified by the gms-posting-callback attribute of the root template or cascading menu template as described under Invoking a Procedure When a Menu is Displayed or Hidden.
- **Selection callback procedure**: Allows you to take an action when a statically defined menu entry is selected or unselected. A selection callback procedure is similar to a posting callback procedure, except that it is called on selection and unselection rather than on display and undisplay. The selection callback procedure associated with a particular menu entry is specified by the gms-selection-callback attribute of the entry's template object, as described in Invoking a Procedure on Selection and Unselection.

Invoking a Procedure Upon Menu Selection

Syntactically, a callback procedure is an ordinary G2 procedure. Its only special property is that it accepts specific arguments supplied by GMS. The activation callback invokes a procedure when the user selects a GMS menu leaf entry. The synax for this callback is:

procedure-name
    (*handle*: integer, *activation-info*: item-or-value, *menu-index*: integer)

The *procedure-name* can be any unique name. The arguments are:

Argument	Description
handle	The handle of the window that contains the menu from which the call originates
activation-info	An object containing detailed information about the context of the call
menu-index	The menu index of the leaf entry from which the call originates

Every activation callback must take exactly these three arguments; you cannot omit any of them or define any in addition to them. This chapter refers to the three arguments that GMS passes to a callback procedure as **callback arguments**.

A callback procedure is free to ignore any or all callback arguments, and to make any desired use of them. Typically, a callback procedure uses its arguments to determine what action to take.

Caution When an activation callback procedure returns, GMS automatically deletes the *activation-info*. Do not try to use the *activation-info* outside the context of the callback procedure invocation that received it.

Using Activation Callback Arguments

The three activation callback arguments specify the complete context within which a callback procedure was invoked. This context is the callback procedure's **activation context**. That context consists of:

- The G2 window containing the menu from which the call originated.
- For a callback procedure accessed through a popup menu, the value of the initiating item that the user clicked to display the popup.
- The top-level menu entry that the user selected.
- Any cascading entries that the user selected while navigating to a leaf entry.
- The leaf entry that the user chose in order to invoke the callback procedure.

This section shows you how to use GMS API calls to obtain activation context information from callback arguments. You cannot retrieve such information directly from the arguments, because GMS stores it in coded form. Use API calls exclusively.

Once you have the needed data from callback arguments, you can use the data as appropriate in the callback procedure.

Obtaining the G2 Window

The *handle* argument to a callback procedure represents the G2 window containing the menu through which the callback procedure was invoked.

To get the G2 window for a handle:

➔ gms-get-window-for-handle
(*handle*: integer)
-> *window*: class g2-window

Returns the G2 window associated with a handle. If *handle* is not a valid handle, GMS signals an error.

You can now use standard G2 techniques to retrieve information from window, and use this information as needed in the callback procedure.

Obtaining the Initiating Item for a Popup Menu

When you use gms-display-popup-menu or gms-manage-popup-menu to display a popup menu, the last argument to the call, *initiating-item*, can be the item that the user clicked to display the popup, or a dummy value of false, as described under Using G2 Configurations to Display a Popup Menu.

When an item displays a popup menu automatically via the gms-popup-subscriber mixin (Displaying Popup Menus), the GMS mechanism that displays the popup automatically supplies the item that the user clicked, just as if it had been provided explicitly as an *initiating-item.*

In either case, when a callback procedure is invoked through a popup menu or a subsidiary cascade menu, the initiating item exists in the *activation-info* argument to the callback procedure.

To obtain the initiating item from an activation path:

➔ gms-get-item-initiating-popup
(*activation-info*: item-or-value)
-> *initiating-item*: class item-or-value

Returns the item that the user clicked to display the popup menu, or the dummy value false if the popup was displayed by a call to gms-display-popup-menu or gms-manage-popup-menu that specified false as the *initiating-item*.

You can now use standard G2 techniques to retrieve information from initiating-item, and use this information as needed in the callback procedure.

Obtaining Menu Entries and Attribute Values

The *activation-info* argument to a callback procedure specifies the complete path the user navigated from a top-level menu to a leaf entry. This path consists of:

- The top-level menu entry.
- Every cascading menu entry that the user selected (if any).
- The leaf entry that the user chose.

An activation path does not contain templates, because the templates might not exist when the callback procedure is invoked. Instead it contains a sequence of menu indexes. You can use these indexes to retrieve information about all menu entries on the path. GMS obtains this information from the compiled resource.

Menu Levels

The menu indexes in an activation path are ordered by menu level. The value of **menu level** is an integer that indicates how deeply a menu is nested within a higher level menu. A top-level menu, whether a menu bar or a popup, has a menu level of 0; a cascade menu derived from a top level menu has a menu level of 1; a cascade menu derived from a level 1 menu has a level of 2; and so on.

Thus the level 0 menu index in an activation path is the index of the top-level menu; the level 1 menu index is the index of the first cascade menu (if any); and so on. If the relevant template objects exist, these indexes are the values assigned by the compiler to their gms-index attributes, as described under Menu Indexes.

The last index in an activation path is the index of the leaf entry that invoked the callback procedure. For convenience, this index also appears separately as the *menu-index* argument to the callback procedure.

Obtaining Menu Indexes from an Activation Path

GMS provides the following two *functions* for using menu levels to retrieve menu indexes from an activation path:

To obtain the level of the leaf entry in an activation path:

➔ gms-get-activation-level
(*activation-info*: item-or-value)
-> *level*: integer

Returns the menu level of the entry that invoked the callback procedure.

To obtain the menu index at a specified level of an activation path:

➔ gms-get-activation-index
(*activation-info*: item-or-value, *menu-level*: integer)
-> *index*: integer

Returns the menu index of the menu entry at level *menu-level*, or -1 if no such entry exists (because *menu-level* was out of range).

Note Gms-get-activation-level and gms-get-activation-index are functions, not procedures.

Obtaining Menu Entry Attributes in an Activation Context

Once you have the index of a menu entry, you can use it to obtain the value of any attribute of the entry. These values were originally specified in the template object that defined the entry. GMS retrieves them from the compiled resource.

To obtain the value of a menu entry attribute from the compiled resource:

➔ gms-get-activation-property
(*activation-info*: item-or-value, *menu-index*: integer,
property-name: symbol)
-> *property-value*: item-or-value

Returns the value of *property-name* in the menu or menu entry specified by *menu-index* in the *activation-info*.

Obtaining All Cascade Menu Entries

Once you have obtained the index of a cascading menu entry from an activation path, you can obtain the indexes of all entries in the cascade menu.

To obtain a list of all cascade menu entry indexes:

➔ gms-return-submenu-entries
(*handle*: integer, *menu-index*: integer, *index-list*: class integer-list)

Given the index of a cascade menu entry, obtains the indexes of all menu entries in the cascade menu.

Argument	Description
handle	The handle that is the first argument to the callback procedure.
menu-index	The index of a cascading menu entry that is referenced in the activation path.
index-list	An integer-list to which GMS appends the indexes of all entries in the cascade menu.

Note that gms-return-submenu-entries provides a list of menu indexes by modifying an existing list, rather than creating and returning a new one. In this way the application programmer is in control of the list and it is in no danger of being deleted by GMS.

Caution Be sure to explicitly delete every *index-list* after you no longer need it. If you fail to do this, the accumulation of undeleted lists will eventually consume all available storage, and G2 will cease to function.

You can also call gms-return-submenu-entries outside of an activation context, as described under Accessing the Compiled Resource.

Providing Additional Data to an Activation Callback

Many situations require an activation callback procedure to have more data available than the attributes of GMS template objects provide. Two approaches to supplying data to an activation callback procedure are:

- Store the information in the keys of template objects.
- Use template object keys to access information in non-template objects.

You could create a subclass of Leaf Entry template, give it class-specific attributes as needed, and give the attributes values that supply the information that a callback procedure requires.

Caution This technique is *not* guaranteed to work with future releases of GMS, and therefore *should not be used*.

The rest of this section describes supported techniques for supplying data to an activation callback procedure.

Storing Information in Template Keys

The GMS compiler includes predefined Leaf Entry template attributes in the compiled resource, so any information you store in them is directly available to an activation callback procedure.

If the data to be provided is a single value that is always the same, you can make that value the key of each menu entry to which it applies. This technique can be used to make one callback procedure serve many related leaf entries.

For example, suppose a menu offers a choice of colors by example: each leaf entry contains an unlabeled icon that illustrates the color that the entry represents.

The key of each such entry could be a symbol that describes the entry's color in a way that is useful programmatically. All of the entries could use the same callback procedure, which would retrieve the key from the entry and use it to set the desired color programmatically.

To retrieve a key:

➔ Use gms-get-activation-property specifying the symbol gms-user-key as the *property-name*.

Using Template Keys to Access Information

If the data to be provided to an activation callback procedure consists of more than one value, or is not always the same, the simplest technique is usually:

1 Store the data in some way that allows it to be retrieved via some identifying value.

2 Make that identifying value the key of the template that invokes the callback procedure.

The callback procedure first retrieves the key, using gms-get-activation-property, then uses standard G2 techniques to obtain the data that it identifies. Because a key can be of any type, this technique is very flexible. For example, it could be a symbol that names an object, or an integer that designates an array element. The following example is of a callback procedure that retrieves the class name of an object to create from the gms-user-key.

```
gmsd-callback-create-item-and-attach-to-mouse
    (GMSUserHandle: integer, ActivationInfo: item-or-value, ActivationIndex: integer)
DesiredClassName: symbol;
Itm: class item;
Win: class g2-window;
Label: text;
begin
    Label = call gms-get-activation-property (ActivationInfo, ActivationIndex,
        the symbol GMS-LABEL);
    inform the operator that "You chose [Label]";
    DesiredClassName = call gms-get-activation-property (ActivationInfo,
        ActivationIndex, the symbol GMS-USER-KEY);
    Win = call gms-get-window-for-handle (GMSUserHandle);
    create an instance Itm of the class named by DesiredClassName;
    transfer Itm to the mouse of Win;
end
```

Other Strategies for Providing Data to a Callback

Nothing requires you to use a key value to provide data to a callback procedure. You can use any template object attribute in any way that works.

For example, if a menu offered a choice of colors by naming them rather than by showing colored icons, a callback procedure could determine what color to set by looking at the label of the leaf entry that called it. The label would then represent the color to both the user and the application.

You can also combine the above strategies. This technique can provide a callback procedure with convenient access to sets of data that have different volatilities.

Distributing Data over Multiple Menu Templates

The preceding examples refer only to leaf entry keys, but the *activation-info* argument to a callback procedure contains the indexes of all entries that the user selected on the way to the leaf entry. You can use the techniques described in this section to retrieve the keys (or other attributes) of any of these entries, and use them as needed.

You can use this capability to factor data needed by the callback procedure(s) of multiple leaf entries to a cascade template that is superior to all of the entries. If the data does not apply to every part of the subtree, you can specify lower-level data that shadows the higher-level data as needed.

Specifying a Default Activation Callback

You can define a default activation callback procedure for any menu specification and for any subtree within a menu specification. This procedure is the value of the gms-activation-callback attribute of the relevant root template or cascading entry template.

If the user chooses a leaf entry that does not specify an activation callback procedure, GMS searches up the menu hierarchy for a default procedure.

- If the leaf entry is in a cascade menu, the search begins with the cascading entry that displayed the menu, and proceeds through any higher-level cascading entries.
- If no cascading entry defines a callback procedure, or the leaf entry is in a top-level menu, GMS looks for a callback procedure defined in the root template.

If GMS encounters a callback procedure definition during this search, it invokes the procedure. If GMS does not find any callback procedure, it signals an error.

This search pattern allows you to define a callback procedure for many leaf entries by specifying it as the callback procedure of a template that is superior to them all. If the procedure does not apply to every leaf entry in the subtree, you can specify callback procedures at lower entries that shadow the higher-level procedure as needed.

To specify a default activation callback procedure for a menu:

➔ Edit the gms-activation-callback attribute of the parent template of the menu to contain the name of the procedure to call.

Invoking a Procedure When a Menu is Displayed or Hidden

A posting callback procedure can use GMS API calls to modify the appearance of a menu in any way, and to take any other action that might be appropriate. When the procedure is invoked before a menu is displayed, it can customize the appearance of the menu to reflect conditions at the time of the call. When it is invoked after a menu is hidden, it can restore the menu to a standard state. A posting callback procedure cannot add or delete menu entries: it can only change the appearance of entries already defined.

Syntactically a posting callback is an ordinary G2 procedure. It receives the same three arguments as an ordinary callback procedure, as described under Invoking a Procedure Upon Menu Selection, plus an additional argument that tells whether the procedure has been called before the display or after the undisplay of the menu. The signature of a posting callback procedure is:

procedure-name
(*handle*: integer, *activation-info*: item-or-value, *menu-index*: integer, *display-status*: truth-value)

The *procedure-name* can be any unique name. The arguments are:

Argument	Description
handle	The handle of the window that will or did display the menu to which the call applies
activation-info	An object containing detailed information about the context of the call
menu-index	The index of the top-level menu or cascading entry from which the call originates
display-status	Tells whether the procedure has been called before menu display (true) or after menu undisplay (false)

A posting callback procedure can use the same techniques, and is subject to the same warnings, as an ordinary callback procedure. If you call gms-get-activation-level on the activation-info, the result is the level of the index of the menu or cascading entry from which the call originates.

Caution To avoid unexpected errors from the menu system, write your posting callbacks so that they do not enter wait states. Actions that will cause a procedure to enter a wait state are allow other processing, collect data, wait, and do in parallel. Avoid the use of these actions and any procedure which uses them. If there is no alternative, use gms-lock-menus and gms-unlock-menus to lock the menus while the posting callback is running.

To specify a posting callback procedure for a menu:

1 Edit the root template or cascading entry template's gms-posting-callback attribute to specify a symbol that is the name of the procedure.

2 Edit the gms-additional-posting-callback of any extensible stub (a Place Holder or a Peer Menu connection post) to specify a symbol that names the procedure.

Because a posting callback procedure is not required for a menu to function correctly, if you do not define a posting callback procedure, GMS does not search for a default procedure, and continues executing without notification or error.

Invoking a Procedure on Selection and Unselection

You can take any useful action when a statically defined menu entry is selected or unselected by specifying a selection callback procedure for the entry. A selection callback procedure is similar to a posting callback procedure, except that it is called on selection (selection-status is true) and unselection (selection-status is false) rather than on display and undisplay.

Syntactically a selection callback is identical to a posting callback procedure. Its last argument that tells whether the procedure has been called before the selection or after the unselection of the menu. The signature of a selection callback procedure is:

```
procedure-name
    (handle: integer, activation-info: item-or-value, menu-index: integer,
     selection-status: truth-value)
```

The *procedure-name* can be any unique name. The arguments are:

Argument	Description
handle	The handle of the window that will or did display the menu to which the call applies
activation-info	An object containing detailed information about the context of the call
menu-index	The index of the top-level menu or cascading entry from which the call originates
selection-status	Tells whether the procedure has been called before menu selection (true) or after menu unselection (false)

Caution To avoid unexpected errors in the menu system, write your selection callbacks so that they do not enter wait states. Actions that will cause a procedure to enter a wait state are allow other processing, collect data, wait, and do in parallel. Avoid the use of these actions and any procedure which uses them. If there is no alternative, use gms-lock-menus and gms-unlock-menus to lock the menus while the selection callback is running.

When a cascading menu entry has both a selection and a posting callback procedure, and the user selects the entry, GMS calls first the selection and then the posting callback procedure. When the user unselects the entry, GMS calls first the posting and then the selection callback procedure.

To specify a selection callback procedure for a menu entry:

➔ Edit the menu entry template's gms-selection-callback attribute to specify a symbol that is the name of the procedure.

Because a selection callback procedure is not required for a menu to function correctly, if you do not define a selection callback procedure, GMS does not search for a default procedure, and continues executing without notification or error.

When GMS invokes a callback procedure, it passes the procedure three arguments that give the complete context of the invocation. GMS supplies various procedures that you can use to access this context and obtain any information about it that the callback procedure needs.

Dynamically Defining and Changing Menus

Shows how to create and modify GMS menus in real time.

Introduction

The techniques described in the previous chapters show you how to specify GMS menus statically, and how to check, disable, and restrict menu entries programmatically. These capabilities may not be adequate to deal with situations and requirements that change unpredictably.

GMS provides a variety of techniques that allow you to cope with unpredictably changing conditions. You can use these techniques to:

- Change the label of any menu entry in any G2 window at any time.
- Invoke a procedure whenever a menu is displayed or undisplayed.
- Invoke a procedure whenever a menu entry is selected or unselected.
- Dynamically define the contents of a popup menu.
- Dynamically define the contents of a cascade menu.
- Extend a menu by loading specifications in additional modules.

To see online examples of the concepts covered in this chapter:

1 Load the *gmsdemo.kb* from the *kbs* directory.

2 Choose the demo called Dynamic Menus.

   This demo shows how to generate a menu at run time without specifying the structure in advance.

3 Choose the demo called Multiple Menu Bars.

   This demo shows how to use the Switch Menu Bar template to provide access to multiple menu bars, possibly from different applications, in a single environment. The menus can therefore be specified in different modules.

Dynamically Constructed Menu Specifications

To provide for unpredictable situations, GMS allows you to completely define any popup or cascade menu programmatically. Dynamically defined popup and cascade menus are generically called **dynamic menus**.

Statically defined menus are preferable wherever they can be used, because they are faster. Dynamic menus are preferable when the need for real-time menu construction is paramount, because you can reconstruct them as needed to reflect current conditions.

Specifying a Dynamic Menu

Whether you specify a GMS menu statically or dynamically, the end result is the same: a menu specification that is input to the GMS compiler. The difference is in the techniques used for creating the menu specification.

To define a static menu, you first clone a root template or cascading entry template, then connect the template to additional templates as needed to define the menu.

To define a dynamic menu, you begin in the same way, by cloning a root or cascading entry template from the GMS Palette. A dynamic popup menu specification begins with a Dynamic Popup template; a dynamic cascade menu is specified with a Dynamic Cascade template. These two types of templates are generically called **dynamic templates**.

A dynamic template does not connect to any subsidiary templates. Instead, it specifies a G2 procedure called a **panel constructor**. When you ask GMS to display a dynamically defined menu, GMS calls the panel constructor specified by the menu's template object. The constructor returns a list of template objects that specify the contents of the desired menu. Such a list is called a **template list**.

- The templates in a template list are the same as they would be in an equivalent static menu specification, except that they are constructed using the G2 create and conclude actions. Such templates are called **dynamically constructed templates**.
- The sequential ordering defined by list membership replaces the sequential ordering defined with Peer Menu Connections in a static menu specification.

To compile a panel in a static menu specification, GMS traverses a sequence of cloned templates linked by connections. To compile a panel in a dynamic menu specification, GMS traverses a sequence of dynamically constructed templates linked by membership in a list. If the templates are the same, the resulting menu is the same.

Thus dynamic menu specification is a form of automatic code generation. All you need to do is clone dynamic templates from the GMS Palette and supply appropriate panel constructors for them. GMS does the rest.

Note Be careful not to confuse dynamic templates with dynamically constructed templates. Dynamic templates use panel constructors to define menus. Dynamically constructed templates are created by panel constructors and define menu entries.

Attributes of Dynamic Templates

The attributes of Dynamic Popup templates and Dynamic Cascade templates are identical to their static counterparts with the addition of attributes that specify the panel constructor function and control the deletion of dynamically constructed templates. These attributes are described in the following table.

Attribute	Description
gms-preconstruct-panel	Specifies that GMS reconstruct the menu every time it is displayed. If true, specifies that GMs constructs the menu at compile time.
Allowable values:	true or false
Default value:	false
Notes:	For more information about constructing dynamic menus see Constructing Dynamic Menus at Compilation Time.
gms-subpanel-constructor	Specifies the name of the panel constructor for GMS to use to build this menu.
Allowable values:	The name of any gms panel constructor function
Default value:	g2
gms-reclaim-templates	Specifies that dynamic templates are saved for later use after they are displayed. If false, the templates are deleted and recreated when needed.
Allowable values:	true or false
Default value:	true
Notes:	For additional information see Reusing Dynamically Constructed Templates.

Dynamic Menu Compilation and Display

By default, GMS completely constructs a dynamic menu from scratch every time the menu is to be displayed. The steps for constructing and displaying a dynamic menu are:

1 Call the panel constructor to obtain a template list that defines the menu.

2 Compile the template list into a menu translation.

3 Store the menu translation in the compiled resource.

4 Use the information in the compiled resource to build a menu instance.

5 Display the menu in a G2 window.

By default, GMS completely deletes a dynamic menu after any callback procedure invoked through the menu has returned. The steps for deleting a dynamic menu are:

1 Delete the menu instance.

2 Delete the menu translation from the compiled resource.

3 Delete all templates in the template list.

4 Delete the template list itself.

By default, GMS repeats this sequence in its entirety every time it displays and undisplays a dynamic menu. Such repetition can be time-consuming, particularly if you repeatedly select and unselect the same dynamic cascading menu entry.

GMS provides techniques for reducing the overhead of repeatedly reconstructing dynamic menus, as described under Reducing Dynamic Menu Overhead.

Panel Constructor Procedure Syntax

Syntactically, a panel constructor is an ordinary G2 procedure. It receives the same three arguments as an ordinary callback procedure, as described under Invoking a Procedure Upon Menu Selection, plus an additional argument that the constructor uses to return a dynamic menu specification. The signature of a panel constructor is:

procedure-name
  (*handle*: integer, *activation-info*: class gms-activation-info,
  menu-index: integer, *template-list*: class item-list)

The *procedure-name* can be any unique name. The arguments are:

Argument	Description
handle	The handle of the window in which the dynamically constructed menu will be displayed.
activation-info	An object containing detailed information about the context of the call to the panel constructor.
menu-index	The menu index of the cascading entry from which the call originates
template-list	An empty item list for holding dynamically constructed templates.

Every panel constructor must take exactly these four arguments: you cannot omit any of them or define any in addition to them. This chapter refers to the first three arguments that GMS passes to a panel constructor as **constructor arguments**.

A panel constructor can use the same techniques and is subject to the same warnings, as an ordinary callback procedure, as described in Writing and Using Callback Procedures. If you call gms-get-activation-level on the *activation-info*, the result is the level of the template that called the panel constructor.

A panel constructor is free to ignore any or all constructor arguments, and to make any desired use of them. Typically a constructor uses its arguments to determine what to include in the menu it constructs. The techniques by which a constructor obtains necessary information and uses it to determine what panel to construct are not part of GMS.

To specify a menu, a constructor defines the constituent templates, and puts them into the *template-list*. The zeroth element of the list holds the template for the topmost menu entry, the first element, the template for the next entry, and so on. Thus a panel constructor returns a menu specification by modifying an existing list, rather than creating and returning a new one.

Writing a Panel Constructor

The general technique for specifying an entry in a dynamic menu is:

1 Determine what the entry should consist of.

2 Use the create action to create an appropriate template object.

3 Use conclude actions to assign values to template attributes as needed.

4 Append or insert the element into the template list.

A panel constructor does not have to execute this sequence as a whole for each template: it can define a template list in any order that works. All that matters is that the list contain a complete and correct panel specification when the constructor returns. If the GMS compiler cannot compile a template list, GMS signals an error.

Caution To avoid unexpected errors in the menu system, write your panel constructor so that it does not enter wait states. Actions that will cause a procedure to enter a wait state are allow other processing, collect data, wait, and do in parallel. Avoid the use of these actions and any procedure which uses them. If there is no alternative, use gms-lock-menus and gms-unlock-menus to lock the menus while the panel constructor is running.

You can include a dynamic template in any menu, including one that was itself defined by a dynamic template, and so on to a maximum of 256 levels. When you nest dynamic templates, GMS does not construct the submenus when it constructs the menu that contains them: it waits until a submenu must be displayed, and constructs it then.

The following procedure is an example of a panel constructor.

```
gms-construct-subclass-panel (GMSUserHandle: integer,
    ActivationInfo: item-or-value, Index: integer, TemplatesList: class item-list)
SelectedOnWindow: truth-value;
MCT: class gms-choice-template;
CDT: class gms-dynamic-cascade-template;
ClassName, SubClassName: symbol;
Label: text;
X: integer = 0;
begin
    Label = call gms-get-activation-property (ActivationInfo, Index,
        the symbol GMS-LABEL);
    ClassName = symbol(Label);
    {inform the operator that "Constructing panel for class: [Label]";}
    for SubClassName = each symbol that is an inferior-class of ClassName do
        X = x + 1;
        if there exists a symbol that is an inferior-class of SubClassName then begin
            create a gms-dynamic-cascade-template CDT;
            conclude that the gms-label of CDT = "[SubClassName]";
            conclude that the gms-reclaim-templates of CDT = true;
            insert CDT at the end of TemplatesList;
            conclude that the gms-subpanel-constructor of CDT =
                the symbol gms-construct-subclass-panel;
        end else begin
            create a gms-choice-template MCT;
            conclude that the gms-label of MCT = "[SubClassName]";
```

```
                insert MCT at the end of TemplatesList;
            end;
            exit if x > 15;
        end;
    end
```

Specifying Labels for Templates in Dynamic Panels

If you are using the internationalization features of GFR, you have two options for specifying labels in the templates that are added to the template list in the panel constructor:

- Continue to use GFR symbols (as described in Internationalizing GMS Menus).

  Continuing to use GFR symbols is advisable if the templates are *reused* in any way for different windows.

- Specify the labels as text.

  If a template is *used only once*, you can insert the translated label text directly, because the language for the labels is unambiguously determined from the window where the current constructor is being used.

Using Template Lists

GMS assumes that it has complete control over template lists, and can create, modify, and delete them at will. Therefore, you should never do anything with a template list except to populate it with templates during execution of the panel constructor that received the list from GMS.

After a panel constructor returns, and GMS has compiled the panel specification in the template list, GMS:

- Always empties the template list.
- Optionally deletes the templates that the list contains.

To avoid the overhead of repeatedly creating the same templates, you can set GMS to not delete dynamically constructed templates, then reuse the templates in subsequent invocations of any panel constructor, as described under Reusing Dynamically Constructed Templates.

Defining and Displaying a Dynamic Cascade Menu

At the top level, defining a dynamic cascade menu is the same as defining a static cascade menu, except that you use a different entry template, a **Dynamic Cascade**

template. This template has the same attributes as an ordinary Cascade Menu template, plus attributes that specify the panel constructor function and control the deletion of dynamically constructed templates.

To define a dynamic cascade menu:

1 Clone a Dynamic Cascade template from the GMS Palette to a workspace.

2 Set attributes of this template as needed for any cascade menu, as described in Defining a Simple Menu..

3 Set the gms-subpanel-constructor of the template to specify the appropriate panel constructor.

4 Include the template in a menu specification just as you would any cascading menu entry.

GMS automatically creates and displays a dynamic cascade menu whenever you select its cascading menu entry. No special action is needed.

Defining and Displaying a Dynamic Popup Menu

At the top level, defining a dynamic popup menu is the same as defining a static popup menu, except that you use a different root template: a **Dynamic Popup template**. This template has the same attributes as an ordinary Popup Menu template, plus attributes that specify the panel constructor function and control the deletion of dynamically constructed templates. These attributes are shown in the following table.

To define a dynamic popup menu:

1 Clone a Dynamic Popup template from the GMS Palette to a workspace.

2 Set attributes of this template as needed for any popup menu, as described in Defining Popup Menus.

3 Set the gms-subpanel-constructor of the template to specify the appropriate panel constructor.

Displaying a dynamic popup is the same as displaying an ordinary popup.

To display a dynamic popup menu:

➔ Use gms-display-popup-menu, as described under Using G2 Configurations to Display a Popup Menu.

or

➔ Use the gms-popup-subscriber mixin, as described under Using gms-popup-subscriber to Display a Popup Menu.

or

➔ Use the techniques described under Configuring GMS to Handle Mouse Events.

Note In a static menu GMS automatically checks and unchecks menu entries on a gms-change-mode-template. In a dynamic menu, the checking and unchecking of gms-change-mode-template menu entries does not happen automatically. It is done using the API procedures gms-check-menu-entry, gms-uncheck-menu-entry, and gms-menu-entry-is-checked. For more information on these procedures see Checking and Unchecking Menu Entries.

Dynamic Menus and Callback Procedures

From the viewpoint of a callback procedure, static and dynamic menus are identical. Every callback procedure technique works in the same way with either type of menu. This invariance exists because:

- A menu translation is the same whether it was compiled from a static or a dynamic menu specification.
- All menu-specific information in a callback procedure's activation context is drawn from the compiled resource.

The GMS compiler generates new menu indexes each time it recompiles a dynamic menu, and may not use the same index values from one compilation to the next. Therefore a callback procedure that is invoked repeatedly from the same dynamic menu must not assume that the menu indexes in callback arguments are the same from call to call.

GMS deletes the menu translation for a dynamic menu as soon as the menu is undisplayed. Therefore a dynamic menu index value obtained from an activation path should never be used outside the context of the callback procedure invocation that obtained the value.

Reducing Dynamic Menu Overhead

The overhead of repeatedly constructing and reconstructing a dynamic menu can be considerable, as described under Dynamic Menu Compilation and Display. GMS offers two techniques for reducing such overhead:

- You can construct a dynamic menu on compilation rather than on display.

- You can reuse dynamically constructed templates in successive panel constructor invocations.

Constructing Dynamic Menus at Compilation Time

By default, GMS reconstructs a dynamic menu every time the menu is displayed. In some cases, a menu changes only under conditions that require the menu to be recompiled in any case.

For example, consider a menu that is frequently redefined while an application is under development, but does not change while the application is in use. Reconstructing such a menu for every display is unnecessary. The menu needs to be constructed only when GMS compiles it. This technique is called **preconstruction**. A menu that uses it is a **preconstructed menu**.

To set GMS to preconstruct a dynamic menu:

1. Define the menu as described in this chapter.
2. Set the dynamic template's gms-preconstruct-panel attribute to true.

When GMS compiles the menu, it will:

- Call the panel constructor to obtain a dynamic menu specification.
- Compile the specification.
- Add the translation to the compiled resource.

GMS thereafter handles the menu exactly as it does a statically defined menu. Thus a preconstructed menu is really not a static or a dynamic menu, but a hybrid of the two, having some of the properties of each.

Reusing Dynamically Constructed Templates

By default, every construction of a dynamic menu is independent of every other. In many cases, successive displays of a dynamic menu have much in common. Recreating every template in such a menu's specification is unnecessary. Only templates that change from one display to the next need to be recreated.

To avoid redundant template creation, you can set GMS to not delete dynamically constructed templates after using them. You can then reuse the templates as needed, rather than recreating them from scratch each time a dynamic menu is to be displayed. You can specify whether dynamic templates are saved or deleted separately for each dynamic menu.

To set GMS to save dynamically constructed templates for a dynamic menu:

➔ Set the dynamic template's gms-reclaim-templates attribute to false.

GMS will not delete dynamically constructed templates returned to the dynamic template by its panel constructor. GMS will still delete the template list that contains the templates: only the templates themselves are preserved.

To reuse a dynamically constructed template:

➔ Use standard G2 techniques to obtain the template and reuse it as needed in a panel constructor; this constructor need not be the same one that originally constructed the template.

Memory Management Considerations

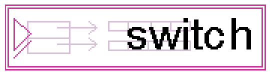
Reusing dynamically constructed templates can result in faster construction of dynamic menus, but it brings with it a responsibility for memory management that does not arise when GMS deletes such templates automatically.

When GMS deletes dynamically constructed templates, you do not need to worry about memory management. When GMS saves such templates, you must be sure to delete them, and any other objects that you create to help keep track of them, as soon as they are no longer needed. If you fail to do this, the accumulation of undeleted objects will eventually consume all available storage, and G2 will cease to function.

Caution When you reuse dynamically constructed templates, perfect storage management is essential! You must make *absolutely certain* that your code does not allow objects to accumulate without limit.

Dynamically Switching between Applications

When more than one G2 application is loaded simultaneously, and each can display a menu bar, the user typically needs a way to switch from one menu bar to another, and thus from one application to another. The **Switch Menu Bar template**, shown below, provides this capability.



In general, every G2 application that uses a GMS menu bar and might be loaded along with another such application should provide the ability to switch menu bars by including a Switch Menu Bar template in its menu bar. Otherwise a user who enters the application by switching to its menu bar will not be able to leave conveniently.

Switching Menu Bars

The Switch Menu Bar template is a subclass of Dynamic Cascade template. When selected, the template calls a panel constructor that:

1. Obtains all compiled menu bars from the compiled resource.
2. Extracts the menu label (the value of gms-label) and the menu index (gms-index) from the root template of each menu bar.
3. Creates a cascade menu that contains a leaf entry for each menu bar. This entry:
    - Has an entry label that is the label of the menu bar.
    - When chosen, calls gms-display-menu-bar to display the menu bar in the window that displays the Switch Menu Bar template.

To include a Switch Menu Bar template in a menu specification:

➔ Clone the Switch Menu Bar template from the GMS Palette and include it in the specification, as with any dynamic cascade template.

Including Additional Features in a Menu

Shows how to include some additional features of GMS in your menu. These features include menu dividers, menu accelerators, menu help information, distributing menu specifications over several workspaces, and creating reusable menu definitions.

Introduction

This chapter describes how to add some additional features to your GMS menu. You learn how to change the appearance of a menu by adding divider templates to the menu specification. You begin to specialize each menu entry by showing that the entry leads to a dialog, adding menu accelerators, and adding help text. You learn how to distribute menu specifications over more than one workspace and how to reuse the same menu definition in several menus.

The concepts in this chapter are covered separately in several demos.

To see online examples of the concepts covered in this chapter:

1 Load the `gmsdemo.kb` from the `kbs` directory.

2 Choose the following demos:

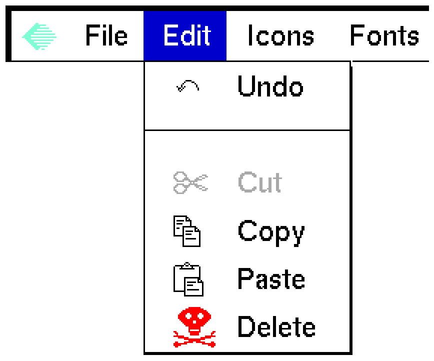
- A Complex Menu Bar
- User Extensibility
- Reusable Panels

Divider Templates

GMS menus can contain one or more of the following dividers to customize the menu appearance.

Separators

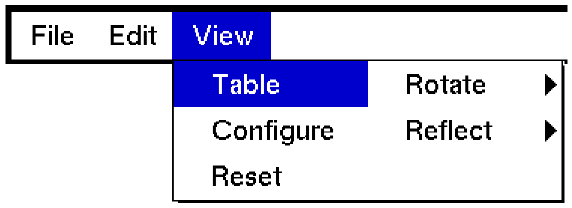
Separators draw horizontal lines between transient menu entries as shown in the following figure:



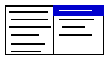
To achieve this, use a **Separator template** (shown below). Place a Separator template between two entry templates in a transient menu specification to tell GMS to place a horizontal line between the two entries.

Breaks

Breaks divide menu bars into rows and transient menus into columns as shown in the following figure:



To achieve this, use a **Break template** (shown below). Place a Break template between two entry templates to tell GMS to display the second entry at the beginning of a new row in a menu bar or the top of a new column in a transient menu.



When you put a break template between two entry templates in a transient menu, GMS displays the entry following the break template at the top of a new column in the menu's panel. GMS can optionally separate the columns with a vertical line.

To specify that GMS should not represent a break in a transient menu by drawing a vertical line:

➔ Verify that the break template's gms-show-vertical-bar attribute has the value false (the default).

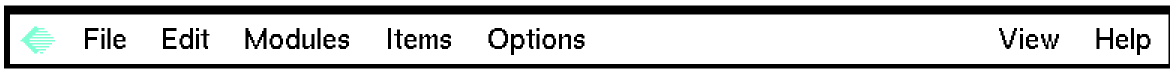
To specify that GMS should represent a break in a transient menu by drawing a vertical line:

➔ Edit the break template's gms-show-vertical-bar attribute to have the value true.

The gms-show-vertical-bar attribute has no effect on a break in a menu bar.

Justifiers

Justifiers cause menu bar entries to be right-justified as shown in the following figure:



To achieve this, use a **Right Justifier template** (shown below). Place a Right Justifier template between two entry templates in a menu bar specification to tell GMS to right-justify the entries that are specified after the template. If the window is resized to be of a smaller width than will accommodate the entire menu bar, GMS will rearrange the entries into multiple rows so that they are all accessible.

Specifying Additional Menu Entry Properties

In Defining a Simple Menu you learned to specify whether a menu entry is initially enabled, and/or initially checked. You also learned to specify the effect of choosing a leaf entry. In this section, you learn to specify additional menu entry properties.

Specifying a Dialog Entry on a Menu Entry Label

A dialog entry is a leaf entry that posts a dialog when chosen. By convention in standard menu systems, the label of a dialog entry is followed by an ellipsis (...).

The G2 text editor uses an ellipsis to indicate the site of a syntax error. You must therefore specify an ellipsis in a dialog entry label using the G2 escape character, @. For example, if you want a label to read "Save As..." you enter "Save As@.@.@. "

The text editor strips escape characters when it closes. If you again edit a label that has an ellipsis, replace the stripped escape characters before you close the editor.

Specifying a Menu Entry Accelerator Label

A menu entry accelerator label is a text string that describes the corresponding accelerator in some way. An **accelerator** is a keystroke that has the same effect as choosing a menu entry. Typing the keystroke executes the same user-defined procedure that choosing the menu entry executes.

The techniques that bind an accelerator to a user-defined procedure are not part of GMS: they are standard G2 techniques. For an example, see Implementing Keyboard Accelerators.

Note GMS does not require any menu entry to have an associated accelerator, nor does GMS require a menu entry with an associated accelerator to contain an accelerator label.

To specify an accelerator label in a menu entry:

➔ Edit the entry template's gms-accelerator-label attribute to specify the desired accelerator label as either:

- A quoted string that gives the text literally.
- A GFR symbol.

The accelerator label that you specify will appear to the right of the menu entry label whenever the entry is visible to the user.

Specifying Help Information

Menu entry Help information is a text string that provides information about the entry in addition to that provided by its entry label.

To specify Help information for a menu entry:

➔ Edit the entry template's gms-help-label attribute to specify the desired Help information as either:

- A quoted string that gives the text literally.
- A GFR symbol.

The specified information will appear in the Help bar at the bottom of the G2 window whenever the menu entry is selected and GMS is configured to display Help information. You can turn Help information on and off, and control the font in which GMS displays it.

To configure GMS to display Help information you edit the attribute table of the User Preference object for the G2 window in which the help information is displayed. For more information on the User Preference object, see Managing User Preferences.

To set GMS to display Help information:

1 Set gms-show-help-message to be true.

The default setting is false, so Help information is not shown unless the system is specially configured to display it.

2 Set gms-priority to be a higher number than that of any of the other preference objects. For more information on setting the priority of user preference objects see Specifying Generic User Preferences.

To specify the font size used to display Help information:

➔ Set gms-help-message-fontsize to be small, large, or extra-large.

Placing Menu Specifications on Multiple Workspaces

You do not have to define a menu specification on a single workspace: you can distribute it across as many workspaces as you like. GMS offers two techniques that facilitate distributed menu specifications: connection posts and SubPanel Containers. These techniques are particularly helpful with a complicated menu specification that contains many levels of cascading entries.

Using Connection Posts to Distribute Menu Specifications

The GMS Palette provides connection posts for both types of GMS connection. You can use these to distribute a menu specification onto subworkspaces. The connection posts are:

- **Submenu Connection Post**: Links a Submenu Connection across workspaces.

- **Peer Menu Connection Post**: Links a Peer Menu Connection across workspaces.

You can also use a Peer Menu Connection Post to distribute a menu specification across modules, as described under Extending Menu Specifications across Modules.

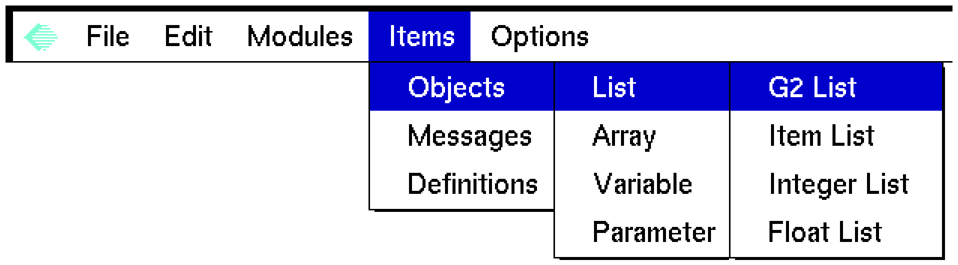
To distribute a menu specification using connection posts:

➔ Use connection posts from the GMS Palette in conjunction with standard G2 techniques for distributing a G2 diagram across workspaces.

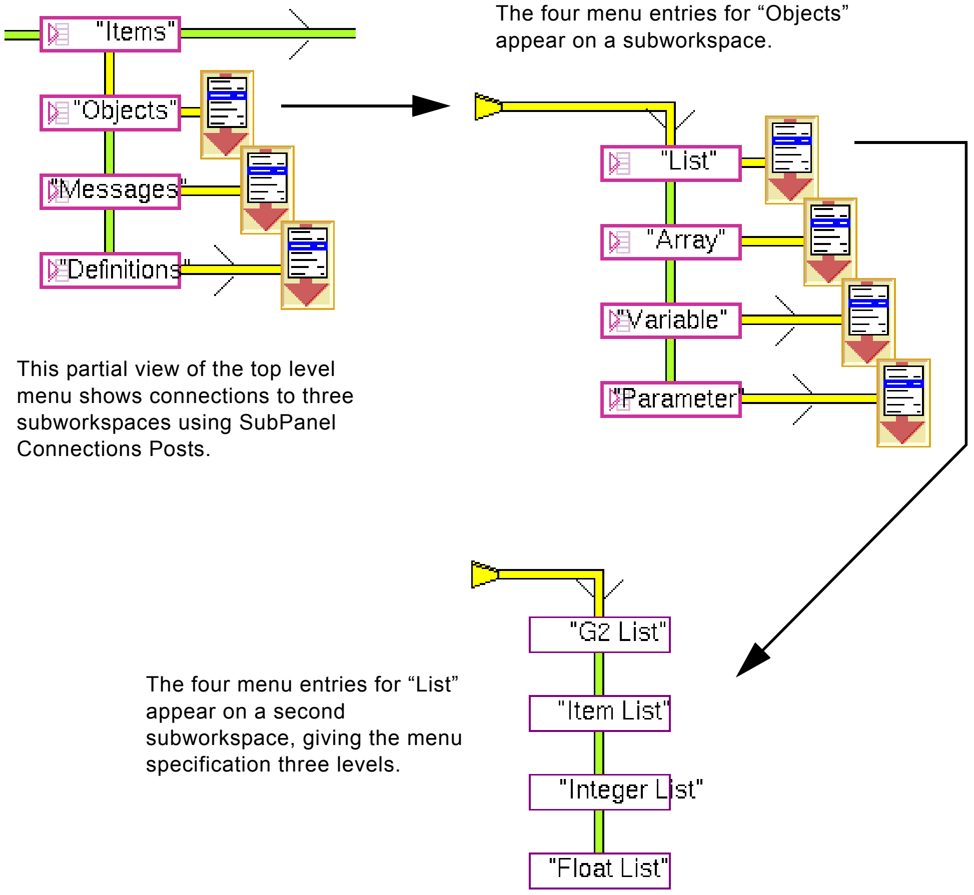
Using SubPanel Containers to Distribute Menu Specifications

The GMS palette contains a template called a **SubPanel Container** that provides a subworkspace for holding part of a menu specification. SubPanel Containers have no functional significance. They are purely a representational convenience.

The following figure shows a walking menu for the “Items” pulldown menu. The last two panels of the walking menu are specified on subworkspaces using SubPanel Containers.



The graphical menu specification appears on three workspaces, as shown in the following figure:



The four menu entries for “Objects” appear on a subworkspace.

This partial view of the top level menu shows connections to three subworkspaces using SubPanel Connections Posts.

The four menu entries for “List” appear on a second subworkspace, giving the menu specification three levels.

To distribute a menu specification using a SubPanel Container:

1 Clone a SubPanel Container to the workspace that contains the menu specification.

2 Use a Submenu Connection to connect the existing menu specification to the SubPanel Container.

3 Create a subworkspace for the SubPanel Container.

4 Open the subworkspace for the SubPanel Container.

The subworkspace contains a generic connection post that you can use like a Submenu Connection Post to continue the menu specification. Alternatively, you can substitute an actual Submenu Connection Post.

To add a Submenu Connection Post to a SubPanel Container subworkspace:

1 Clone a Submenu Connection Post to the subworkspace.

2 Copy the value of the superior-connection attribute of the predefined connection post to the superior-connection attribute of the Submenu Connection Post.

3 Delete the predefined connection post from the subworkspace.

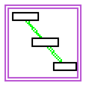
You can nest SubPanel Containers to any depth, provided that the resulting specification does not have more than 256 levels.

Extending Menu Specifications across Modules

In some G2 applications, completely defining a menu in one module is inconvenient or impossible. For example, consider a proprietary module that is to be extended by users in ways that cannot be anticipated. The users cannot change the module, because it is proprietary, but they must be able add to its menus, or they will not be able to extend its capabilities conveniently.

To provide for such situations, GMS allows you to extend a menu specification across modules using *extensible stubs*. There are two template objects for extending your menu specification across modules:

- **Place Holder** allows you to extend a menu specification across two modules. It specifies a connection between two menu entries in a panel. The connection spans two modules.



- **Peer Menu Connection Post** that is connected to the last entry on the right side of a menu specification.

Both of these stubs must be given Names in order for them to be “continued” in another module. When the GMS compiler encounters an unresolved stub, it ignores the stub, just as it would a dangling connection stub. Thus you can use stubs to provide optional hooks for extending menu specifications. A menu specification that uses this technique is called an **extensible menu specification**.

To create an extensible menu specification:

1. Begin the menu specification as you would any GMS menu specification.
2. Perform *one* of these actions:
   - Hook up a Place Holder within the definition of a menu panel.
   - Connect the last entry in a menu specification to a Peer Menu Connection Post.
3. Give the stub a unique name.

To continue an extensible menu specification in another module:

1. In the module that defines the extension to the specification, perform *one* of these actions:
   - Create a Place Holder and change its gms-compiled-stub-name attribute to the name of the Place Holder in the panel.
   - Create a Peer Menu Connection Post that has the name of an unresolved post on the specification that is to be extended.
2. Connect the stub to the first entry to be added by the module.
3. Define the rest of the menu as you would in any menu specification.

The menu specification in the second module can again contain a Place Holder or end with a uniquely named Peer Menu Connection Post, which may or may not be continued in a third module, and so on through any number of modules.

Note This technique works only with Peer Menu Connection Posts: you cannot include an unresolved Submenu Connection Post in a menu specification. The GMS compiler will signal an error if it encounters such a post.

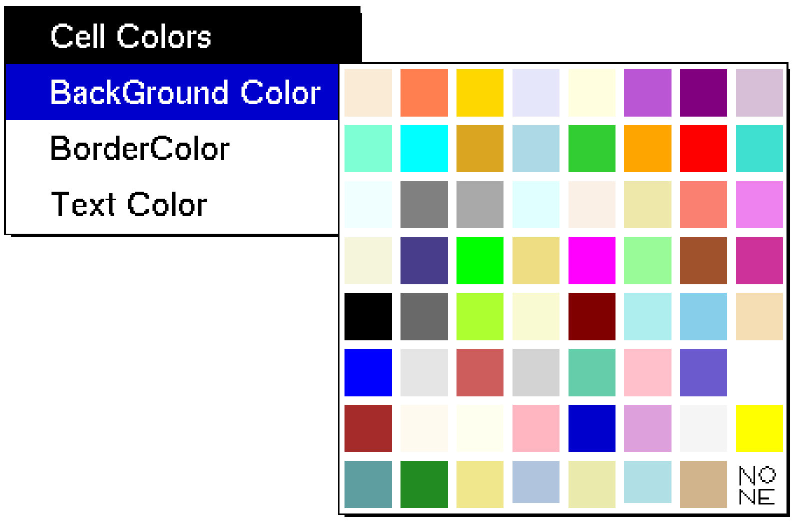
Creating Reusable Cascade Menus

Sometimes a cascade menu is useful in more than one place in a menu specification. For example, a menu that offers a choice of colors might be useful for setting text color, foreground color, background color, and various other colors.

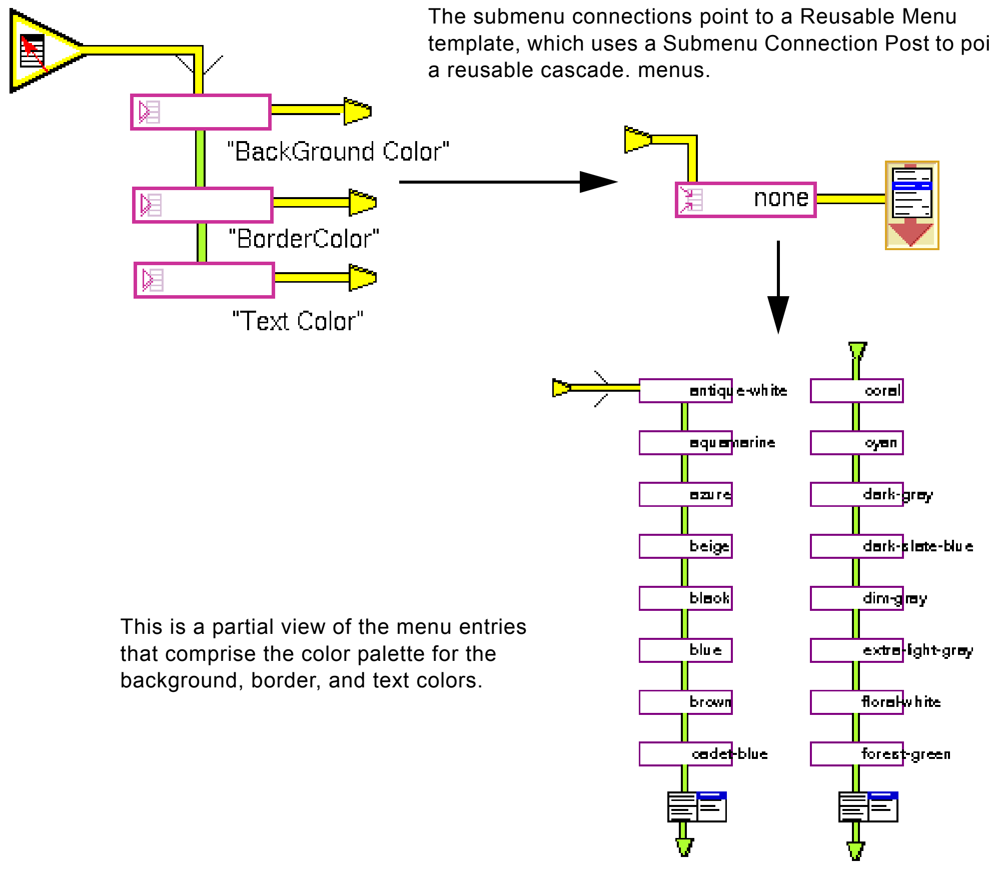
You could define a separate, identical cascade menu for each color setting, but such redundant effort would be wasteful.

GMS provides a **Reusable Panel template** that you can use to include the same cascade menu in a menu specification as many times as you like.

In the following figure, a popup menu enables you to select colors for background, borders, and text. The color selections are the same for all three menu entries. These entries are specified in a reusable panel.



The menu specifications use the Reusable Panel template, which uses submenu connections to attach to the reusable cascade menus.



The submenu connections point to a Reusable Menu template, which uses a Submenu Connection Post to point to a reusable cascade. menus.

This is a partial view of the menu entries that comprise the color palette for the background, border, and text colors.

To use a cascade menu more than once in a menu specification:

1. Clone a Reusable Panel template to a subworkspace.
2. Use Submenu Connections to connect the template to every cascading entry that is to display the reusable cascade menu. To avoid complex many-to-one connections, use Submenu Connection Posts.
3. Define the reusable cascade menu as a submenu of the Reusable Panel template. The connection between the reusable template and the first entry in the cascade menu is a Submenu Connection, *not* a Peer Menu Connection.

The cascading menu will appear in the menu whenever the user selects any cascading entry whose template connects to the Reusable Panel template.

Part III

Controlling GMS

Chapter 10: **Controlling Access to Menus**

Shows how to control access to menus by locking all menus against user input, disabling and enabling individual menu entries, and restricting menu entries in specified user modes.

Chapter 11: **Controlling the Appearance of Icons**

Shows how to specify an icon for a menu entry, and customize its color and size.

Chapter 12: **Internationalizing GMS Menus**

Summarizes the techniques for displaying GMS menus in different languages, using Gensym Foundation Resources (GFR) to provide internationalization.

Chapter 13: **Configuring Global GMS Characteristics**

Shows how to set properties that affect GMS as a whole such as automatic GMS startup, preserving the compiled resource on reset, specifying the maximum number of menu entries, suppressing global consistency checking, and internationalization.

Chapter 14: **Customizing the GMS Interface to the User**

Shows how to set GMS properties for individual G2 users. These properties include: font and separator size, help display, text and background colors, navigation modes, menu blinking, initial menu, and menu language.

Chapter 15: Specifying the Interface between GMS and G2

Shows how to configure GMS to work with and extend G2 capabilities such as accessing a compiled resource from G2, implementing keyboard accelerators, and handling mouse events for popup menus.

Chapter 16: Managing GMS Programmatically

Shows how to manage programmatically the internal operations of GMS. You can use the techniques described to manage menu building and compilation, change global settings, manage user preferences, and reset GMS without resetting G2.

Controlling Access to Menus

Shows how to control access to menus by locking all menus against user input, disabling and enabling individual menu entries, and restricting menu entries in specified user modes.

Introduction

This chapter introduces three methods of controlling access to menus.

To see online examples of the concepts covered in this chapter:

1 Load the *gmsdemo.kb* from the *kbs* directory.

2 Choose the demo called User Mode Restrictions.

This demo shows one of the methods of controlling access to menus.

Controlling Access to Menus

GMS provides a variety of API calls that poll and set the accessibility of menus and menu entries. The calls control:

- Locking all menus against user input.
- Disabling and enabling individual menu entries.
- Restricting menu entries in specified user modes.

The dimensions of locking, disabling, and restricting are orthogonal. They can be changed independently of one another and do not modify each other's effects in any way.

Locking All Menus Against User Input

You can **lock** all GMS menus in a G2 window against user input at any time. While menus are locked, GMS discards all mouse events:

- Moving or clicking the mouse over a menu entry does nothing.
- Clicking an object that would otherwise display a popup menu does nothing.

When you **unlock** menus, mouse input resumes. When you lock or unlock menus, the appearance of the menus does not change, and any transient menus that are visible remain on display.

You can use menu locking to prevent the user from choosing menu commands while an application is in the process of loading or initializing, or is performing some function that precludes asynchronous user input.

By default, GMS menus remain active while a callback procedure executes. In some cases, menus should be locked during callback procedure execution, to prevent additional menu selections from changing the environment that the procedure assumes.

To lock menus during execution of a callback procedure:

➔ Edit the entry template's gms-lock-during-callback attribute to have the value true.

To lock all GMS menus in a G2 window:

➔ gms-lock-menus
   (*handle*: integer)

   Locks the menus of the window referenced by *handle*. If the menus are already locked, the call has no effect.

To unlock all GMS menus in a window:

➔ gms-unlock-menus
  (*handle*: integer)

  Unlocks the menus of the window referenced by *handle*. If the menus are already unlocked, the call has no effect.

To determine whether menus are locked:

➔ gms-menu-is-locked
  (*handle*: integer)
  -> *status*: truth-value

  Returns true if the menus of the window referenced by *handle* are locked, and false otherwise.

Disabling and Enabling Menu Entries

You can **disable** or **enable** any menu entry in any window at any time. An enabled menu entry is fully functional. A disabled menu entry is only partly functional:

- Selecting a disabled cascading entry does *not* display its cascade menu.
- Choosing a disabled leaf entry does *not* execute its callback procedure.

When you disable an entry, GMS changes its color for the duration of the disablement. You can specify the color that GMS uses, as described under Default Highlightable Icon Colors.

To disable a menu entry:

➔ gms-disable-entry
  (*handle*: integer, *menu-index*: integer)

  Disables the menu entry referenced by *menu-index* in the window referenced by *handle*. If the entry is already disabled, the call has no effect.

To enable a menu entry:

➔ gms-enable-entry
  (*handle*: integer, *menu-index*: integer)

  Enables the menu entry referenced by *menu-index* in the window referenced by *handle*. If the entry is already enabled, the call has no effect.

To determine if an entry is disabled:

➔ gms-entry-is-disabled
(*handle*: integer, *menu-index*: integer)
-> *status*: truth-value

Returns true if the menu entry referenced by *menu-index* in the window referenced by *handle* is disabled, and false otherwise.

Restricting Menus in Specified User Modes

You can disable individual menu entries on a per-user-mode basis at any time. When you **restrict** an entry for a user mode, GMS disables the entry in any G2 window where that mode is the current user mode. When you **unrestrict** an entry for a user mode, you return it to normal function relative to that mode.

When different windows that display a restricted entry are in different user modes, the effect of the restriction depends on the user mode of each window. Thus a restricted entry can be disabled in one window, and enabled in another.

You can restrict or unrestrict a menu entry for any user mode, and for any number of user modes. Calls that specify restrictions or unrestrictions that are already in effect execute without error and have no effect.

You can specify a restriction either positively or negatively:

- A positive specification restricts an entry when the user *is* in a specified mode. To give such a restriction, specify the mode as a symbol.
- A negative specification restricts an entry when the user *is not* in a specified mode. To give such a restriction, specify the mode as a symbol prefixed by a tilde (~).

Caution The symbol tilde (~) is not a valid character for symbol names. You must therefore specify a tilde in the text editor using the G2 escape character @. For example, if you want to use the symbol ~developer in your procedure, you must enter @~developer in the text editor.

To restrict a menu entry:

➔ gms-restrict-entry
 (*menu-index*: integer, *mode-restriction*: symbol)

Restricts the menu entry referenced by *menu-index* as specified by *mode-restriction*. For example:

```
gms-restrict-entry (1, the symbol developer)
```

restricts the menu entry whose index is 1 such that the entry is disabled in any G2 window whose user mode is developer. Conversely:

```
gms-restrict-entry (1, the symbol ~developer)
```

restricts the menu entry whose index is 1 such that the entry is disabled in any G2 window whose user mode is *not* developer.

You can call gms-restrict-entry as many times as needed to restrict all user modes for which restriction is desired.

Calling gms-restrict-entry on a root template will have the following effect on the menus:

- If the current menu that is currently displayed becomes inaccessible as a result of adding the restriction, it will be hidden.
- Access to a menu bar through a switch menu panel is restricted.

If the menu later becomes accessible, either through a mode change or through a call to gms-unrestrict-entry, the menu bar will again be displayed.

To unrestrict a menu entry:

➔ gms-unrestrict-entry
 (*menu-index*: integer, *mode-restriction*: symbol)

Unrestricts the menu entry referenced by *menu-index* as specified by *mode-restriction*. The effect is to undo the effect of the analogous call to gms-restrict-entry. For example:

```
gms-unrestrict-entry (1, the symbol developer)
```

unrestricts the menu entry whose index is 1 such that the entry is not disabled due to restriction in any G2 window whose user mode is developer. Conversely:

```
gms-unrestrict-entry (1, the symbol ~developer)
```

unrestricts the menu entry whose index is 1 such that the entry is not disabled due to restriction in any G2 window whose user mode is *not* developer.

You can call gms-unrestrict-entry as many times as needed to unrestrict all user modes for which restriction is not desired.

To determine if an entry is restricted:

➔ gms-entry-is-restricted
(*menu-index*: integer, *mode-restriction*: symbol)
-> *status*: truth-value

Returns true if the menu entry referenced by *menu-index* is currently restricted as specified by *mode-restriction*, and false otherwise. For example, after executing:

```
gms-restrict-entry (1, the symbol developer)
```

the call:

```
gms-entry-is-restricted (1, the symbol developer)
```

returns true. Conversely, after executing:

```
gms-unrestrict-entry (1, the symbol ~developer)
```

the call:

```
gms-entry-is-restricted (1, the symbol ~developer)
```

returns false.

Undisplaying All Menus

GMS cannot detect a mouse event that was not initiated over a GMS menu. Hence clicking the mouse outside of any menu has no effect on any transient menus that are on display. Some applications need to undisplay all transient menus when the user clicks outside of any menu.

To undisplay all transient GMS menus:

➔ gms-dismiss
(*handle*: integer)

Undisplays all transient menus on the window referenced by *handle*. If no transient menus are currently displayed in the window, gms-dismiss has no effect. Calling gms-dismiss does not affect menu bars.

11

Controlling the Appearance of Icons

Shows how to specify an icon for a menu entry, and customize its color and size.

Introduction

A menu entry icon is a G2 icon that appears next to the entry's label and denotes the entry's effect graphically. You can include an icon in any GMS menu entry.

This chapter shows you how to specify an icon for a menu entry, and customize its color and size.

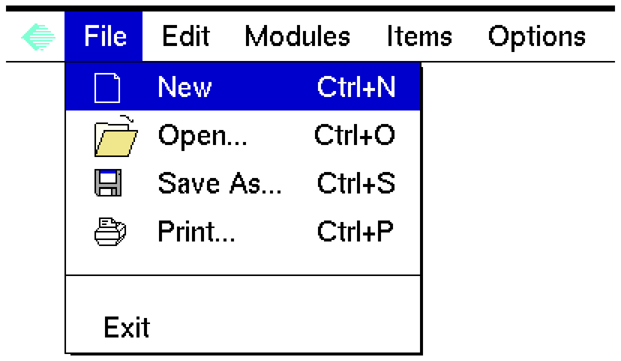
In order to customize the color or scaling of a GMS icon, you must understand GMS icon configuration in general, as described in Configuring a GMS Icon.

To see online examples of the concepts covered in this chapter:

1. Load the *gmsdemo.kb* from the *kbs* directory.
2. Choose the demo called Displaying Icons.

Specifying a Menu Entry Icon

Unlike G2 icons that represent objects, menu entry icons have no functional significance: they are purely illustrative. The menu entries in the next figure have icons as well as accelerator labels.



GMS provides the predefined class gms-icon for use in defining GMS icons. Every GMS icon is an instance of a subclass of gms-icon. The subclass definition defines the icon in its icon-description attribute, as with any G2 class.

GMS creates GMS icon instances as needed to display to the user. All you need to do is define a gms-icon subclass, and reference it in the entry template of each menu entry that uses the icon. You can define GMS icon subclasses on any workspace.

You do not have to use a direct or single descendent of gms-icon to specify a menu entry icon. You can define intermediate subclasses and use multiple inheritance as needed to create families of related icon definitions. For simplicity, this section assumes that your definition is a direct descendent of gms-icon.

To define a menu entry icon class:

1. Choose KB Workspace > New Definition > class-definition > object-definition.
2. Edit the class-name attribute to be any desired name.

   You use this name to reference the icon class in entry template definitions that use the icon.
3. Edit the direct-superior-classes attribute to include gms-icon.

To define the image definition to be used in the icon:

1 Choose KB Workspace > New Definition > image-definition.

2 Edit the following attributes of the image definition:

- Edit the name attribute to be any desired name.
- Give the file-name-of-image attribute a string value containing the pathname of the image definition.

3 Choose edit icon from the menu of the class definition.

4 Delete all layers of the default icon.

5 Provide the name of the image.

To specify a menu entry icon class for an entry template:

➔ Edit the entry template's gms-inline-icon-class attribute to specify the desired icon class.

The icon of the class that you specify will appear to the left of the menu entry label whenever the entry is visible to the user.

Controlling Icon Color

A GMS icon defined as a direct descendent of gms-icon has the colors specified in its icon description under all circumstances. You cannot use the change the icon color action to change GMS icon colors, because GMS does not provide direct access to icon instances.

However, you can use techniques described in this chapter to define a GMS icon class that uses a metacolor that changes color automatically. The metacolor can take on different colors depending on whether an entry that uses the icon class is selected, unselected, or disabled.

A class of GMS icons that can change color is called a **highlightable icon class**, and an icon of such a class is a **highlightable icon**.

To define a GMS icon class that changes color:

1 Create a new object definition as described above.

2 Include gms-highlightable-icon rather than gms-icon as the new class's direct-superior-classes.

3 Give the class an icon description that specifies one or more layers whose metacolor is foreground.

GMS changes the color of a highlightable icon by changing the color of all layers whose metacolor is foreground.

Caution When you define a highlightable icon, you must specify one or more layers whose metacolor is foreground in the icon description. Otherwise GMS will attempt to set the color of an unused metacolor, and G2 will signal an error.

4. Proceed as described in Configuring a GMS Icon below.

Configuring a GMS Icon

The attributes that control the color and scaling of a GMS icon are not part of the gms-highlightable-icon subclass that defines the icon. Defining them in the icon subclass would force every menu entry to use a given GMS icon class in the same way. To provide greater flexibility, the attributes that configure GMS icon appearance are defined separately by each template that uses the icon.

The necessary attributes could be specified in the template object itself, but this practice would burden every template with many attributes that typically are not needed. To prevent this:

1. A template object has only one attribute that relates to icon configuration: gms-inline-icon-description.
2. A separate class, gms-icon-specification, defines all icon configuration attributes.
3. A template that needs icon configuration includes a gms-icon-specification as the value of its gms-inline-icon-description attribute.
4. Attributes in the gms-icon-specification specify any special configuration applicable to the icon.

The instructions in this section tell you how to set up and access a gms-icon-specification.

To prepare to configure a menu entry's GMS icon:

1. Open the table of the template object that defines the entry.

   The template's gms-inline-icon-class attribute specifies the class of the icon. The template's gms-inline-icon-description attribute has the value none.
2. Click on the gms-inline-icon-description attribute somewhere away from the current value of the attribute (none).

   A G2 menu appears.
3. Select add optional-subtable > gms-icon-specification.

   The subtable of a gms-icon-specification appears.
4. Edit subtable attributes as desired.

The value of gms-inline-icon-description is now a gms-icon-specification.

To reconfigure a menu entry's GMS icon:

1 Open the table of the template object that defines the entry.

The template's gms-inline-icon-class attribute specifies the class of the icon. The template's gms-inline-icon-description attribute has the value a gms-icon-specification.

2 Click on the gms-inline-icon-description attribute.

A G2 menu appears.

3 Choose subtable.

The subtable of the gms-icon-specification appears.

4 Edit subtable attributes as desired.

Attributes of a GMS Icon Specification

The attributes of a gms icon specification are shown in the following table:

Attribute	Description
gms-icon-subclasses-exist	Specifies if icon subclasses have been defined for use with the three font sizes which can be used in menus.
Allowable values:	true or false
Default value:	false
Notes:	For more information on subclassing icons see Providing Different Icons for Different Fonts.
gms-maximize-icon	Controls font scaling by allowing GMS to enlarge an icon's size.
Allowable values:	true or false
Default value:	false

Attribute	Description
gms-preserve-ratio	Controls font scaling by directing GMS to preserve the icon’s aspect ratio when it scales the icon.
Allowable values:	true or false
Default value:	false
gms-do-not-reduce-width	Controls font scaling by directing GMS to preserve the width of an icon that is too wide, and to make the menu entry wider as needed to contain the icon.
Allowable values:	true or false
Default value:	false
gms-do-not-reduce-height	Controls font scaling by directing GMS to preserve the height of an icon that is too tall, and to make the menu entry taller as needed to contain the icon.
Allowable values:	true or false
Default value:	false
gms-no-margin-for-check	Specifies if GMS should leave room on the left of the menu entry for a check mark, the default, or if that space can be used for the icon.
Allowable values:	true or false
Default value:	false
Notes:	For more information see Making Additional Room for an Icon.

Attribute	Description
gms-icon-normal-color	Specifies the color of the icon's foreground layer in a menu entry that is enabled and not selected.
Allowable values:	Any G2 color
Default value:	none
gms-icon-highlighted-color	Specifies the color of the icon's foreground layer in a selected menu entry.
Allowable values:	Any G2 color
Default value:	none
gms-icon-disabled-color	Specifies the color of the icon's foreground layer in a disabled menu entry.
Allowable values:	Any G2 color
Default value:	none

The rest of this chapter refers to a gms-icon-specification as an **icon configuration object**, and assumes that such an object exists for each menu entry template for which icon configuration is needed.

Default Highlightable Icon Colors

If you use a highlightable icon in a menu entry, and do not do anything to specify the color(s) that its foreground layer(s) should have, GMS uses the text colors defined in the current User Preferences object.

When the menu entry is...	GMS sets the foreground color to...	Default...
Enabled but not selected	gms-normal-text-color	black
Selected	gms-highlighted-text-color	white
Disabled	gms-disabled-text-color	gray

Specifying Highlightable Icon Colors

You can also specify foreground colors that differ from the text colors defined in the current User Preferences object.

To specify the foreground color of layer(s) when an entry is enabled but not selected:

➔ Specify the color as the value of the icon configuration object's gms-icon-normal-color attribute.

To specify the foreground color of layer(s) when an entry is selected:

➔ Specify the color as the value of the icon configuration object's gms-icon-highlighted-color attribute.

To specify the foreground color of layer(s) when an entry is disabled:

➔ Specify the color as the value of the icon configuration object's gms-icon-disabled-color attribute.

Any or all of the colors you choose can be the same color. You can specify any icon color attribute as none (the default). GMS then defaults to the relevant text color, as listed in the previous table.

Other Uses for Highlightable Icons

You do not have to use the foreground color of a highlightable icon to indicate the menu entry state. For example, suppose you needed a menu that offered a choice of colors by showing examples of them. You could:

- Define a gms-highlightable-icon subclass that is a square whose color is foreground.
- Use this icon class for all leaf entries in the menu.
- Use icon color attributes differently in each leaf entry to give its icon the appropriate color.

Specifying Icon Scaling

By default, GMS shrinks a GMS icon as needed to fit into the space available in the menu entry that contains the icon. If such scaling does not provide an acceptable appearance, you can use techniques described in this chapter to scale the icon in ways that provide better results.

The space available to hold a GMS icon varies with the font size of the menu entry that uses the icon, as follows:

When the menu entry font size is...	The space available for an icon is...
Small	18x18 workspace units
Large	23x23 workspace units
Extra Large	33x33 workspace units

By default, GMS fits an icon into the space available for it as follows:

- When the icon can fit into the available space, GMS shows the icon without scaling it.
- When an icon is larger than the available space, GMS shrinks the length and/or width of the icon as needed to fit into the space.

These defaults usually produce an acceptable appearance, but sometimes they do not:

- Shrinking an icon causes loss of resolution that may obscure an icon's details.
- Shrinking length and width separately distorts an icon's aspect ratio.
- An icon smaller than the available space does not expand to take advantage of the space.

GMS offers a variety of techniques you can use to control GMS icon scaling. With these techniques, you can give an acceptable appearance to almost any icon. Be sure you have read Configuring a GMS Icon before you proceed.

Avoiding Icon Scaling Entirely

The simplest way to control icon scaling is to make it unnecessary. GMS does not need to scale an icon when the icon fits into the available space. If you know that your application will use only one font size, and can supply an icon that fits into the space indicated for that size, as shown in the previous table, you do not need to worry about scaling, because none will occur.

Providing Different Icons for Different Fonts

GMS allows you to avoid icon scaling by supplying three different gms-highlightable-icon subclasses, one for each font size. Each time GMS displays a menu entry that uses per-font icon classes, GMS uses the class that is correct for the font then in use. A set of per-font icon classes is called an **icon family**.

GMS recognizes icon families by their names. Each name begins with the value of the gms-inline-icon-class attribute of the template that uses the classes, suffixed by:

- -small for the icon class to use when the font is Small.
- -large for the icon class to use when the font is Large.
- -xlarge for the icon class to use when the font is Extra-Large.

To specify that a menu entry displays different icons with different fonts:

1 Define three subclasses of gms-highlightable-icon named *iconclass*-small, *iconclass*-large, and *iconclass*-xlarge, where *iconclass* is any unique G2 symbol.

2 Define an icon for each of the three icon classes such that *iconclass*-small has an icon whose size is ≤ 18x18, *iconclass*-large has an icon ≤ 23x23, and *iconclass*-xlarge has an icon ≤ 33x33.

3 Make *iconclass* the value of the gms-inline-icon-class attribute of the template that uses the icon family.

4 Set the icon configuration object's gms-icon-subclasses-exist attribute to be true.

GMS will now use the specified icon class with each font, switching automatically whenever the font changes. If you specify an icon that is larger than the space available when its font is in use, GMS scales the icon just as it would an icon that is not part of an icon family.

Controlling Icon Scaling

GMS provides four configuration object attributes that you can use to control icon scaling. By default, each of them is false.

Set this attribute to true...	To get this effect...
gms-maximize-icon	GMS can scale the icon by enlarging it as well as by shrinking it.
gms-preserve-ratio	GMS does not change an icon's aspect ratio when it scales the icon.
gms-do-not-reduce-width	GMS does not reduce the width of an icon that is too wide, but makes the menu entry wider as needed to contain the icon.
gms-do-not-reduce-height	GMS does not reduce the height of an icon that is too tall, but makes the menu entry taller as needed to contain the icon.

You can set these attributes in any combination to specify exactly the scaling behavior that you want.

Making Additional Room for an Icon

Ordinarily, GMS reserves space on the left side of a menu entry to hold a check mark. When you know that the entries in a menu will never be checked, you can tell GMS to make this space available to hold an icon.

To configure GMS to not reserve space for a check mark in a menu entry

➔ Set the configuration object's gms-no-margin-for-check attribute to true.

In general, if you set this attribute to true for one menu entry, you should set it for all entries in that menu, or the contents of the entries will not be aligned on the left.

Internationalizing GMS Menus

Summarizes the techniques for displaying GMS menus in different languages, using Gensym Foundation Resources (GFR) to provide internationalization.

Introduction

GMS menu entries typically convey information textually, and so must be in some language. If your application must run in different language environments, you can use Gensym Foundation Resources (GFR) to provide internationalization.

GFR allows you to specify text values as symbols called **GFR symbols** instead of strings. When GMS is configured to use GFR, and encounters a GFR symbol in a menu specification, GMS passes the symbol to GFR, which returns a string appropriate to the current language environment. GMS then uses that string as if it had appeared literally in the menu specification.

To see online examples of the concepts covered in this chapter:

1 Load the `gmsdemo.kb` from the `kbs` directory.

2 Choose the demo called Language Support.

Making Menu Text International

Menu entry labels, accelerator labels, and Help labels all convey information textually, and so must be in some language. If your application runs in only one language environment, you do not need to internationalize its menus. You can type text strings directly into GMS menu specifications.

If your application must run in different language environments, you can use Gensym Foundation Resources (GFR) to provide internationalization. GFR allows you to specify text values as GFR symbols instead of strings.

When GMS encounters a GFR symbol in a menu specification, it passes the symbol to GFR, which returns a string appropriate to the current language environment. GMS then uses that string as if it had appeared literally in the menu specification.

You can also use GFR as a convenience to avoid wiring strings into menu specifications. Such use can be convenient when labels are volatile, or when many menu entries use the same label. It is also a good practice generally; analogous to defining constants symbolically rather than wiring their values directly into code.

Note When you use GFR, you should specify all text in all menu specifications, using GFR symbols only. If you try to compile a menu specification that uses both text strings and GFR symbols to define textual information, GMS posts a warning or signals an error, as described under Suppressing Global Consistency Checking. If you mix text strings and GFR symbols, you must turn off global consistency checking.

In order to provide a translation of a GFR symbol, GFR needs three pieces of information:

- The translation dictionary in which the symbol is defined.
- The language into which to translate the symbol.
- The symbol itself.

For complete information on GFR, see the *G2 Foundation Resources User's Guide*. This section assumes you are familiar with GFR, and shows you how to configure GMS to use it.

Specifying that GMS is to Use GFR

The gms-use-translations attribute of the current Global Settings object controls whether GMS uses GFR to provide internationalization. For information about the global settings object see Configuring Global GMS Characteristics.

To specify that GMS does not use GFR to provide internationalization:

➔ Specify gms-use-translations in the current Global Settings object as false (the default).

All labels in menus must be strings. GMS will signal an error if it tries to compile a menu that uses a GFR symbol as a label.

To specify that GMS uses GFR to provide internationalization:

➔ Specify gms-use-translations in the current Global Settings object as true.

All labels in menus must be GFR symbols. GMS will signal an error if it tries to compile a menu that uses a string as a label.

Specifying the Language of Translation

When gms-use-translations is true, you must specify a language of translation for every user, either individually or by default.

To specify the default language into which GFR symbols are translated:

➔ Set gms-default-language in the current Global Settings object to specify the language.

To specify the language into which GFR symbols are translated for a user:

➔ Set gms-language in the User Preferences object applicable to the user to specify the language.

In addition to the name of a language, you may also specify that GMS use the default language for the window.

To specify that the language to be used is the language of the window:

➔ Set gms-language in the User Preferences object applicable to the user to the symbol window-or-system-default.

Using this symbol causes GFR to obtain the language through a call to gfr-language, using as an argument the window that the user is logged in on. If a window-specific language is designated for the window, gfr-language returns that language. If not, this function returns the value of the current-language attribute of the Language Parameters system table.

Specifying the Translation Dictionary

A GMS translation dictionary is called a **resource group** (gfr-text-resource-group). You can define any number of resource groups, and use them as needed in menu specifications.

You can specify a resource group for any template in a menu specification. That resource group then applies to all parts of the menu specification that are subordinate to the template and that do not specify a resource group of their own. Lower-level specifications shadow higher-level specifications.

To specify a resource group in a menu specification:

➔ Edit the gms-text-resource-group attribute of the relevant template object to name the resource group.

When gms-use-translations is true, every template object must have a resource group specified for it, either explicitly or by inheritance from a higher level of the menu specification. Thus every root template must specify a resource group.

The GMS Text Resources

To display the English gms-text-resources, click on the gms-text-resources icon on the GMS top level workspace. This is the local text resource for GMS. It contains symbols and translations for all help labels on the GMS palette and for all error messages. For more information about using local text resources see the *G2 Foundation Resources User's Guide*.

To view the gms-text resources:

1 Display the table for the English gms-text resources.

2 Set the value of the attribute gfr-file-location to a text string, giving the pathname of a file where the resource can be saved to or loaded from. It is not necessary to create the file, because GMS will do that.

3 Choose write resource to file on the menu for the English gms-text resources.

This will place a copy of the text resource in the file you designated.

4 Use any appropriate text editor to read the file.

Internationalizing Extensible Menus

You can use Peer Menu Connection Posts to extend a menu specification across modules, as described under Extending Menu Specifications across Modules. When you use this technique, resource group specifications apply just as they would if the whole menu specification existed in one module.

When a menu specification extends across module boundaries, the module that contains the higher-level part of the menu is typically proprietary, while the module that contains the extensions to the specification is not.

In such a case, the resource group that the proprietary part of the specification uses is typically proprietary also, and therefore cannot be modified. The module that defines the menu extensions will then need to define a resource group of its own to define the translations that it needs, and use that group to shadow the proprietary resource group used by the proprietary module.

Internationalizing Dynamic Menus

No special GFR considerations apply to dynamic menus. You can use internationalization in dynamically defined menus exactly as you can in statically defined menus.

Alternatively, you can optionally perform the translation yourself and specify labels as text directly. However, Gensym recommends that you use GFR when reusing dynamic menu templates in different menus.

If you are using a dynamic menu template *only once*, you can specify labels as text directly. The text for the labels is then unambiguous since there is a unique language associated with every window.

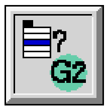
Configuring Global GMS Characteristics

Shows how to set properties that affect GMS as a whole such as automatic GMS startup, preserving the compiled resource on reset, specifying the maximum number of menu entries, suppressing global consistency checking, and internationalization.

Introduction

Most GMS behavior is defined in menu specifications. Some higher-level GMS properties do not apply to any particular menu, but to GMS overall. Those properties are called **global settings**. All GMS global settings are specified by attributes of a **Global Settings Object**.

The following figure shows a Global Settings Object:



This chapter describes all GMS global settings, and shows you how to use them to configure GMS.

To see online examples of the concepts covered in this chapter:

1. Load the *gmsdemo.kb* from the *kbs* directory.
2. Look at the Global Settings object that is part of each example in the GMS Demo.

Attributes of a Global Settings Object

The class-specific attributes of a Global Settings object are:

Attribute	Description
gms-priority	Sets the priority that GMS uses to select which Global Settings object to put into effect.
Allowable values:	Any nonnegative integer
Default value:	0
gms-initiate-automatically	Specifies whether GMS creates and displays menus automatically when G2 starts.
Allowable values:	true, false
Default value:	true
gms-keep-compiled-resource	Specifies whether GMS keeps compiled resources.
Allowable values:	true, false
Default value:	false
gms-use-translations	Specifies whether GMS uses GFR to internationalize menus.
Allowable values:	true, false
Default value:	false

Attribute	Description
gms-default-language	When gms-use-translations is true, specifies the language to translate to when the user has specified no other.
Allowable values:	Any G2 language for which GFR translations are defined.
Default value:	none
gms-maximum-entries-count	The largest number of menu entries that the GMS compiler should be prepared to compile.
Allowable values:	Any positive integer
Default value:	300
gms-check-for-global-consistency	Whether GMS should syntax-check menu specifications before compiling them.
Allowable values:	true, false
Default value:	true
gms-inactive-keys	Specifies a list of menu keys for those items that should not be displayed when the menu is displayed.
Allowable values:	any value -array
Default value:	a value-array

Managing Global Settings

GMS predefines one Global Settings object, called the **default global settings object**. This object specifies **default global settings** that are useful in many situations. All attributes of the default global settings object have the values shown in the previous table for Global Settings objects generally.

If the default global settings do not provide what you need, you can create your own Global Settings object and give its attributes the needed values.

To create a new Global Settings object:

➔ Clone a Global Settings object from the GMS palette to any workspace.

or

➔ Use the create action to create an instance of gms-global-settings programmatically.

To specify global settings:

➔ Edit attributes of the Global Settings object.

or

➔ Use the conclude action to specify the desired settings programmatically.

When more than one Global Settings object exists, GMS uses the one whose gms-priority attribute had the highest value when GMS compiled the menus currently in use. This object is called the **Current Global Settings object**, and the values of its attributes are the **current global settings**.

To indicate the desired Global Settings object to GMS:

➔ Give the object a gms-priority higher than that of any other Global Settings object.

Changes to global settings may require GMS to recompile menus. Therefore GMS does not put such changes into effect immediately. When GMS compiles menus, it locates the Global Settings object with the highest priority, puts its settings into effect, and compiles the menus in that context.

To put the highest-priority Global Settings object into effect:

➔ Compile menus as described under Compiling Menus.

If more than one Global Settings object with the highest priority exists, GMS uses one of the objects and ignores the others.

To change a global setting that is currently in effect:

1 Make the desired setting the value of the appropriate attribute of the current Global Settings object.

2 Compile menus as described under Compiling Menus.

Global Settings for Multiple Applications

When you load two or more applications that use GMS into G2, and each of them contains a Global Settings object, one of the objects must have a higher priority than any other, or GMS will be unable to determine which settings to use when it compiles menus.

The details of inter-application negotiation of global settings are not part of GMS: You can use any standard G2 technique that results in a unique highest-priority Global Settings object. Two common approaches are:

- Predefine a precedence order by giving different priorities to different Global Settings objects in the various applications.
- Dynamically select the desired global settings by changing Global Settings object priorities immediately before compiling menus.

Specifying Global Settings

The gms-initiate-automatically attribute controls GMS startup. The rest of this chapter shows you how to specify the various global settings to configure GMS. A reference to a particular setting refers to the value of the relevant attribute in the current Global Settings object.

Specifying Automatic GMS Startup

By default, GMS automatically compiles menus if necessary, assigns a handle to every G2 window, and builds menus for every G2 window, as soon as you start G2. GMS can also display an initial menu bar, as described under Designating an Initial Menu Bar.

Alternatively, you can configure GMS to do nothing when G2 starts, then manage the details of GMS startup explicitly using techniques described under Compiling and Building Menus.

To specify that GMS is to start automatically when G2 starts:

➔ Specify gms-initiate-automatically as true (the default).

To specify that GMS is to do nothing when G2 starts:

➔ Specify gms-initiate-automatically as false.

Preserving the Compiled Resource on Reset

The gms-keep-compiled-resource attribute controls whether GMS preserves or discards the compiled resources on reset or restart. By default, GMS discards the compiled resource whenever you reset G2. If GMS is configured to start automatically, as described under Specifying Automatic GMS Startup, it will recompile all menus next time you start G2, resulting in a new compiled resource.

If you know that discarding the compiled resource when you reset G2 would serve no purpose, you can configure GMS to preserve the compiled resource on reset.

To specify that GMS is to keep the compiled resource when G2 is reset:

➔ Specify gms-keep-compiled-resource as true.

To specify that GMS is to discard the compiled resource when G2 is reset:

➔ Specify gms-keep-compiled-resource as false (the default).

In order to keep the compiled resource, GMS transfers the object that implements the resource to the workspace that contains the Global Settings object that is currently in effect, then makes the object permanent.

GMS always invalidates all handles and deletes all menu instances when you reset G2, regardless of the value of gms-keep-compiled-resource. If GMS is configured to start automatically, it assigns new handles and rebuilds menus next time G2 starts.

Specifying the Maximum Number of Entries

When GMS compiles menus, it allocates space to hold various tables that it needs during compilation. If these tables are too small to hold all entries being compiled, GMS must obtain more space and rebuild all of the tables. Such rebuilding wastes time.

You can prevent GMS from having to rebuild tables during compilation by telling it how many menu entries to be prepared to compile. GMS then allocates sufficient table space in advance, removing any need for rebuilding.

To specify the maximum number of entries that GMS should expect to compile:

➔ Specify gms-maximum-entries-count to be an integer that gives the number of entries.

The number you specify should not be very much higher than necessary, because the table space needed for each menu entry is relatively large. Allocating many unnecessary table entries therefore wastes space.

Suppressing Global Consistency Checking

When GMS compiles menus, it checks for inconsistencies between the menu specifications and the global settings currently in effect. Such an inconsistency is called a **global inconsistency**.

For example, if gms-use-translations is true in the current Global Settings object (indicating that GMS uses GFR to translate textual information), all text information should be specified as GFR symbols rather than as text strings. The presence of a text string in a menu specification is therefore a global inconsistency.

By default, GMS checks all menu specifications for global inconsistencies before it begins compilation. If any are found, GMS signals an error and cancels compilation. By default, the error text is posted on the Operator Logbook. If you are sure that no global inconsistencies exist, you can suppress global consistency checking, resulting in faster compilation.

To suppress global consistency checking:

➔ Specify gms-check-for-global-consistency as false.

To receive global consistency checking:

➔ Specify gms-check-for-global-consistency as true (the default).

If a global inconsistency exists, and you have suppressed global inconsistency checking, the GMS compiler will signal an error when it encounters the inconsistency.

Specifying Internationalization

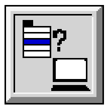
Several factors, including the global settings gms-use-translations and gms-default-language, control GMS internationalization. For information on internationalizing GMS menus, refer to Internationalizing GMS Menus.

Customizing the GMS Interface to the User

Shows how to set GMS properties for individual G2 users. These properties include: font and separator size, help display, text and background colors, navigation modes, menu blinking, initial menu, and menu language.

Introduction

Menu specifications and global settings establish GMS properties that are the same for all users. You can customize some GMS properties to be different for different users. GMS properties that you can customize on a per-user basis are called **user preferences.** All GMS user preferences are specified by attributes of a **User Preferences Object**. The following figure shows a Global Settings Object.



User preferences control the way GMS menus appear to the user. They are similar in many ways to global settings, and in some cases can override them. When several users log in to the same G2, GMS customizes the menus that each user sees as specified by the preferences applicable to that user.

This chapter describes all GMS user preferences, and shows you how to use them to customize GMS to meet the requirements of individual users. Refer to the GMS Demo called Configurations for illustrations of the concepts covered in this chapter.

To see online examples of the concepts covered in this chapter:

1 Load the `gmsdemo.kb` from the `kbs` directory.

2 Choose the demo called Configurations.

User Preferences Objects

The class-specific attributes of a User Preferences object are:

Attribute	Description
gms-priority	Sets the priority that GMS uses to select which User Preferences object to put into effect.
Allowable values:	Any nonnegative integer
Default value:	0
gms-applicability	Specifies the scope of the User Preferences object.
Allowable values:	by-registration, default
Default value:	by-registration

Attribute	Description
gms-language	When gms-use-translations is true in the current global settings, specifies the language to translate to.
Allowable values:	Any G2 language for which GFR translations are defined or the symbol window-or-system-default. Using window-or-system-default as the value for gms-language sets the language to the window-specific language designated for the window or the current-language specified in the Language Parameters system table.
Default value:	english
gms-font-size	Specifies the font size in which all menu entries are displayed.
Allowable values:	small, large, extra-large
Default value:	large
gms-normal-text-color	Specifies the color of the text in a menu entry that is neither selected nor disabled.
Allowable values:	Any G2 color
Default value:	black
gms-normal-background-color	Specifies the color of the background in a menu entry that is neither selected nor disabled.
Allowable values:	Any G2 color
Default value:	white

Attribute	Description
gms-highlighted-text-color	Specifies the color of the text in a selected menu entry.
Allowable values:	Any G2 color
Default value:	white
gms-highlighted-background-color	Specifies the color of the background in a selected menu entry.
Allowable values:	Any G2 color
Default value:	medium-blue
gms-disabled-text-color	Specifies the color of the text in a disabled menu entry.
Allowable values:	Any G2 color
Default value:	gray
gms-header-background-color	Specifies the color of the background in a popup menu header.
Allowable values:	Any G2 color
Default value:	black
gms-header-text-color	Specifies the color of the text in a popup menu header.
Allowable values:	Any G2 color
Default value:	light-goldenrod-yellow

Attribute	Description
gms-sticky-menus	Specifies that sticky menus are available.
Allowable values:	true, false
Default value:	true
gms-show-help-message	Sets GMS to display the Help label of any menu entry that has one whenever that entry becomes selected.
Allowable values:	true, false
Default value:	false
gms-help-message-fontsize	Specifies the font size in which Help labels are displayed in the help bar.
Allowable values:	small, large, extra-large
Default value:	large
gms-blinks-on-activation	Sets the number of times a chosen menu entry blinks before the effect of the choice begins.
Allowable values:	Any nonnegative integer
Default value:	0
gms-initial-menu-bar	A symbol that is the key of the menu bar that GMS should display when the user first logs in.
Allowable values:	Any symbol, true, false, or none
Default value:	none

Attribute	Description
gms-raise-menu-bar-interval	An integer specifying the number of seconds after which the menu bar is raised to the top of the workspace stack.
Allowable values:	Any nonnegative integer
Default value:	0
gms-separator-height	The height occupied by a separator in a transient menu.
Allowable values:	default, or an integer not less than 6
Default value:	default

Managing User Preferences

GMS predefines one User Preferences object, called the **system default preferences object**. This object specifies **system default preferences** that give good results when no other preferences apply to a user. All attributes of the system default preferences object have the values shown in the previous table for User Preferences objects generally.

Obtaining the Current User Preferences Object

The current User Preference object holds the setting that are presently in place in a given G2 window.

To obtain the current User Preferences object:

➔ gms-get-current-preference
    (*handle*: integer)
    -> *prefs*: class gms-preferences

Obtains the current User Preferences object for the G2 window referenced by *handle*.

Creating a User Preferences Object

If the system default preferences do not provide what you need, you can create your own User Preferences object and give its attributes the needed values.

To create a new User Preferences object:

➔ Clone a User Preferences object from the GMS palette to any workspace.

or

➔ Use the create action to create an instance of gms-preferences programmatically.

To specify user preferences:

➔ Edit attributes of the User Preferences object to specify the desired preferences.

or

➔ Use the conclude action to specify the desired preferences programmatically.

Registering Individual User Preferences

After you create and customize a User Preferences object, you can specify that the preferences it specifies apply to an individual user. GMS binds a user to a Preferences Object via the user's login name.

To specify that a User Preferences object applies to a particular user:

➔ Set the gms-applicability of the object to be by-registration (the default).

or

➔ gms-register-preferences-for-user
(*prefs*: class gms-preferences, *user*: symbol)

Specifies that the user preferences specified by *prefs* are in effect for the user whose login name is *user*.

Any number of users can register for the same User Preferences object. No user can be registered for more than one at a time. When a user already registered for one User Preferences object registers for another, GMS automatically unregisters the user from the previous object. You can also unregister a user from any User Preferences object.

To specify that no User Preferences object applies to a particular user:

➔ Call gms-register-preferences-for-user specifying *prefs* as gms-default-configuration and *user* as the login name of the user.

Specifying Generic User Preferences

You can also specify a **generic user preferences object**. This object shadows the system default preferences object, and supplies user preferences for all users who are not registered for any other preferences object.

To specify a generic User Preferences object:

➔ Set the gms-applicability of the object to be default.

Multiple objects can exist whose gms-applicability is default. You can use the gms-priority attribute to indicate the object GMS uses.

To indicate the User Preferences object to use when more than one exists:

➔ Give the object a gms-priority higher than that of any other User Preferences object.

When more than one object exists with the highest priority, GMS chooses one at random.

Searching for User Preferences

When more than one User Preferences object exists, GMS searches for the correct object to use. The order of the search is:

1. Look for a User Preferences object for which the current user is registered.
2. Look for a generic User Preferences object. If more than one exists, use the one with the highest priority, or one selected at random from among those with the highest priority.
3. Use the system default preferences object.

Changing the User Preferences Currently in Effect

You can change user preferences in various ways while the user is logged in, as described under Managing User Preferences.

Specifying User Preferences

You cannot change the system default preferences. The rest of this chapter shows you how to specify generic or individual user preferences. A reference to particular preference refers to the value of the relevant attribute in the applicable User Preferences object.

Specifying Font and Separator Size

You can specify the font in which GMS displays menu entries, and the height of any separators that appear between them.

To specify the font size used to display menu entries:

➔ Set gms-font-size to be small, large, or extra-large.

To specify the height of separators between menu entries:

➔ Set gms-separator-height to be either default or an integer not less than 6.

Specifying default tells GMS to make separators the same height as a menu entry. That height depends on the setting of gms-font-size. Specifying a positive integer tells GMS to give separators the indicated height in workspace units.

Caution If you set the gms-separator-height to be any value other than the default, and a separator exists on a panel that is a candidate for scrolling, the panel will not scroll.

Controlling Help Information

You can turn Help information on and off, and control the font in which GMS displays it.

To set GMS to display Help information:

1. Set gms-show-help-message to be true.
2. Set gms-priority to be a higher number than that of any of the other preference objects.

To set GMS to not display Help information:

➔ Set gms-show-help-message to be false (the default).

To specify the font size used to display Help information:

➔ Set gms-help-message-fontsize to be small, large, or extra-large.

Controlling Colors

You can control the text and background colors of menu entries and popup menu headers. Menu entry colors can vary depending on whether the entry is unselected, selected, or disabled. The following table lists each user preference that controls color, its default, and the color that it controls:

Preference	Default	Description
gms-normal-text-color	black	The color of the text in a menu entry that is neither selected nor disabled.
gms-normal-background-color	white	The color of the background in a menu entry that is neither selected nor disabled.

Preference	Default	Description
gms-highlighted-text-color	white	The color of the text in a selected menu entry.
gms-highlighted-background-color	medium-blue	The color of the background in a selected menu entry.
gms-disabled-text-color	gray	The color of the text in a disabled menu entry.
gms-header-text-color	light-goldenrod-yellow	The color of the text in a popup menu header.
gms-header-background-color	black	The color of the background in a popup menu header.

Specifying Menu Navigation Modes

GMS provides two menu navigation modes, called walking menus and sticky menus, as described under Choosing a Menu Entry. Walking menus are always available. Sticky menus are optional.

To specify that sticky menus are available:

➔ Set gms-sticky-menus to be true (the default).

To specify that sticky menus are not available:

➔ Set gms-sticky-menus to be false.

Controlling Menu Blinking

By default, GMS undisplays transient menus and invokes a leaf entry's callback procedure as soon as you choose the entry. You can set GMS to first blink a chosen menu entry one or more times.

To specify that GMS does not blink a chosen menu entry:

➔ Set gms-blinks-on-activation to be 0 (the default).

To specify that GMS blinks a chosen menu entry:

➔ Set gms-blinks-on-activation to be a positive integer indicating the number of blinks.

Designating an Initial Menu Bar

You can set GMS to display a particular menu bar to the user whenever G2 starts.

To specify that GMS does not display a menu bar on startup:

➔ Set gms-initial-menu-bar to be false or none (the default).

This setting does not prevent a subsequent API call from showing a menu bar. It specifies only that GMS does not do so automatically on startup.

To specify that GMS displays a designated menu bar on startup:

➔ Set gms-initial-menu-bar to be the key of the root object of the relevant menu specification.

If no menu bar has the specified key, GMS does not display any menu bar, just as if the value were false or none.

To specify that GMS displays a randomly selected menu bar on startup:

➔ Set gms-initial-menu-bar to be true.

This setting is appropriate when only one menu bar is defined, because only one application is loaded, and the user wants to use the application without needing any information about GMS internals.

Raising the Menu Bar to the Top

Displaying other workspaces in a window may obscure any menu bar in that particular window. Obscuring a menu bar may make it inaccessible. You may specify an interval, in seconds, at which GMS displays the menu bar on top of all other workspaces.

To specify that GMS should raise the menu bar to the top regularly:

➔ Set gms-menu-bar-raise-interval to be any non-negative integer.

If you specify 0 for this attribute, GMS does not perform this function.

Specifying Internationalization

Several factors, including the user preference gms-language, control GMS internationalization. For information on internationalizing GMS menus, see Internationalizing GMS Menus.

Specifying the Interface between GMS and G2

Shows how to configure GMS to work with and extend G2 capabilities such as accessing a compiled resource from G2, implementing keyboard accelerators, and handling mouse events for popup menus.

Introduction

Most aspects of the interface between GMS and G2 are handled automatically by GMS. However, some tasks require explicit programming to configure GMS to work with G2 as desired. Those tasks involve:

- Accessing GMS resources in code running outside of GMS.
- Implementing keyboard accelerators.
- Handling mouse events for popup menus.

This chapter shows you how to configure GMS to have the desired behavior in each of these areas.

To see online examples of the concepts covered in this chapter:

1 Load the `gmsdemo.kb` from the `kbs` directory.

2 Choose the Implementing Accelerators demo in the USING THE API section of the GMS demo KB.

  This demo shows how a developer can implement menu accelerators by using G2 configurations to map keystrokes to execute callbacks.

3 Choose the Advanced Popups demo in the POPUP MENUS section of the GMS demo KB.

  This demo shows how a developer can negotiate user mouse gestures on an object and still use GMS to post menus when appropriate.

Accessing GMS Resources from Outside GMS

In most cases:

- The information in the compiled resource is used either by GMS to display menus, or by callback procedures that need to determine what action to take.
- Callback procedures are invoked by GMS when the user chooses a leaf item in a GMS menu.

However, some situations require G2 code running outside of GMS to access the compiled resource and/or execute a callback procedure. This section shows you how to obtain such access.

Accessing the Compiled Resource

When a callback procedure needs information from a compiled resource, it obtains the information by using gms-get-activation-property, as described under Obtaining Menu Entry Attributes in an Activation Context. Code that runs outside of an activation context cannot use this call, because:

- The call obtains the requested information from an activation path that GMS passed to the callback procedure.
- An activation path ceases to be valid as soon as the callback procedure that received the path returns.

Therefore, GMS provides calls that can obtain information directly from the compiled resource, without the intermediary of an activation path.

To obtain information about properties from the compiled resource:

➔ gms-get-property
(*handle*: integer, *menu-index*: integer, *property-name*: symbol)
-> *property-value*: item-or-value

Returns the value of *property-name* in the menu or menu entry specified by *menu-index* in the compiled resource. The *handle* argument is provided for compatibility with future releases. Specify the value 0 (zero).

You can use gms-get-property to display any attribute that is in an object attribute list or table. You cannot use gms-get-property to access information about the dynamically created part of a dynamic menu specification. Such information exists only transiently, and is deleted as soon as GMS no longer needs it.

To obtain information about menu structure from the compiled resource use the procedure gms-return-submenu-entries as described in Obtaining All Cascade Menu Entries.

Executing a Callback Procedure

Using a keyboard accelerator requires G2 to execute the same callback procedure that GMS would execute if the user selected some GMS menu leaf entry. A button or a rule also may have reason to execute a callback procedure.

Such execution requires the callback procedure to receive the standard callback procedure arguments, which only GMS can construct. GMS therefore provides an API call that can execute any callback procedure and provide it with the arguments that it expects.

To execute a callback procedure from code running outside GMS:

➔ gms-execute-activation-callback
(*handle*: integer, *menu-index*: integer)

Executes the callback procedure of the menu entry referenced by *menu-index* in the context of the window referenced by *handle*. The *menu-index* must reference a menu entry defined in a static menu specification.

You cannot use gms-execute-activation-callback to execute the callback procedure of an entry that is defined in a Reusable Panel template, or in any submenu descended from such a template, because such an entry does not have a unique activation path unless the user has explicitly navigated to it with the mouse.

Implementing Keyboard Accelerators

Specifying a keyboard accelerator is largely a matter of writing G2 code. This code binds a keystroke to a call to gms-execute-activation-callback. For details:

- Examine the Implementing Accelerators demo in the USING THE API section of the GMS demo KB.
- Consult the *G2 Reference Manual* for additional information as needed.

To implement a keyboard accelerator:

1. Create a User Menu Choice whose action invokes gms-execute-activation-callback, either directly or by starting a procedure.
2. Specify the applicable-class of the User Menu Choice to be the class for which the accelerator is to work.
3. Configure the class referenced in applicable-class to bind the desired keystroke to the User Menu Choice.

Handling Mouse Events for Popup Menus

A GMS popup menu appears in response to an API call. The call typically occurs because the user clicked the mouse on an object. A popup menu disappears automatically when the user selects a leaf entry from the menu or a subsidiary cascade menu.

G2 includes a predefined popup menu capability that operates below the level of GMS. When you use GMS, you can configure objects to display GMS popups rather than G2 popups. GMS offers three ways to associate a GMS popup menu with an item:

- Use G2 item configurations to invoke the procedure gms-display-popup-menu when the user clicks on the item, as described under Using G2 Configurations to Display a Popup Menu.
- Include the mixin gms-popup-subscriber in the item's class definition, as described under Using gms-popup-subscriber to Display a Popup Menu.
- Configure GMS to handle mouse events, as described in this section.

In order to configure GMS to handle mouse events, you should:

- Understand G2 mouse tracking, as described in the *G2 Reference Manual*.
- Examine the Advanced Popups demo in the GMS demo KB.

Configuring GMS to Handle Mouse Events

Displaying, navigating, selecting, and choosing from a popup menu require GMS to handle mouse events obtained from G2. GMS uses the procedure gms-track-menu to handle mouse events.

G2 mouse tracking allows you to associate a procedure with an item such that all mouse events starting with mouse-down on that item and continuing until the next mouse-up are passed to invocations of that procedure.

You can write your own tracking procedure to handle mouse events. You may need to use GMS facilities to display popup menus while your tracking procedure is active. In order for GMS popup menus to function properly while the mouse is down and your tracking procedure is handling mouse events, your tracking procedure must forward every mouse event to GMS by calling gms-track-menu.

Managing a Popup from a Mouse Tracking Procedure

GMS provides two API calls for handling mouse events for popup menus:

- gms-manage-popup-menu
- gms-track-menu

The signature for gms-manage-popup-menu is:

gms-manage-popup-menu
(*handle*: integer, *menu-index*: integer, *X*: integer, *Y*: integer, *initiating-item*: item-or-value)

Displays the popup menu referenced by *menu-index* in the G2 window referenced by *handle* at window coordinates (*X*, *Y*). GMS passes *initiating-item* to any callback procedure that is invoked via the popup menu.

You can set *initiating-item* to be item that the user clicked to display the popup. If you don't need to pass this item to a callback procedure, specify a dummy value of false. Do not specify any other dummy value, or GMS will signal an error.

Caution Call gms-manage-popup-menu *only* in response to a mouse-down event, or while the mouse button is already down. If you call gms-manage-popup-menu in response to a mouse-up event, or while the mouse button is already up, GMS will be unable to detect the mouse state, and will be corrupted.
To recover from this error, reset G2 or call gms-reset, as described under Resetting GMS without Resetting G2.

After calling gms-manage-popup-menu, all further tracking events must be forwarded to gms-track-menu, until the mouse button is released. Failure to do so will result in GMS malfunctioning. For example, if you have a tracking procedure that handles mouse events on your item and want to display a popup menu, then it must be as follows:

```
my-tracking-procedure
     (Event: symbol, Win  : class g2-window, Itm1 : item-or-value,
     Itm2 : item-or-value, WinX : integer, WinY : integer, Time : integer,
     Kbd  : integer)

begin
      if (Event = the symbol START-TRACKING) then begin
          ...
          call gms-manage-popup-menu (Handle, PopupIndex, WinX, WinY, false);
          ...
          return
     end;
     ...
     call gms-track-menu (Event, Win, Itm1, Itm2, WinX, WinY, Time, Kbd);
end
```

The gms-track-menu procedure is an ordinary mouse tracking procedure, such as those described in the *G2 Reference Manual*.

To display a popup menu in response to a mouse-up event, or while the mouse button is already up, use gms-display-popup-menu as described under Using G2 Configurations to Display a Popup Menu.

16

Managing GMS Programmatically

Shows how to manage programmatically the internal operations of GMS. You can use the techniques described to manage menu building and compilation, change global settings, manage user preferences, and reset GMS without resetting G2.

Introduction

This chapter shows you how to manage programmatically the internal operations of GMS. You can use the techniques described to gain a finer-grained control over what GMS does than is possible using the techniques described in the previous chapters.

Use the information in this chapter if all of the following are true when you use GMS:

- GMS does not start automatically when G2 starts, as described under Specifying Automatic GMS Startup.
- Global settings change while GMS is in use.
- User preferences change while a given user is logged in.
- GMS needs to be reset at times when G2 is not reset.

If your use of GMS requires one or more of these not to be true, use the techniques described in this chapter to provide the GMS behavior that you need.

Compiling and Building Menus

When you reset G2, GMS always invalidates all window handles and deletes all menu instances. Unless gms-keep-compiled-resource is true in the current Global Settings object, GMS also deletes the compiled resource, if any.

When you start G2, and gms-initiate-automatically is true in the current Global Settings object, GMS does the following.

1 If no compiled resource exists, compile all menus as specified by the attributes of the current Global Settings object.

2 Assign a handle to every G2 window.

3 Build an instance of every menu in the compiled resource for every G2 window.

4 For every user whose current User Preferences object so specifies, display an initial menu bar in the user's G2 window.

When gms-initiate-automatically is false, and you start G2, GMS does nothing. The actions listed then occur only in response to explicit GMS API calls and/or choices in the G2 menus of GMS objects. The rest of this section describes those calls and choices, and their various effects.

Compiling Menus

You can use G2 menu choices and/or GMS API calls to compile any or all menu specifications. The API call to compile menus is:

gms-compile
(*specs*: item-or-value, *window*: class g2-window, *build?*: truth-value)

Compiles the menu specifications indicated by *specs*. Calling gms-compile:

- Deletes the compiled resource, if any.
- Invalidates any window handles.
- Deletes any menu instances.
- Creates a new compiled resource containing translations of the compiled menus.

Argument	Description
specs	A single menu specification, a list of menu specifications, or true, indicating compilation of all existing specifications.
window	A G2 window. If you specify *build?* as true, GMS builds menus for the window after compilation is complete. Otherwise give *window* the value gfr-default-window.
build?	If true, GMS calls gms-create-menu to build menus for *window* after the compilation is complete.

Note To save compiled resources created by GMS, you must save all modules in the KB. This ensures that the state of the GMS templates is consistent with the compiled resources.

You can also compile one or all menu specifications by choosing commands from a root template's G2 menu.

To compile a single menu specification:

➔ Choose compile tree from the G2 menu of the menu specification's root template.

or

➔ Call gms-compile giving the menu specification's root template as the value of *specs*.

To compile several menu specifications:

1 Create an item list that contains the specifications to be compiled.

2 Call gms-compile giving the item list as the value of *specs*.

To compile all menu specifications:

➔ Choose compile all from the G2 menu of any menu specification's root template.

or

➔ Call gms-compile giving *specs* the value true.

Building Menus

If you call gms-compile giving *build?* the value true, GMS automatically builds menus for the window designated by *window* after compilation is complete. You can also ask for menu building explicitly.

To build menus for a window:

➔ gms-create-menu
  (*window*: class g2-window)
  -> *handle*: integer

The call does the following:

- If no compiled resource exists, calls gms-compile to compile all menu specifications.
- Assigns a handle to *window*, replacing any previously assigned.
- Builds menus for the window. The call builds a menu for every menu translation in the compiled resource.

To delete all menu instances for a window:

➔ gms-delete-menu
  (*handle*: integer)
  -> *success?*: truth-value

If *handle* is a valid window handle, gms-delete-menu:

- Deletes all menu instances for the window referenced by *handle*.
- Makes *handle* invalid.
- Returns true.

If *handle* is not a valid window handle, gms-delete-menu:

- Returns false.
- Does not signal an error.

Changing Global Settings

When you compile menus, the current global settings become part of the compiled resource. Making changes to global settings therefore requires you to recompile all menu specifications.

To change a global setting that is currently in effect:

1 Set the attributes of the Global Settings object to specify the desired settings.

2 Compile menus as described under Compiling Menus.

Managing User Preferences

When GMS builds menus for a G2 window, it:

- Obtains the current User Preferences object for the user logged into that window.
- Includes in each menu instance the preferences specified by the User Preferences object.
- Associates the preferences object with the G2 window.

When GMS obtains and uses a User Preferences object, the identity of the user is significant only for determining which object to obtain. GMS thereafter manages user preferences by referencing the G2 window with which they are associated, *not* the user whose preferences they are.

To obtain the current User Preferences object:

➔ gms-get-current-preference
(*handle*: integer)
-> *prefs*: class gms-preferences

Obtains the current User Preferences object for the G2 window referenced by *handle*.

Changing User Preferences

When you change the current User Preferences object, or register the user for a different User preference's object, the change does not take effect immediately: the previous preferences still exist in the menu instances, and the same User Preferences object is still associated with the G2 window.

Rebuilding menus inherently puts the new preferences into effect. You can also put new preferences into effect by modifying the existing menu instances, avoiding the overhead of rebuilding them. The new preferences can be specified by changing the current User Preferences object, or putting by another into effect.

To put changes to the current User Preferences object into effect:

➔ gms-refresh-from-preferences
(*handle*: integer)

Updates the user preferences for the window referenced by *handle* as needed to reflect the current User Preferences object.

To put a different User Preferences object into effect:

➔ gms-switch-preferences
(*handle*: integer, *preferences*: class gms-user-preferences)

Makes *preferences* the User Preferences object for the window referenced by *handle*, putting the preferences that it specifies into effect, and associates the object with the window. The object does not have to meet the criteria by which GMS selects a current User Preferences object automatically.

Switching preferences does not replace the User Preferences object for which the user is currently registered. Next time GMS rebuilds menus, that object will again take effect.

Resetting GMS without Resetting G2

Sometimes it is convenient to reset GMS without resetting G2. For example, the various menu specifications, global preferences, and user preferences may have changed so much that completely initializing GMS is more convenient than putting the changes into effect one by one, yet G2 may contain transient information that must not be lost.

To reset GMS without resetting G2:

➔ gms-reset
(*delete-compiled-resource*: truth-value, *window*: class g2-window)

Resets GMS exactly as would happen if G2 were reset, but does not affect the rest of G2. If *delete-compiled-resource* is false, the reset does not affect the compiled resource (if any).

The *window* argument exists for compatibility with future releases of GMS. Specify the value gfr-default-window.

Appendixes

GMS API Reference

Describes all supported GMS API calls in alphabetical order.

gms-check-entry
(*handle*: integer, *menu-index*: integer)

Checks the menu entry referenced by *menu-index* in the window referenced by *handle*. If the entry is already checked, the call has no effect.

gms-check-radio-entry
(*handle:* integer, *CheckIndex*: integer, *RadioStartIndex*: integer, *RadioEndIndex*: integer)

Allows you to check any one entry of a specified group. It will uncheck any previously checked entry in the group.

CheckIndex, *RadioStartIndex*, and *RadioEndIndex* must reference menu entries on the same panel. The menu entry specified by *CheckIndex* will be checked and any previously checked entry between the last two will be unchecked.

gms-compile
(*specs*: item-or-value, *window*: class g2-window, *build?*: truth-value)

Compiles the menu specifications indicated by *specs*.

Argument	Description
specs	A single menu specification, a list of menu specifications, or true, indicating compilation of all existing specifications.
window	The g2-window from which this call is initiated. If there is none, then pass gfr-default-window.
build?	If true, GMS calls gms-create-menu to build menus for all windows after the compilation is complete.

gms-create-menu

(*window*: class g2-window)
-> *handle*: integer

- If no compiled resource exists, calls gms-compile to compile all menu specifications.
- Assigns a handle to *window*, replacing any handle previously assigned.
- Builds menus for the window. The call builds a menu for every menu translation in the compiled resource.

gms-delete-menu

(*handle*: integer)
-> *success?*: truth-value

If *handle* is a valid window handle, gms-delete-menu:

- Deletes all menu instances for the window referenced by *handle*.
- Makes *handle* invalid.
- Returns true.

If *handle* is not a valid window handle, gms-delete-menu:

- Returns false.
- Does not signal an error.

gms-disable-entry

(*handle*: integer, *menu-index*: integer)

Disables the menu entry referenced by *menu-index* in the window referenced by *handle*. If the entry is already disabled, the call has no effect.

gms-dismiss

(*handle*: integer)

Undisplays all transient menus on the window referenced by *handle*. If no transient menus are currently displayed in the window, gms-dismiss has no effect. Calling gms-dismiss has no effect on menu bars.

gms-display-menu-bar

(*handle*: integer, *menu-index*: integer)

Displays the menu bar referenced by *menu-index* at the top of the G2 window referenced by *handle*.

gms-display-popup-menu

(*handle*: integer, *menu-index*: integer, *x*: integer, *y*: integer, *initiating-item*: item-or-value)

Displays the popup menu referenced by *menu-index* in the G2 window referenced by *handle* at window coordinates (*x*, *y*). GMS passes *initiating-item* to any callback procedure that is invoked via the popup menu.

You can set *initiating-item* to be the item that the user clicked to display the popup. If you do not need to pass this item to a callback procedure, specify a dummy value of false. If you pass a dummy value other than false for *initiating-item*, GMS signals an error.

gms-display-popup-menu-at-last-event-location

(*handle*: integer, *menu-index*: integer, *initiating-item*: item-or-value)

Displays the popup menu referenced by *menu-index* in the G2 window referenced by *handle* at the coordinates of the last event in the window.

The gms-display-popup-menu-at-last-event-location determines the last event like the system procedures g2-last-input-event and g2-last-input-event-info. The gms-display-popup-menu-at-last-event-location procedure uses g2-last-input-event-info to determine the coordinates of the location for posting the menu.

GMS passes *initiating-item* to any callback procedure that is invoked via the popup menu.

You can set *initiating-item* to be the item that the user clicked to display the popup. If you do not need to pass this item to a callback procedure, specify a dummy value of false. If you pass a dummy value other than false for *initiating-item*, GMS signals an error.

gms-enable-entry

(*handle*: integer, *menu-index*: integer)

Enables the menu entry referenced by *menu-index* in the window referenced by *handle*. If the entry is already enabled, the call has no effect.

gms-entry-is-checked
(*handle*: integer, *menu-index*: integer)
-> *status*: truth-value

Returns true if the menu entry referenced by *menu-index* in the window referenced by *handle* is checked, and false otherwise.

gms-entry-is-disabled
(*handle*: integer, *menu-index*: integer)
-> *status*: truth-value

Returns true if the menu entry referenced by *menu-index* in the window referenced by *handle* is disabled, and false otherwise.

gms-entry-is-restricted
(*menu-index*: integer, *mode-restriction*: symbol)
-> *status*: truth-value

Returns true if the menu entry referenced by *menu-index* is currently restricted as specified by *mode-restriction*, and false otherwise.

gms-execute-activation-callback
(*handle*: integer, *menu-index*: integer)

Executes the callback procedure of the menu entry referenced by *menu-index* in the context of the window referenced by *handle*. The *menu-index* must reference a menu entry defined in a static menu specification. It must also be contained in the hierarchy of the menu bar being currently displayed on the window referenced by *handle*.

gms-get-activation-index
(*activation-info*: item-or-value, *menu-level*: integer)
-> *index*: integer

Returns the menu index of the menu entry at level *menu-level in the current activation*, or -1 of no such entry exists (because *menu-level* was out of range).

Note gms-get-activation-index is a function, not a procedure.

gms-get-activation-level
(*activation-info*: class gms-activation-info)
-> *level*: integer

Returns the menu level of the entry that invoked the callback procedure.

Note Gms-get-activation-level is a function, not a procedure.

gms-get-activation-property

(*activation-info*: item-or-value, *menu-index*: integer, *property-name*: symbol)
-> *property-value*: item-or-value

Returns the value of *property-name* in the menu or menu entry specified by *menu-index* in the *activation-info*.

gms-get-current-menu-bar-height

(*handle*: integer)

Returns the number of pixels in a menu bar on the G2 window. If no menu bar is visible, get-current-menu-bar-height returns 0. Use the current menu bar height to determine, for example, where to place a dialog just below the current menu bar without obscuring the menu bar.

gms-get-current-preference

(*handle*: integer)
-> *prefs*: class gms-preferences

Obtains the current User Preferences object for the G2 window referenced by *handle*.

gms-get-handle-for-window

(*window*: class g2-window)
-> *handle*: integer

Returns the handle associated with a G2 window, or -1 if the window has no handle.

gms-get-index-for-key

(*handle*: integer, *key*: value)
-> *menu-index*: integer

Returns the menu index of the entry referenced by *key*, or -1 if no such entry exists. If *key* is not unique, the call returns the index of a specified entry having that key.

If you are pulling down dynamic menus, your menu index does not reference different menu entries on different windows. *See also* gms-get-key-for-index.

gms-get-item-initiating-popup

(*activation-info*: item-or-value)
-> initiating-item: class item-or-value

Returns the item that the user clicked to display the popup menu, or the dummy value false if the popup was displayed by a call to gms-display-popup-menu or gms-manage-popup-menu that specified false as the *initiating-item*.

gms-get-key-for-index

(*handle*: integer, *index*: integer)
-> *key*: value

Returns the key of the entry referenced by *index*, or false if the entry does not exist or has no key.

If you are pulling down dynamic menus, your menu index will no longer reference different menu entries on different windows. *See also* gms-get-index-for-key.

gms-get-label

(*handle*: integer, *menu-index*: integer)
-> *label*: text

Returns the label of the entry referenced by *menu-index* in the window referenced by *handle*. If the entry has no label, the call returns an empty string.

gms-get-menu-bar-index

(*handle*: integer)
-> *menu-index*: integer

Returns the index of the menu bar currently on display in the G2 window referenced by handle, or -1 if no such menu bar exists.

gms-get-native-id-for-key

(*key*: symbol, *window*: g2-window)
-> *handle*: integer

Returns the integer handle for a native menu item. The *key* is typically the gms-user-key of a gms-template. When referring to native GMS menus, use this API procedure instead of gms-get-index-for-key to return a handle to the native menu item.

gms-get-property

(*handle*: integer, *menu-index*: integer, *property-name*: symbol)
-> *property-value*: value

Returns the value of *property-name* in the menu or menu entry specified by *menu-index* in the window referenced by *handle*. If the entry referenced by *menu-index* has been statically compiled, you can pass 0 for *handle*. You can use gms-get-property to access any attribute that is in an object attribute list or table.

gms-get-version()
-> *current-version*: text

Returns the current version of GMS, for example, 8.0 Rev. 0.

gms-get-window-for-handle

(*handle*: integer)
-> *<u>window</u>*: class g2-window

Returns the G2 window associated with a handle. If *handle* is not a valid handle, GMS signals an error.

gms-hide-menu-bar

(*handle*: integer)

Undisplays the menu bar currently displayed in the G2 window referenced by *handle*. If no menu bar is visible, the call has no effect.

gms-lock-menus

(*handle*: integer)

Locks the menus of the window referenced by *handle*. If the menus are already locked, the call has no effect.

gms-manage-popup-menu

(*handle*: integer, *menu-index*: integer, *X*: integer, *Y*: integer,
initiating-item: item-or-value)

Displays the popup menu referenced by *menu-index* in the G2 window referenced by *handle* at window coordinates (*X*, *Y*). GMS passes *initiating-item* to any callback procedure that is invoked via the popup menu.

You can set *initiating-item* to be item that the user clicked to display the popup. If you don't need to pass this item to a callback procedure, specify a dummy value of false. Do not specify any other dummy value, or GMS will signal an error.

gms-menu-is-locked

(*handle*: integer)
-> *<u>status</u>*: truth-value

Returns true if the menus of the window referenced by *handle* are locked, and false otherwise.

gms-redisplay-menu-bar

(*handle*: integer)

Redisplays the menu bar most recently displayed in the G2 window referenced by *handle*. If no menu bar was ever displayed in the window, the call has no effect.

gms-refresh-from-preferences

(*handle*: integer)

Updates the user preferences for the window referenced by *handle* as needed to reflect the current User Preferences object.

gms-register-preferences-for-user

(*prefs*: class gms-preferences, *user*: symbol)

Specifies that the user preferences specified by *prefs* are in effect for the user whose login name is *user*.

gms-reset

(*delete-compiled-resource*: truth-value, *window*: class g2-window)

Resets GMS exactly as would happen if G2 were reset, but does not affect the rest of G2. If *delete-compiled-resource* is false, the reset does not affect the compiled resource (if any).

The *window* argument exists for compatibility with future releases of GMS. Specify the value gfr-default-window.

gms-restrict-entry

(*menu-index*: integer, *mode-restriction*: symbol)

Restricts the menu entry referenced by *menu-index* as specified by *mode-restriction*. You can call gms-restrict-menu-entry as many times as needed to restrict all user modes for which restriction is desired.

gms-return-submenu-entries

(*handle*: integer, *menu-index*: integer, *index-list*: class integer-list)

Given the index of a cascade menu entry, obtains the indexes of all menu entries in the cascade menu.

Argument	Description
handle	The GMS handle of a G2 window.
menu-index	The index of a cascading menu entry.
index-list	An integer-list to which GMS appends the indexes of all entries in the cascade menu.

gms-set-label

(*handle*: integer, *menu-index*: integer, *label*: text)

Sets the label of the entry referenced by *menu-index* in the window referenced by *handle* to be *label*. The *label* must be a quoted string: call GFR to translate any GFR symbol. To specify no label, give an empty string.

gms-switch-preferences

(*handle*: integer, *preferences*: class gms-user-preferences)

Makes *preferences* the User Preferences object for the window referenced by *handle*, putting the preferences that it specifies into effect, and associates the object with the window. The object does not have to meet the criteria by which GMS selects a current User Preferences object automatically.

gms-uncheck-entry

(*handle*: integer, *menu-index*: integer)

Unchecks the menu entry referenced by *menu-index* in the window referenced by *handle*. If the entry is already unchecked, the call has no effect.

gms-unlock-menus

(*handle*: integer)

Unlocks the menus of the window referenced by *handle*. If the menus are already unlocked, the call has no effect.

gms-unrestrict-entry

(*menu-index*: integer, *mode-restriction*: symbol)

Unrestricts the menu entry referenced by *menu-index* as specified by *mode-restriction*. The effect is to undo the effect of the analogous call to gms-restrict-menu-entry. You can call gms-unrestrict-menu-entry as many times as needed to unrestrict all user modes for which restriction is not desired.

GMS Common Names and Formal Names

Gives the common name and the formal name of every GMS entity.

Common Name	Formal Name
Accelerator label	gms-accl-text
Built-in G2 Menu template	gms-builtin-template
Break Template	gms-break-template
Cascade Menu Template	gms-cascade-template
Change User Mode Template	gms-change-user-mode-template
Dynamic Cascade Template	gms-dynamic-cascade-template
Dynamic Popup Template	gms-dynamic-popup-template
Global Settings object	gms-global-settings
GMS Icon	gms-icon gms-highlightable-icon
Help label	gms-help-textl
Icon Configuration object	gms-icon-specification
Key	gms-user-key
Label	See the particular type of label.

Common Name	Formal Name
Leaf Entry Template	gms-choice-template
Menu Bar Template	gms-menu-bar-template
menu entry label	gms-label
Peer Menu Connection	gms-peer-menu-connection
Peer Menu Connection Post	gms-peer-menu-connection-post
Placeholder	gms-placeholder-stub
Popup Menu Template	gms-popup-menu-template
Right Justifier Template	gms-right-justifier
Reusable Panel Template	gms-reusable-panel-template
Separator Template	gms-separator-template
Show Workspace Template	gms-show-workspace-template
Switch Menu Bar Template	gms-switch-bar-template
Submenu Connection	gms-sub-menu-connection
Submenu Connection Post	gms-sub-menu-connection-post
SubPanel Container	gms-subpanel
User Preferences object	gms-preferences

Glossary

A

accelerator: A keystroke that has the same effect as choosing a menu entry: typing the keystroke executes the same user-defined procedure that choosing the menu entry executes.

accelerator label: A label that indicates a keystroke that has the same effect as choosing the menu entry. Examples: Control-O, Alt-S.

activation callback procedure: A user-defined procedure that executes when you choose a leaf entry.

activation context: The complete context within which a callback procedure was invoked.

B

break: A divider that formats a menu bar into multiple rows, and a transient menu into multiple columns.

Break Template: A template that appears between two entry templates to tell GMS to display the second entry at the beginning a new row (menu bar) or the top of a new column (transient menu).

C

callback arguments: The three standard arguments that GMS passes to every callback procedure.

cascade menu: A subsidiary menu that appears when you select a higher-level menu entry.

Cascade Menu Template: A template that defines a cascading entry.

cascading entry: A menu entry that displays a cascade menu when selected.

Change User Mode Template: A template that changes the user mode to a specified mode.

checked menu entry: A menu entry that has a check mark to the left of its entry label.

choose: To raise the mouse button over a selected menu entry. Choosing a leaf entry executes the associated callback procedure.

compiled resource: A repository in which GMS keeps all menu translations. At most one compiled resource exists at a given time.

constructor arguments: The first three arguments that GMS passes to a panel constructor.

current global settings: The global settings specified by the current Global Settings object.

current Global Settings object: The Global Settings object that is in effect when GMS compiles menus.

D

default global settings: The global settings specified by the current default Global Settings object.

default Global Settings object: A predefined Global Settings object that GMS uses when no other has been defined.

dialog entry: A leaf entry whose callback procedure displays a dialog.

disabled menu entry: A menu entry that can be selected, but is otherwise inactive.

display points: A pair of points, one on a G2 window, the other on a workspace, that specify where a Show Workspace Template displays a workspace.

dividers: Generic term for breaks, justifiers, and separators.

dynamic menu specification: A menu specification that defines a menu only partially, and includes templates that programmatically complete the menu definition just before the menu is displayed

dynamic menus: Generic term for dynamically defined popup and cascade menus.

Dynamic Cascade Template: A template similar to a Cascade Menu Template, with additional attributes that control dynamic menu construction.

Dynamic Popup Template: A root template similar to a Popup Menu Template, with additional attributes that control dynamic menu construction.

dynamic templates: Generic term for Dynamic Popup Templates and Dynamic Cascade Templates.

dynamically constructed templates: Templates constructed using the G2 `create` and `conclude` actions for use in a dynamic menu specification.

E

enabled menu entry: A fully functional menu entry, as distinct from a disabled menu entry.

Entry label: Text that describes what will happen if you choose a menu entry, specified using the gms-label attribute of an entry template. Examples: Open, Save As.

Entry templates: Generic term for Cascade Menu Templates and Leaf Entry Templates.

extensible menu specification: A menu specification that uses Peer Menu Connection Posts to provide optional hooks for extending the menu.

G

generic user preferences object: A User Preferences object that shadows the system default preferences object, and supplies user preferences for all users who are not registered for any other preferences object.

GFR symbol: A symbol that represents a text string. When GMS encounters a GFR symbol in a menu specification, it passes the symbol to GFR, which returns a string appropriate to the current language environment.

global inconsistency: An inconsistency between a menu specification and the global settings in effect when the specification is compiled.

global settings: GMS properties do not apply to any particular menu, but to GMS overall.

Global Settings object: An object that specifies a collection of global settings.

GMS icon: An icon that appears next to a menu entry label and denotes the menu entry's effect graphically.

GMS menus: Generic term for menu bars, popup menus, and cascade menus displayed by GMS

GMS Palette: A palette that contains master copies of all GMS template objects.

H

handle: An integer that GMS assigns to a G2 window to represent the window in GMS API calls.

header: A title block that appears above the panel of a popup menu and displays the menu label of the popup menu specification's root template. This label typically describes the menu's origin or purpose. Menu bars and cascade menus do not have headers.

Help bar: A bar that appears at the bottom of the G2 window to display Help labels as the user selects menu entries.

Help label: A label that provides information intended to help the user understand what a menu entry does.

highlightable icon: A GMS icon that can change colors depending on the selection status of the menu entry that contains the icon.

I

icon configuration object: An object that specifies non-default scaling and color attributes for a GMS icon.

icon family: A set of per-font icon classes. Each time GMS displays a menu entry that uses per-font icon classes, GMS uses the class that is correct for the font then in use.

initiating item: The item that the user clicked to display a popup menu.

J

justifier: A divider in a menu bar specification that causes all subsequent menu entries in the bar to be right justified. When a break follows a justifier in a menu bar, the new row is also right-justified.

K

key: An attribute (gms-user-key) of a root or entry template that identifies the menu or entry and/or associates information with it.

L

label: Any text string that forms part of a menu definition. Sometimes used as a convenient shorthand for the "menu label" of a root template or the "entry label" of an entry template.

leaf entry: A menu entry that, when chosen, executes a user-defined procedure.

Leaf Entry Template: A template that defines a leaf entry.

location attributes: Attributes of a Show Workspace Template that define its display points.

lock menus: To set GMS to discard all mouse input, preventing the user from selecting or choosing menu entries.

M

menu bar: A horizontal menu that extends along the top of a G2 window.

Menu Bar Template: A root template that defines a menu bar.

menu entry: A rectangular cell in a menu that can contain an entry label, a GMS icon, and an accelerator label. All of these components are optional.

menu index: An identifying integer assigned to a menu or menu entry by the GMS compiler. The compiler includes the index in the compiled resource, and makes it the value of the gms-index attribute of the relevant template.

menu instance: A menu that GMS builds for a particular G2 window from data in the compiled resource.

menu label: Text that identifies a menu as a whole, specified using the gms-label attribute of a root template.

menu level: An integer that indicates how deeply a menu is nested within higher-level menus.

menu specification: Generic term for a static menu specification or a dynamic menu specification.

menu translation: The compiled form of a menu specification, produced by the GMS compiler and stored in the compiled resource.

P

panel: The entries in a GMS menu, occupying either one or more rows (menu bar) or one or more columns (transient menu).

panel constructor: A procedure called when the user selects a dynamic menu entry, or when a preconstructed menu is compiled. The procedure constructs and returns a specification for a transient menu.

Peer Menu Connection: A connection that links two adjacent entry or divider templates in a menu.

Popup Menu: A freestanding menu that can appear anywhere in a G2 window in response to a mouse click.

Popup Menu Template: A template that defines a popup menu.

posting callback procedure: A procedure that GMS calls whenever a particular menu is displayed or hidden.

preconstructed menu: A menu defined by a panel constructor at compilation time rather than at display time.

preconstruction: The practice of constructing a dynamic menu once when its specification is compiled, rather than reconstructing it every time it is to be displayed.

R

resource group: A GMS translation dictionary.

restricted menu entry: An entry that becomes disabled whenever the G2 window that displays it is in a specified user mode or modes.

Reusable Panel Template: A template permits a cascade menu appear in more than one place in a menu specification.

Right Justifier Template: A template that appears between two entry templates in a menu bar specification to tell GMS to right-justify the entries that are specified after the template.

root template: A template that represents the menu as a whole. It does not correspond to any particular menu entry.

S

select: To depress the mouse button over an object, or to move the mouse over it with the mouse button already depressed. Selecting an enabled cascading entry displays its cascade menu.

selection callback procedure: A procedure that GMS calls whenever a particular menu entry is selected or unselected.

separator: A horizontal divider that separates the panel of a transient menu into two sections.

Separator Template: A template placed between two entry templates to tell GMS to draw a horizontal line in the menu between the two entries that they specify.

Show Workspace Template: A template that displays a workspace at a specified location.

snap grid: An invisible grid on a GMS workspace. When a workspace has a snap grid, GMS positions new and moved template objects to appear at vertices of the grid.

static menu specification: A menu specification that completely defines a menu in advance of its compilation and display.

Sticky menus: A menu navigation mode in which a cascade menu appears when a cascading menu entry is chosen.

Submenu Connection: Connects a root template to the first entry template in its menu, or a cascading entry template to the first entry template in its cascade menu.

SubPanel Container: A template that provides a subworkspace for holding part of a menu specification.

Switch Menu Bar Template: A predefined subclass of Dynamic Cascade Template that displays the labels of all compiled menu bars, and allows the user to choose which should be the current menu bar.

system default preferences: The user preferences specified by the current system default preferences object.

system default preferences object: A predefined User Preferences object that provides preferences for users to whom no other User Preferences object applies.

T

template list: A list of GMS templates passed to and populated by a panel constructor for use in specifying a dynamic menu.

template objects: Predefined GMS objects that appear on the GMS Palette and can be cloned for use in constructing menu specifications.

template: A template object.

transient menus: Generic term for popup menus and cascade menus.

U

unchecked menu entry: A menu entry that does not have a check mark to the left of its entry label.

unlock menus: To set GMS to again accept mouse input, allowing the user to select and choose menu entries.

unrestricted menu entry: An entry whose enablement is independent of the user mode of the G2 window that displays the entry.

User Preferences object: An object that specifies a collection of user preferences.

user preferences: GMS properties that can be customized on a per-user basis.

W

Walking menus: A menu navigation mode in which a cascade menu appears when an enabled cascading menu entry is selected, and disappears when that or any other entry is chosen.

Index