Telewindows2 Toolkit

Java Developer's Guide

Components and Core Classes

Version 1.2 Rev. 2

gensym

Telewindows2 Toolkit Java Developer's Guide, Components and Core Classes

May 2002

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

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

Gensym Corporation
52 Second Avenue
Burlington, MA 01803 USA
Telephone: (781) 265-7100
Fax: (781) 272-7101

Part Number: DOC069-122

Contents Summary

Contents

Chapter 12 Using Workspace View Components *(continued)*

Chapter 13 Customizing Popups for Selected Items 181

Chapter 15 Using Dialog Components *(continued)*

Chapter 16 Launching Custom Item Properties Dialogs 305

Chapter 16 Launching Custom Item Properties Dialogs *(continued)*

Chapter 17 Customizing Automatically Generated Dialogs 321

Chapter 18 Launching General Dialogs 337

Preface

Describes this guide and the conventions that it uses.

gensym

About this Guide

This guide describes the Telewindows2 (TW2) Toolkit components and core classes for creating Java-based, native, client, user interfaces for G2 applications.

The TW2 Toolkit components are packaged in a JAR file, which you can load into or a JavaBeans-compliant visual programming environment, such as Symantec Visual Café or Borland J Builder.

If you are working strictly in a visual programming environment, you do not need to use any of the core classes that this guide documents. For such users, these core classes are documented to provide background for using the visual components.

If you are working in a Java development environment, you use the core classes to perform basic functionality, such as connecting to G2, creating a secure login, and handling associated events. Depending on the amount of customization you wish to do, you might also use some of the core classes to customize the way in which TW2 Toolkit represents and displays G2 KB workspaces, or to create and launch dialog resources that you create in a visual Java programming environment.

Audience

This guide is written primarily for the UI developer, who creates client user interfaces for G2 applications. UI developers use and configure TW2 Toolkit and third-party vendor components in Java. Because the TW2 Toolkit components are JavaBeans compliant, UI developers can also work in a JavaBeans-compliant visual programming environment.

Additionally, this guide addresses to some extent the component developer, who needs to use and extend TW2 Toolkit components and core classes to create applications that access and manipulate KB data and knowledge.

A Note About the API

The techniques by which Telewindows2 Toolkit implements its capabilities are subject to change at any time without notice or explanation, and are expected to change as the toolkit evolves. These techniques, and any changes to them, will not be described in any documentation.

Therefore, it is essential that you use TW2 Toolkit exclusively through its API as described here and in the API documentation. Any methods or classes that are not included in the API are subject to change without notice. Any code that calls undocumented methods may cease to work in newer versions.

Conventions

Typographic

Convention Examples	Description
g2-window, g2-window-1, gfr-top-level, sys-mod	G2 class names, instance names, workspace names, and module names
history-keeping-spec, temperature	G2 attribute names
true, 1.234, ok, “Burlington, MA”	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

Convention Examples	Description
conclude that the x of y ...	Text of G2 procedures, methods, functions, formulas, and expressions
new-argument	User-specified values in syntax descriptions
text-string	Return values of G2 procedures and methods in syntax descriptions
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 top-level menu choices and native popup menu choices
workspace	Glossary terms
`c:\Program Files\Gensym\g2`	Windows pathnames
`/usr/gensym/g2/kbs`	UNIX pathnames
`spreadsh.kb`	File names
`g2 -kb top.kb`	Operating system commands
`public void main()` `gsi_start`	Java, C and all other external code

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

Procedure Signatures

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

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

Related Documentation

Telewindows2 Toolkit

Online Files

The following document is available in the following directory, depending on your platform:

NT: `%SEQUOIA_HOME%\readme-tw2.html`

UNIX: `$SEQUOIA_HOME/readme-tw2.html`

Java Developer's Guides

The online files are located in this directory, by default, depending on your platform:

NT: `c:\Program Files\Gensym\g2-6.1\doc\tw2\Java\docs\guides\`

UNIX: `/usr/gensym/g2-6.1/doc/tw2/Java/docs/guides/`

- *Telewindows2 Toolkit Release Notes*
- *Telewindows2 Toolkit Java Developer's Guide: Components and Core Classes*
- *Telewindows2 Toolkit Java Developer's Guide: Application Classes*
- *Telewindows2 Toolkit Java Demos Guide*
- *BeanXporter User's Guide*

G2 JavaLink

Online Files

The following document is available in the following directory, depending on your platform:

NT: `%JAVALINK_HOME%\readme-javalink.html`

UNIX: `$JAVALINK_HOME/readme-javalink.html`

User's Guides

The online files are located in this directory, depending on your platform:

NT: `c:\Program Files\Gensym\g2-6.1\doc\javalink\docs\guides\`

UNIX: `/usr/gensym/g2-6.1/doc/javalink/docs/guides/`

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

Java Reference Material

- JDK 1.3 documentation set *
- *The Java Language Specification* (Gosling, Joy, Steele. Addison Wesley) *
- *The Java Bean Specification V1.0 **
- *These and other Java documents can be downloaded from Sun Microsystems' Java web site at `http://www.javasoft.com`.

G2 Core Technology

- *G2 Bundle Release Notes*
- *Getting Started with G2 Tutorials*
- *G2 Reference Manual, Volumes I and II*
- *G2 Developer's Guide*
- *G2 System Procedures Reference Manual*
- *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 Developer's Interface User's Guide*
- *G2 Menu System User's Guide*
- *G2 XL Spreadsheet User's Guide*
- *G2 Dynamic Displays User's Guide*
- *G2 GUIDE User's Guide*
- *G2 GUIDE/UIL Procedures Reference Manual*
- *G2 OnLine Documentation Developer's Guide*
- *G2 OnLine Documentation User's Guide*

G2 Diagnostic Assistant

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

Bridges and External Systems

- *G2 WebLink User's Guide*
- *G2 ActiveXLink User's Guide*
- *G2 CORBALink User's Guide*
- *G2 OPCLink User's Guide*
- *G2-Oracle Bridge Release Notes*
- *G2-Sybase Bridge Release Notes*
- *G2-ODBC Bridge Release Notes*
- *G2 Database Bridge User's Guide*

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 `http://www.gensym-support.com`.

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

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

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

➔ Use the following numbers and addresses:

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

Part I

Introduction

1

Overview of Telewindows2 Toolkit

Presents a brief overview of Telewindows2 Toolkit components and core classes, and describes how to use them.

Introduction

Welcome to Gensym's Telewindows2 (TW2) Toolkit components and core classes for Java programmers. The TW2 Toolkit components comprise a set of JavaBeans-compliant Graphical User Interface (GUI) tools for G2. These components are:

- Modular, reusable, and specifically designed for creating a G2 user interface.
- Not limited to a single environment and can be designed to launch from within a Web browser, a Windows application, or a Java application.

The TW2 Toolkit components documented in this guide consist of:

- **Connectivity components** for connecting to G2 and establishing a secure login session, if required.
- **Workspace view components** for displaying one or more KB workspaces in a client.
- **Data-aware controls** for viewing and editing attributes of G2 items through a dialog, including a text field, a check box, a radio box, a pulldown list, a list box, and an invisible control for representing G2 structures.
- **Dialog command components** for controlling dialog actions when, for example, the user clicks the OK, Apply, or Cancel button.

In addition to these components, this guide documents several core classes:

- Connectivity classes, which are Java classes that underlie the connectivity components for connecting to G2, providing connection and login information to the connection, and handling connection and KB events.
- Classes and interfaces for handling dialog resources that you create using TW2 Toolkit components.

To use advanced features of connectivity components, you need to understand the connectivity classes. If you are programming in a Java environment, you use connectivity classes rather than the component equivalents to create a G2 connection.

If you use TW2 Toolkit components to create custom properties dialogs for G2 items, you must register the dialog resource with a dialog manager, from either G2 or your Java application. If you are programming in a Java environment, you can also implement your own classes for reading and launching your own types of dialog resources.

What Can You Do with Telewindows2 Toolkit Components?

Using TW2 Toolkit components and core classes, you can create a range of G2 client user interfaces to suit your needs. You can implement, for example:

- A small applet to perform simple monitoring tasks through a Web browser, including displays of important G2 object values in a read-only format.
- A fully functional client application that controls your entire manufacturing plant.

Developers can combine TW2 Toolkit components and core classes to create a client application capable of connecting to a G2 process, and obtaining data from and updating data to a G2 knowledge base (KB). Such a client application can access and represent virtually all of a KB's knowledge, including complete views of KB workspaces capable of displaying graphically complex items, such as trend

charts and animated icons. Java programmers can also build their own TW2 Toolkit components.

This document presents the TW2 Toolkit components and core classes available to a Java programmer, and describes how to use them to create a native client GUI for G2.

For information on using TW2 Toolkit classes that support building applications that connect to one or more G2 processes, refer to the *Telewindows2 Toolkit Java Developer's Guide: Application Classes*.

Who Uses Telewindows2 Toolkit Components?

Most developers use TW2 Toolkit to create client user interfaces for G2 applications. This document refers to such users as **UI developers** and primarily addresses their needs.

Other developers, referred to as **component developers**, might not be building a complete user interface. Instead, these users need to use and extend TW2 Toolkit components and core classes to create applications that access and manipulate KB data and knowledge.

UI developers use and configure TW2 Toolkit and third-party vendor components within any Java programming environment. Because the TW2 Toolkit components are packaged as Java Beans, UI developers can work with these beans in a **JavaBeans-compliant visual programming environment**. They do this by loading JAR files of Java Beans, editing the properties of those beans through a properties table, and hooking up event triggers from one component to target methods in another component.

A number of **integrated development environments (IDEs)**, such as Symantec Visual Café or Borland J Builder, provide visual Java programming environments within a fully integrated environment for developing Java applications.

This guide does not include examples of working with third-party vendor components, although it does discuss the requirements for using TW2 Toolkit components within visual Java programming environments.

The UI developer requires a working knowledge of Java, its JavaBeans specification capabilities, AWT, Swing, and the Telewindows2 Toolkit API.

This guide is written for Java programmers familiar with software component design and technology; it does not attempt to describe the theory or purpose of creating and developing Java Bean components. Numerous well-written documents already exist to fulfill that requirement. See the Java Sun website at `www.java.sun.com` for references. Instead, this guide presents the major TW2 Toolkit components, and other relevant classes and facilities, as a developer needs to understand them to create G2 client applications.

How Do You Work with Telewindows2 Toolkit Components and Core Classes?

As a UI developer, you work with Telewindows2 Toolkit components and core classes in one of two ways:

- By loading the TW2 Toolkit components into a JavaBeans-compliant visual programming environment and using these and third-party vendor controls to develop dialog classes and G2 client applications.
- By writing Java programs, in a third-party IDE or in pure Java, that use TW2 Toolkit components and core classes.

You can also use tools provided by G2 JavaLink to create a visual JavaBeans interface for any G2 system-defined or user-defined class. You can then use these components in a JavaBeans-compliant visual programming environment, a third-party IDE, or pure Java.

TW2 Toolkit allows you to launch and view source code for:

- A number of demonstrations that illustrate the use of TW2 Toolkit components and core classes in applets, in a Web browser, in Java containers, and in TW2 Toolkit applications.
- The TW2 Toolkit default application shell, which allows you to access a G2 KB through a native client user interface and edit textual attributes of items by using a native Text Editor.

While the TW2 Toolkit default application shell uses some TW2 Toolkit components and core classes, it is primarily an example of using TW2 Toolkit application classes; thus, it is described fully in the *Telewindows2 Toolkit Java Developer's Guide: Application Classes*.

Because the Text Editor is a feature of editing items on workspace views, it is described fully in this guide.

Using Telewindows2 Toolkit Components

The classes that comprise the Telewindows2 Toolkit components are organized into packages. This table categories these classes and packages and provides a brief description of each class:

Package/Category/Component	Description
com.gensym.controls	
Connectivity Components	
ItemRetriever	Connects to G2 and obtains an item.
TwConnector	Connects to G2.
com.gensym.controls	
Non-Visual Controls	
ItemProxy	Holds a G2 item for editing or viewing its data.
StructureMUX	An invisible control that represents attributes that are a type of G2 structure.
com.gensym.jcontrols	
Data-Aware Controls	
G2Label	Supplies a textual descriptor for fields, headings, or messages.
G2TextField	Supplies a user type-in field.
G2Checkbox	Turns an option on and off.
G2Radiobox	Represents multiple options from which the user can choose one.
G2ComboBox	Presents a list of options in a drop down list from which a user can choose one.
G2Listbox	Represents a single selection from a list or a collection of multiple values in a single control.

Package/Category/Component	Description
com.gensym.controls	
Dialog Controls	
G2Button	Propagates an action.
DialogCommand	Informs dialog listeners of dialog events such as applying dialog edits, closing the dialog, and canceling the dialog.
com.gensym.wksp	
Workspace View Components	
ScalableWorkspaceView	Displays a single KB workspace that is scalable.
MultipleWorkspaceView	Displays any of several KB workspaces.
MultipleWorkspacePanel	Displays a multiple workspace view that has scrollbars.
WorkspaceView	Displays a single KB workspace. **Note**: This component exists for backward compatibility for Active X developers only.

JAR Files

The components that TW2 Toolkit supplies are packaged in Java ARchive (JAR) files. TW2 Toolkit includes these JAR files:

JAR File	Description
`sequoia.jar`	All the classes necessary to support TW2 Toolkit functionality, for example, `TwGateway` and `ScalableWorkspaceView`. It also contains TW2 Toolkit components for creating dialogs that communicate with G2 items and KB workspaces, including connectivity components, data-aware controls, dialog controls, and workspace view components. You load this JAR file into a JavaBeans-compliant visual programming environment, such as Symantec Visual Café or Borland J Builder. **Note:** This JAR file contains both AWT and Swing versions of these components.
`coreui.jar`	Telewindows2 Toolkit UI classes used for creating applications, including commands, command-aware containers such as menus, toolbars, and palettes, MDI application classes, and dialogs.
`ax2jbeans.jar`	Classes in support of packaging ActiveX components as Java Beans, using the G2 BeanXporter. You do not load this JAR file.

Data-Aware Components

The `com.gensym.jcontrols` package contains a number of visual controls used for viewing and editing attributes of G2 items through a dialog. It also contains an invisible component that lets you represent G2 structures. These components are called **data-aware components**, which means they:

- Can perform automatic updates to and from G2.
- Transparently handle G2 data types.

The com.gensym.jcontrols package provides two categories of data-aware components:

- **Scalar controls**, which you use to view and edit "atomic" data types, such as integers, floats, text, symbols, and truth values:
  - G2Label
  - G2TextField
  - G2Checkbox
  - G2RadioBox
  - G2ComboBox
  - G2ListBox
- **Aggregate controls**, which you use to edit data structures, such as G2 lists, sequences, and structures. Because these data types have no simple analog in the AWT component set, you must use one of the scalar controls to edit subparts of the data structure. The aggregate controls are:
  - G2ListBox — A dialog control for viewing and editing G2 sequence data types, which holds a collection of like elements in a collection list, which can grow or shrink, and in which the user can edit the currently selected element through some other dialog control.
  - StructureMUX — An invisible component used for viewing and editing G2 structure data types, which allow you to route the attributes of a structure into separate dialog components for editing and viewing.

Note that by specifying a property of the G2ListBox control, you can use it to edit atomic data types or data structures.

A more accurate, but painfully long name for StructureMUX would be G2StructureMultiplexerDemultiplexer.

Change and Update Events

You use the data-aware components in conjunction with an ItemProxy to handle updates to and from G2 by means of two events:

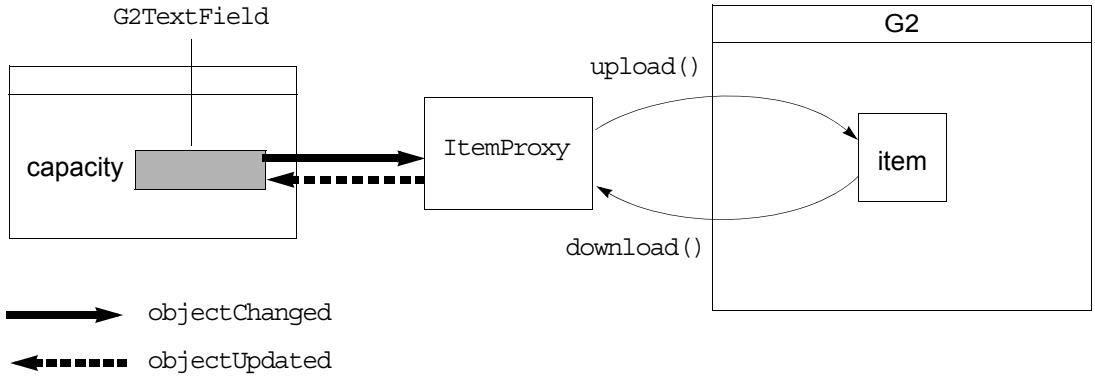
- ObjectChangeEvent — Occurs when a scalar control in a dialog changes. The ItemProxy component implements ObjectChangeListener, as do the aggregate controls such as G2ListBox, which means they are notified when atomic data in a dialog changes.
- ObjectUpdateEvent — Occurs when an item in the G2 server gets updated or when an aggregate control changes. All scalar and aggregate controls implement ObjectUpdateListener, which means they receive notification via an ItemProxy when data in the G2 server gets updated.

Thus, ObjectUpdateEvents and ObjectChangeEvents flow between data-aware components and the ItemProxy component, as follows:

- An ItemProxy generates ObjectUpdateEvents for downloading changes from G2 to the data-aware controls.
- An ItemProxy receives ObjectChangeEvents for uploading changes from the data-aware controls back into G2.

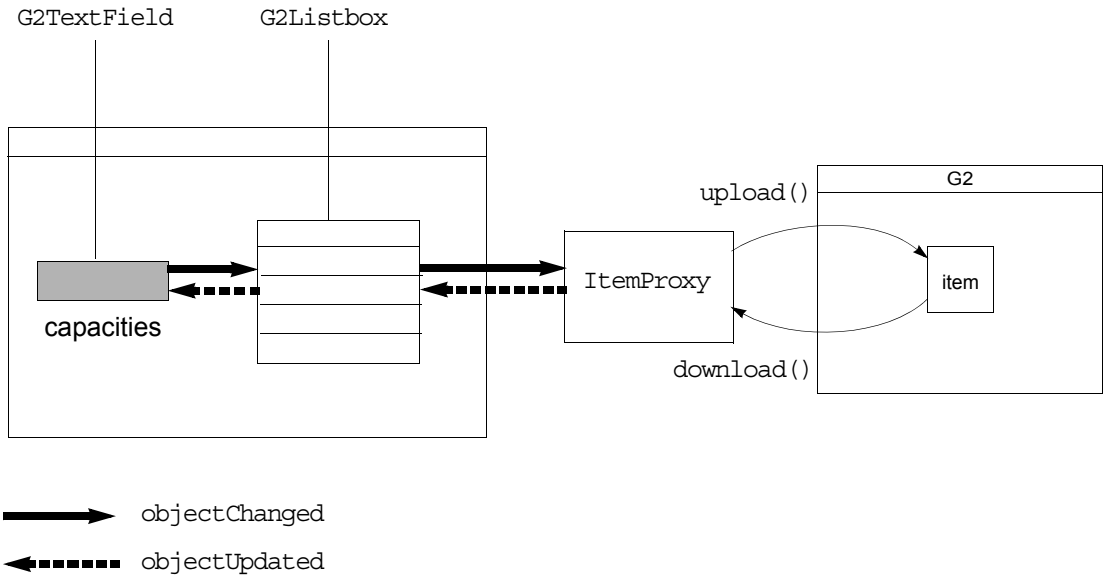
You can explicitly invoke methods on ItemProxy to initiate the upload or download, or you can rely on event handling or state changes to cause uploading and downloading to occur automatically.

By connecting these events between a data-aware control and an ItemProxy, you can create dialogs that change the attributes of G2 items and get updated when the value of an attribute of a G2 item changes, as this figure illustrates:



When you use data-aware controls to edit data structures, ObjectUpdateEvents flow from aggregate controls to scalar controls. They are usually initiated when a dialog is launched or when a dialog receives an update from G2. ObjectChangeEvents flow in the opposite direction, from scalar controls to aggregate controls, and they are usually initiated when the end user edits a scalar control.

This figure shows how these events flow between a `G2TextField` and a `G2Listbox` used as an aggregate control:



Data Type Conversion

Data-aware components handle data type conversion transparently. For example, if a `G2TextField` component represents an item attribute whose value is a G2 float, no special coding is required to convert the text value that a user enters.

Each of the data-aware components includes a `fieldType` property, whose value can be any of the following G2 data types:

- symbol
- text
- integer
- float
- truthvalue
- structure
- sequence

Internally, this property is an instance of a `com.gensym.controls.FieldType` object, which handles the conversion of the G2 data types that G2 JavaLink supplies into the data type that the underlying Java component requires.

For example, if your application uses a `G2TextField` to edit a G2 item attribute whose value is an integer, the `FieldType` object translates the integer into a

`java.lang.String` before calling its `setText` method. Conversely, after a user changes the value in the `G2TextField`, the `FieldType` object translates the string that the user entered back into a `java.lang.Integer` value.

In addition, the `G2TextField` component provides functionality that checks whether the user input matches the field type and allows you to specify whether an empty field should be interpreted as `null`. The ability to interpret empty text fields lets you to distinguish between an attribute that "does not exist," from the perspective of the G2 compiler, and an attribute whose value is none in the G2 attribute table.

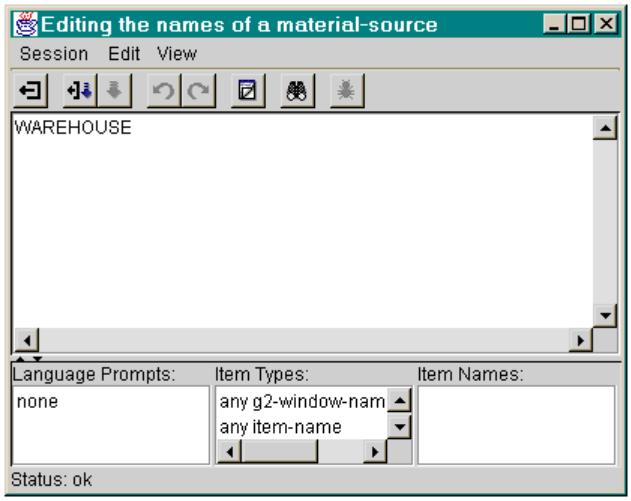
Workspace View Components

While the workspace view components are not data aware in the same sense that the dialog controls are data aware, they also perform automatic updates to and from G2. In particular, once a workspace view is displayed in an application shell, all of its items automatically update G2 of any changes in the client, and G2 automatically updates the items of any changes in the server.

TW2 Toolkit accomplishes this by means of methods that update changes in G2 data into the client UI, and conclude updates from the client UI back into G2. Your TW2 application can subscribe to these updates through the components in the `com.gensym.controls` package.

Text Editor

TW2 Toolkit includes its own text editor, which the `ScalableWorkspaceView` component invokes any time you edit a textual attribute of an item in a workspace view:



Chapter 11, "Using the Text Editor" on page 147 describes the TW2 Toolkit text editor.

Internationalization

You can localize all text displayed within TW2 Toolkit components, such as dialog text. TW2 Toolkit supports localization through the use of text key variables. You store the textual representations of the keys in a resource file, which you can localize, as necessary.

Accessor Methods

The get and set accessor methods for the TW2 Toolkit components are described fully in the API documentation and are presented in simple tables in the relevant chapters throughout this document.

Using Telewindows2 Toolkit Core Classes

In addition to the components, this guide describes the following core classes in the following categories and packages, for use in Java applications:

Package/Category/Class	Description
com.gensym.ntw	
Connectivity Classes	
TwGateway	Allows you to connect a client application to G2 to gain access to complete TW2 Toolkit functionality.
TwConnectionInfo	Sets the properties that a G2 connection request requires.
TwConnectionListener	When registered as a listener, receives notification of connection events.
WorkspaceShowingListener	When registered as a listener, receives notification of G2 programmatic show and hide workspace events.
LoginRequest	Sets the properties that a login session to a G2 connection requires.
com.gensym.ntw	
KB Classes	
KbModuleListener	When registered as a listener, receives notification of G2 module events.

Package/Category/Class	Description
KbMessageListener	When registered as a listener, receives notification of G2 Message Board and Operator Logbook message events.
com.gensym.dlgruntime	
Dialog Resource Classes	
DialogManager	Handles the registration of dialog resources as properties dialogs of G2 items, plus launches and reads those dialogs.
DefaultDialogReader	Reads dialog resources.
DefaultDialogLauncher	Launches dialog resources.
DialogReader	An interface that you can implement to read your own type of dialog resources.
DialogLauncher	An interface that you can implement to launch dialog resources in any type of container.
DialogResource	The type of object that you register using a DialogManager, which takes the dialog resource, a DialogReader, and a DialogLauncher.
com.gensym.gcg	
Generated Dialog Classes	
GeneratedDialogFactory	An interface responsible for automatically generating item properties dialogs.
DefaultGeneratedDialogFactory	A default implementation of the GeneratedDialogFactory interface.
AttributeLabel	The label associated with an item attribute.

Package/Category/Class	Description
AttributeInfo	An object that encapsulates all the information you might need about an attribute, including its name, type, defining class, whether it is system defined, and so on.
G2AttributeEditor	An object that encapsulates the group name, label, editor, item, and attribute name.
G2AttributeGroup	A list of all the editors that belong to that group, plus the group name.
G2ReadOnlyTextArea	A JScrollPane used for displaying complex attributes such as sequences and structures or attributes with a grammar or attributes that display items.
SubDialogLauncher	Launches a subdialog for editing an attribute of an item that is an object.
G2TextArea	An implementation of SubDialogLauncher that launches the text editor for editing attributes with a grammar or a properties dialog for editing the attributes of the subobject.
G2ColorField	A com.gensym.jcontrols. G2TextField whose background is a color, used for editing color attributes.

Using Java-Based Visual Programming Environments

You can use JavaBeans-compliant visual programming environments to create custom dialogs for editing the attributes of items in the client. Dialogs that you create in visual Java programming environments are typically Java classes.

The technique you use for registering custom dialog classes created in a visual Java programming environment is similar to the technique you use for registering dialog resources, except that you use a slightly different version of the registration method. In addition, when creating your Java dialog class, you must implement particular TW2 Toolkit classes to keep track of the G2 item the dialog is editing.

Telewindows2 Toolkit has been tested with the following two Java-based IDEs:

- Symantec Visual Café
- Borland J Builder

Creating Components from G2 Classes

Telewindows2 Toolkit through G2 JavaLink supplies a component interface to all system-defined G2 classes. This means you can call the accessor methods and class methods on system-defined items from any Java application. To use these components within a Java-based visual programming environment, you use the G2 Bean Builder to create a visual Java Bean to represent the item.

You can also create a component interface to any user-defined G2 class. If all you need is a Java class representation of the item, you can use the G2 Download Interfaces wizard tool provided with G2 JavaLink. If you need to use the item in a visual environment, again, you use the G2 Bean Builder to create a visual bean to represent your item.

Once the component is available, you can call a method on the class to get a handle on the item. In a pure Java development environment, you cast the result to the appropriate G2 JavaLink class.

To download data for the instance, you register the item as a listener for item events. Once the instance is downloaded to the client, getting a value for the item is a local call on the client. Setting a value requires a remote call to G2, unless you perform a batch operation. For more information, see the *G2 JavaLink User's Guide*.

Components that you create using the G2 Bean Builder define only the class attributes of the current class; they do not define inherited attributes.

The G2 Bean Builder makes it easier to use Java Beans components that represent system-defined and user-defined G2 classes in other graphical development environments. For more information about this tool, see the *G2 Bean Builder User's Guide*.

Using Telewindows2 Toolkit Demonstrations for Java

Telewindows2 Toolkit includes numerous demonstrations illustrating various functionality for Java programmers. These demos show how to use Java Beans components, Java UI components, and Java application classes to build applets and applications that connect to a G2 server, display workspace views, and manipulate data.

These demos are located in this directory, depending on your platform:

NT: `%SEQUOIA_HOME%\classes\com\gensym\demos`

UNIX: `$SEQUOIA_HOME/classes/com/gensym/demos`

To run the demos, you must either:

- Place G2 as the first G2 in your `PATH` environment variable.
- Define the `SEQUOIA_G2` environment variable to point to this version of G2.

A number of the demos make use of TW2 Toolkit components exclusively. These demos are described throughout this guide. Others demos are designed to illustrate how to use TW2 Toolkit application classes, although some of these demos also use TW2 Toolkit components.

For details on running these demonstrations, see the *Telewindows2 Toolkit Java Demos Guide*.

The table below lists the source code location of each Java demo that uses TW2 Toolkit components and provides a description of each:

Java Demos

Source Code	Description

NT:
`itemaccessdemo\ ItemAccessDemo.java`
UNIX:
`itemaccessdemo/ ItemAccessDemo.java`
	Creates a simple Java frame that lets you connect to G2 and launch a Java dialog for getting and setting an attribute of a user-defined item in G2.

NT:
`internationalizationdemo\ InternationalizationDemo.java`
UNIX:
`internationalizationdemo/ InternationalizationDemo.java`
	Shows how to internationalize dialog text for the `itemaccessdemo`.

NT:
`listenerdemo\ ListenerDemoApplet.java`
UNIX:
`listenerdemo/ ListenerDemoApplet.java`
	Creates an applet that listens for changes to attribute values of a user-defined item, and updates those values in a graphical display.

Java Demos

Source Code	Description

NT:
`wkspapplet\WkspApplet.java`
UNIX:
`wkspapplet/WkspApplet.java`
	Creates an applet and corresponding HTML file for connecting to G2 and displaying a workspace view in a Web browser.

NT:
`wkspdemo\WorkspaceFrame.java`
UNIX:
`wkspdemo/WorkspaceFrame.java`
	Creates a Java application that displays a workspace view in a Java frame.

NT:
`customdialogs\rundemo.bat`
UNIX:
`customdialogs/rundemo.sh`
	Overrides the automatically generated item properties dialog, using a custom dialog.

NT:
`wkspbeans\rundemo.bat`
UNIX:
`wkspbeans/rundemo.sh`
	Shows how to use Java Beans on a `WorkspaceView`.

NT:
`palettedemo\rundemo.bat`
UNIX:
`palettedemo/rundemo.sh`
	Shows how to create a palette of G2 objects and a native palette directly from a GFR palette.

NT:
`docs\connectivitydemos\ rundemo.bat`
UNIX:
`docs/connectivitydemos/ rundemo.sh`
	Loads a G2 application to which to connect, using a number of simple Java applications that demonstrate various aspects of basic TW2 Toolkit connectivity.
	These demos appear in Part II, "Connecting to G2" of this guide.

NT:
`docs\launchdialog\ LaunchDialog.java`
UNIX:
`docs/launchdialog/ LaunchDialog.java`
	Shows how to manage dialog resources in a Java application, using your own dialog.
	These demos appear in Chapter 18, "Launching General Dialogs" of this guide.

Java Demos

Source Code	Description

NT:
`wksppanel\`
`SimpleWorkspaceApplication.java`
UNIX:
`wksppanel/`
`SimpleWorkspaceApplication.java`
	Creates a TW2 Toolkit UI application that lets you connect to a single G2 and display workspace views within a multiple workspace panel.

NT:
`wksppanel\BrowserApplication.java`
UNIX:
`wksppanel/BrowserApplication.java`
	Creates a TW2 Toolkit application that allows you to connect to a single G2 and display workspace views within a multiple workspace panel inside a single document frame.

NT:
`singlecxnsdiapp\`
`BrowserApplication.java`
UNIX:
`singlecxnsdiapp/`
`BrowserApplication.java`
	Creates a TW2 Toolkit application that allows you to connect to a single G2 and display workspace views within a single document frame.

NT:
`singlecxnmdiapp\`
`SingleConnectionApplication.java`
UNIX:
`singlecxnmdiapp/`
`SingleConnectionApplication.java`
	Creates a TW2 Toolkit application that allows you to connect to a single G2 and display workspace views within a multiple document frame.

NT:
`multiplecxnsdiapp\`
`WorkspaceBrowserApp.java`
UNIX:
`multiplecxnsdiapp/`
`WorkspaceBrowserApp.java`
	Creates a TW2 Toolkit application that allows you to connect to multiple G2s and display workspace views within a single document frame.

Java Demos

Source Code	Description

NT:
`multiplecxnmdiapp\Shell.java`
UNIX:
`multiplecxnmdiapp/Shell.java`
	Creates a TW2 Toolkit application that allows you to connect to multiple G2s and display workspace views within a multiple document frame.

NT:
`classes\com\gensym\shell\ Shell.java`
UNIX:
`classes/com/gensym/shell/ Shell.java`
	Shows the source code for Telewindows2 Toolkit default application shell.

Part II

Connecting to G2

2

Overview of Connectivity

Presents an overview of the components and core classes you use to connect to G2, the differences in their functionality, and guidelines for using them.

Introduction

Connecting to G2 is essential to any Telewindows2 (TW2) Toolkit client application. Once complete, this connection enables most of the functionality that the TW2 Toolkit components provide.

A G2 server can support multiple clients, both classic Telewindows and Telewindows2 Toolkit. Both a G2 server and a TW2 Toolkit client are capable of initiating a connection.

All of the Telewindows2 Toolkit connectivity classes described in this guide create and manage one connection to a single G2 process.

To create a TW2 Toolkit application capable of creating and handling connections to multiple G2 processes, use the `com.gensym.shell.util.ConnectionManager` class, which is described in detail in Chapter 9, "Creating Telewindows2 Toolkit Applications" the *Telewindows2 Toolkit Java Developer's Guide: Application Classes*.

Connectivity Components

TW2 Toolkit supplies two JavaBeans-compliant components for connecting to a G2 server from within a visual programming environment:

- `ItemRetriever`
- `TwConnector`

Developers writing Java applications in a non-visual programming environment can use the `com.gensym.ntw.TwGateway` class to make a G2 connection.

Before connecting a client to G2, you should consider these issues:

- Which connectivity class to use.
- Which type of connection to make.
- Whether to use a middle-tier server.

As part of connecting to G2 and specifying the properties of that connection, TW2 Toolkit components use a `TwConnectionInfo` object, described in Chapter 5, "Using Connection Information Objects" on page 61.

To gain access to Telewindows2 Toolkit functionality after making a connection to G2, the application must log in to G2, using a `LoginRequest` object, as presented in Chapter 7, "Establishing a G2 Login Session" on page 101.

This chapter supplies background information about the connection options that TW2 Toolkit includes, presents the connectivity components and core classes, and describes the issues to consider before connecting to G2.

Remaining chapters in this part describe:

- Creating a connection, using connectivity components.
- Using connection information objects.
- Creating a connection in a pure Java application.
- Logging in to G2.
- Information about using a middle-tier server.

G2 JavaLink Connectivity to G2

G2 JavaLink supplies the underlying substrate for all of the TW2 Toolkit connectivity to and communication with G2. Much of the G2 JavaLink functionality relies on the G2 attribute access facility, which provides access to almost all G2 item attributes and properties, and generally serves as a doorway into the G2 world. Through attribute access, previously inaccessible G2 class and data structures are available.

G2 includes a set of internal and interrelated system procedures, referred to as the **G2 application programmer's interface (API)**, that provide access to G2 data and actions. The G2 API permits client applications to subscribe to the state of attributes programmatically, rather than having to create "whenever" rules for every item that a client application wishes to monitor. This subscription API allows G2 JavaLink to create remote replications of G2 items, called **stubs**, that remain synchronized, without polling G2.

Further, G2 JavaLink encapsulates G2 Gateway to present G2 knowledge and data from a Java Beans perspective. Through a G2 JavaLink connection, each G2 item becomes a Java class, with of a set of properties, events, and methods, rather than a piece of data accessible only through a set of API calls. G2 JavaLink supplies full type mapping between the G2 data types and Java types, as presented in "Data Type Conversion" on page 12.

By default, G2 JavaLink supplies persistent connections; once G2 JavaLink establishes a G2 connection, that connection remains active regardless of the KB run state. The KB can be paused, reset, or restarted, but the G2 JavaLink connection remains intact.

Understanding the Connectivity Classes

For the UI developer, Telewindows2 Toolkit supplies three classes to connect a Java client to G2. Collectively, this document refers to the following classes as **connectivity classes**:

- `ItemRetriever` component
- `TwConnector` component
- `TwGateway` class

Each of these classes creates a connection to a single G2 process.

Because TW2 Toolkit relies on the connectivity functionality that G2 JavaLink classes supply, you should become familiar with the relevant packages of both products:

Package	Product
`com.gensym.jgi`	G2 JavaLink
`com.gensym.ntw`	Telewindows2 Toolkit

Please refer to the API documentation for specific details of these packages and their classes and methods.

Class Hierarchy of Connectivity Classes

The G2 JavaLink `com.gensym.jgi` package includes two interfaces related to connectivity:

- `G2Access` — Supplies the outbound methods for accessing a G2 connection to make RPC calls or get an item.
- `G2Callbacks` — You do not need to be concerned with this interface.

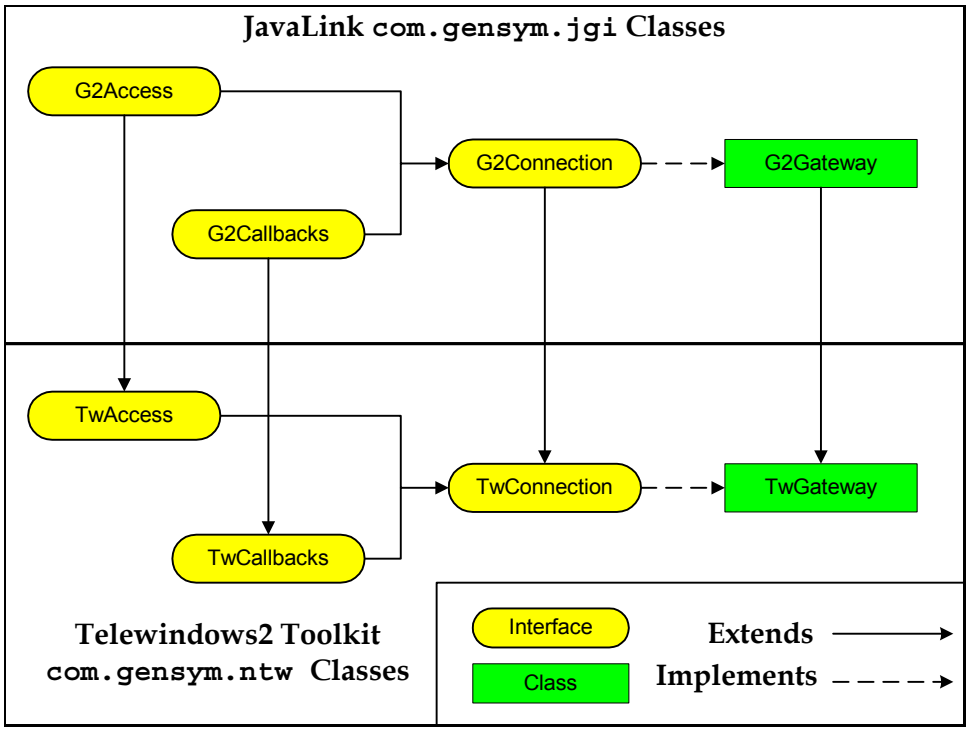
The `G2Connection` interface extends these two interfaces, and the `G2Gateway` class implements `G2Connection`, providing methods for creating a connection.

Telewindows2 Toolkit extends G2 JavaLink functionality through its comparable connectivity interfaces:

- `TwAccess` — A subclass of `G2Access`.
- `TwCallbacks` — A subclass of `G2Callbacks`.

`TwGateway` implements `TwConnection`, which extends these two interfaces.

This diagram illustrates the connectivity class hierarchy:

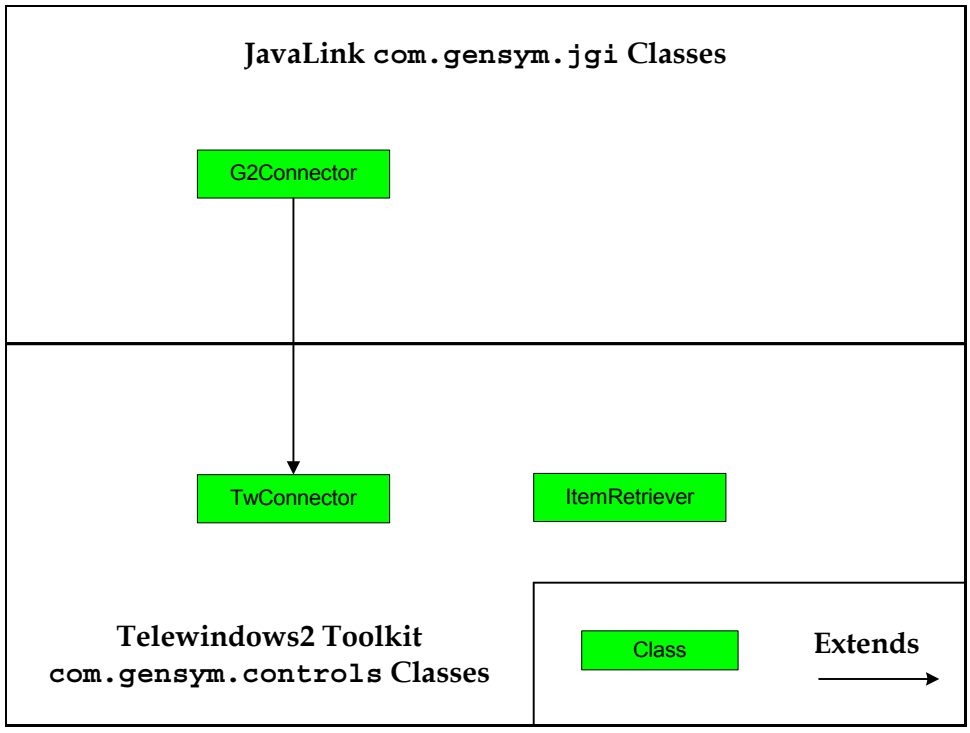


Class Hierarchy of Connectivity Components

The `com.gensym.jgi` package includes `G2Connector`, a component wrapper around the `G2Gateway` class. TW2 Toolkit provides two connectivity components that are based on the `G2Connector` component:

- `ItemRetriever` — An aggregate of the G2 JavaLink `G2Connector` component, which uses the `getOrMakeConnection` method of a `com.gensym.jgi.G2Gateway` to connect to G2.
- `TwConnector` — A wrapper for the `TwGateway` class, which uses the `TwGateway.openConnection` static method to connect to G2.

This diagram illustrates the connectivity class hierarchy:



Determining the Connectivity Class to Use

In most cases, the development environment and the runtime requirements determine which connectivity class to use in your application, as this table describes:

Use...	To develop in...	For an application...
An `ItemRetriever` component	A JavaBeans-compliant visual programming environment	Requiring basic connectivity to G2 to obtain a single item at a time, including a workspace view capable of holding multiple items.
A `TwConnector` component	A JavaBeans-compliant visual programming environment	Requiring full TW2 Toolkit functionality.
The `TwGateway` class	An integrated development environment (IDE) or pure Java development environment	Requiring full TW2 Toolkit functionality.

For information on...	See...
`ItemRetriever`	Chapter 3, “Using ItemRetriever.”
`TwConnector`	Chapter 4, “Using TwConnector.”
`TwGateway`	Chapter 6, “Using TwGateway.”

Choosing a Connection Type

Once you choose a connectivity class, you can then decide which connection type to create.

By default, any application that uses the TW2 Toolkit connectivity classes automatically creates a Telewindows2 Toolkit connection type. However, since TW2 Toolkit connectivity classes extend G2 JavaLink classes, your client has access to all of the G2 JavaLink capabilities, including the ability to create a G2 JavaLink connection type through a TW2 Toolkit client.

Compared with a Telewindows2 Toolkit connection type, a G2 JavaLink connection offers less functionality, as this table summarizes:

Functionality	G2 JavaLink	TW2 Toolkit
RPC calls	✓	✓
Data-aware objects	✓	✓
Middle-tier server support	✓	✓
Shares connections by default	✓	
Forces new connection by default		✓
Get unique named item	✓	✓
Obtain list of available workspaces		✓
Display workspace views		✓
Displays text editor for editing grammatical item properties		✓
Log in to G2		✓
Get and set user mode		✓
Subscribe to KB events		✓
Requires floating TW license		✓

Typically, applications that do not need to display a workspace view can use a G2 JavaLink connection type, while those requiring these capabilities should create a Telewindows2 Toolkit connection type.

All of the connectivity classes support both types of connection. Information about setting the connection type is presented in "Connection and Interface Classes" on page 67.

Result of Connecting to G2

When a TW2 Toolkit client connects to G2, these two operations occur:

- The connection method returns an instance of a connection class to the client.
- G2 creates an instance of an interface class in the G2 server KB that communicates with the client.

Together, the connection class object that the client returns and the interface class item that G2 creates determine the connection type, as follows:

This type of connection...	Returns an instance of this class in the client...	And creates an instance of this class in the G2 server...
G2 JavaLink	`com.gensym.jgi.G2Gateway`	gsi-interface
Telewindows2 Toolkit	`com.gensym.ntw.TwGateway`	ui-client-interface

Creating a G2 JavaLink Connection

A Telewindows2 Toolkit application can create G2 JavaLink connections by calling a version of this methods on a `com.gensym.jgi.G2Gateway`:

```
getOrMakeConnection
```

A G2 JavaLink connection provides basic component access to items and the ability to subscribe to item changes. Connecting to G2 through G2 JavaLink precludes the use of the workspace views and other functionality specific to TW2 Toolkit. Thus, when connected to G2 through G2 JavaLink, you can specify the class of an item to retrieve, such as kb-workspace, and provide the name of the item. However, your application will be unable to display a workspace view, because this functionality is available only through a Telewindows2 Toolkit connection with a login session.

Shared Connections

By default, G2 JavaLink connections are shared, which means other connection requests to the same host and port use an existing connection, if one exists. For detailed information about shared connections, see “Sharing a Connection” on page 69.

Interface Class

For each G2 JavaLink client connection, G2 automatically creates an instance of a gsi-interface object. The interface object further defines the type of connection, because it contains knowledge about the connection. However, by design, the interface object omits any login information.

Creating a Telewindows2 Toolkit Connection

A client application creates a Telewindows2 Toolkit connection by calling a version of this static methods on a `com.gensym.ntw.TwGateway`:

`openConnection`

Shared Connections

By default, Telewindows2 Toolkit connections are not shared. When a connection is not shared, connection requests to the host and port of an existing connection always force a new connection. To change this property, see "Sharing a Connection" on page 69.

Interface Class

For each TW2 Toolkit client connection, G2 automatically creates an instance of a ui-client-interface object. This assumes you are connecting using a two-tier connection whereby you provide a host and port only. For more information on creating a two-tier connection, see Chapter 5, "Using Connection Information Objects" on page 61. For information on connecting using a three-tier connection, see "Using a Middle-Tier Server" on page 34.

Login Requirements

Once a TW2 Toolkit client has created a connection, it must subsequently call the `login` method on a `TwGateway`, which causes G2 to create an instance of a ui-client-session. Once you have established a successful login and a ui-client-session exists, you have access to full TW2 Toolkit functionality through the connection.

Without establishing a login session, a client can connect to G2 by using the `TwGateway.openConnection` static method, but it will be unable to create a workspace view.

For more information about logging in to either a secure or non-secure G2, see Chapter 7, "Establishing a G2 Login Session" on page 101.

Using a Middle-Tier Server

When your application connects to a G2 process, it can optionally use a middle-tier server that uses Java Remote Method Invocation (RMI). By default, Telewindows2 Toolkit uses a two-tier connection configuration, in which all communication occurs directly between the client and G2 through the use of messages and remote procedure calls (RPCs).

All of the Telewindows2 Toolkit connectivity classes support the use of a middle-tier.

Using a middle-tier server is described in more detail in Chapter 8, “Using a Middle-Tier Server” on page 117.

3

Using ItemRetriever

Describes how to use an ItemRetriever component to connect to G2 and obtain a single item.

Introduction

The `ItemRetriever` component connects to G2 and lets your application:

- Create either a G2 JavaLink or Telewindows2 Toolkit connection type.
- Support two- and three-tier connections.
- Obtain a single, named, G2 item.

Normally you use an `ItemRetriever` with an `ItemProxy` component while developing a custom item properties dialog in a JavaBeans-compliant visual programming environment. You can use an `ItemRetriever` to connect to G2 and retrieve the item so you can test such a dialog without first having to register it with G2, as described in Chapter 16, "Launching Custom Item Properties Dialogs" on page 305.

Because you use an `ItemRetriever` component only during development, the component does not provide a method for closing the connection.

The ItemRetriever component uses:

- A default set of connectivity properties, which you can set explicitly through the properties dialog or by using accessor methods.
- Default login information for use with a non-secure G2, which you can set explicitly to log in to a secure G2.

This chapter describes how to use an ItemRetriever component and includes these sections:

- Using an ItemRetriever component in a Java programming environment.
- Reference information for the ItemRetriever component's properties, events, and methods.

To connect to G2 and retrieve an item in a non-visual programming environment, use these methods on a com.gensym.jgi.G2Gateway:

- getOrMakeConnection
- getUniqueNamedItem

See the G2 JavaLink API documentation for details.

This chapter describes how to set basic connectivity properties only. For information on setting advanced connectivity properties, see:

- Chapter 5, "Using Connection Information Objects."
- Chapter 7, "Establishing a G2 Login Session."

Packages Covered

The ItemRetriever component is part of the com.gensym.controls package. Its connectivity, however, derives from capabilities in the G2 JavaLink com.gensym.jgi package.

Thus, this chapter describes certain classes in the jgi package, along with the connectivity-related interfaces and classes in the com.gensym.controls package.

com.gensym.controls

Interfaces

ItemRetrievalListener

Classes

ItemRetrievalEvent
ItemRetriever

com.gensym.jgi

Exceptions

```
ConnectionTimedOutException
G2AccessInitiationException
G2AccessException
```

Using an ItemRetriever Programmatically

For Java applications that need to get a G2 item, most developers will typically use the `TwGateway.openConnection` method to make a connection, then call the JavaLink method `G2Gateway.getUniqueNamedItem` to retrieve an item. However, for developers who wish to use an `ItemRetriever`, this section describes the use of this component within a Java application or applet to retrieve an item.

Using ItemRetriever Constructors

The `ItemRetriever` component has two constructors.

To use the default constructor:

➔ **`ItemRetriever ()`**

The default constructor creates the component with all of its properties set to their default values. You must then use accessor methods to set each property value.

To specify properties in the constructor:

➔ **`ItemRetriever`**
(`G2ConnectionInfo` *`connectionInfo`*,
`String` *`itemClassName`*,
`String` *`itemName`*)

Argument	Description
`connectionInfo`	A `G2ConnectionInfo` object in which you can set any number of connectivity properties, as described in Chapter 5, "Using Connection Information Objects" on page 61.
`itemClassName`	The G2 class of the item to retrieve. The default is `ITEM`.
`itemName`	The name of the item to retrieve. The default is `null`.

Retrieving an Item

The ItemRetriever component creates a connection to G2 whenever a call is made to its retrieveItem method. If necessary, it also calls the login method on the connection object, as Chapter 7, "Establishing a G2 Login Session" on page 101 describes.

Internally, the component establishes a connection by using the JavaLink G2Gateway.getOrMakeConnection method, which takes as its argument an instance of a G2ConnectionInfo, described in Chapter 5, "Using Connection Information Objects" on page 61.

The ItemRetriever uses the JavaLink G2Gateway.getUniqueNamedItem method to obtain an item from G2.

Once you have set the connection and item information, you can retrieve the item.

You cast the retrieved item to its particular class so you can reference properties and methods of the returned class directly.

When you retrieve an item from a connection, you need to handle this exception:

```
com.gensym.jgi.G2AccessException
```

To retrieve an item:

➔ `ItemRetriever.retrieveItem()`

The following sections show how to set ItemRetriever properties and show examples of retrieving an item, using each constructor.

Setting ItemRetriever Properties

When using an ItemRetriever in a Java application, you must always set the itemName property.

Although the default class name for the component is ITEM, which represents all public G2 classes, we recommend that you always set the itemClassName property explicitly to prevent retrieving the wrong item.

Unless you are making a connection to the default host localhost on port 1111, you must also set connectivity properties.

To do this, you can use the default constructor, then set the relevant information by using accessor methods.

If you do not set any connectivity properties, the ItemRetriever uses a G2ConnectionInfo object with default values to connect to G2. Each time you use an accessor method to set any connectivity properties, the ItemRetriever creates a new G2ConnectionInfo object to use the next time the ItemRetriever.retrieveItem method is called.

To use the default ItemRetriever constructor and set its properties:

```
//Retrieve an item using default constructor
  public static void getWorkspace1 () {
    //Create retriever
    retriever = new ItemRetriever();
    //Set class and item information
    retriever.setItemClassName("KB-WORKSPACE");
    retriever.setItemName("WKSP-1");
    retriever.setHost("localhost");
    retriever.setPort("1112");
    try {
       //Retrieve item and cast result
      kbWorkspace = (KbWorkspace)retriever.retrieveItem();
       //Handle exceptions
    } catch (Exception e) {
    }
  }
```

Passing a Connection Information Object to an ItemRetriever

If the connection information is known at compile time, you can also pass an instance of a `TwConnectionInfo` as an argument to the `ItemRetriever` constructor. The connection information object contains the required connection information.

The next example illustrates how to do this, while specifying the GSI-INTERFACE class as the item to retrieve.

For information on creating a connection information object, see Chapter 5, "Using Connection Information Objects" on page 61.

To use the ItemRetriever constructor with a G2ConnectionInfo object:

```
//Retrieve an item, using connection info object
  public static void getWorkspace2 () {
    //Create a connection info object
    TwConnectionInfo connectionInfo = new TwConnectionInfo
       ("localhost", "1112");
    //Create retriever
    retriever = new ItemRetriever(connectionInfo, "KB-WORKSPACE",
                                  "WKSP-2");
    try {
       //Retrieve item and cast result
      kbWorkspace = (KbWorkspace)retriever.retrieveItem();
       //Handle exceptions
    } catch (Exception e) {
    }
  }
```

Using JavaLink Methods

`ItemRetriever` uses two JavaLink methods to create the connection and retriever the item.

Using getOrMakeConnection

The `ItemRetriever` uses this static method on `com.gensym.jgi.G2Gateway` to make a G2 connection:

```
getOrMakeConnection
    (G2ConnectionInfoObject info)
    -> G2Connection connectionType

ConnectionTimedOutException
G2AccessInitiationException
```

Argument	Description
info	A `G2ConnectionInfo` consisting of the properties you specify. If you do not specify properties, uses all default values.

Return	Description
`connectionType`	An implementation of the `G2Connection` interface. Depending on the type of connection being made, this is either a `G2Gateway` or a `TwGateway`.

Exception	Description
`ConnectionTimedOutException`	A new connection was not initiated within the default timeout period.
`G2AccessInitiationException`	An error occurred during the connection attempt.

Using getUniqueNamed Item

The `ItemRetriever` uses this method on `com.gensym.jgi.G2Gateway` to retrieve an item from G2:

```
getUniqueNamedItem
    (Symbol itemClass,
     Symbol itemName)
    -> Item returnedItem

G2AccessException
```

Argument	Description
itemClass	A symbol naming the G2 class of the item to retrieve.
itemName	A symbol naming the item to retrieve.

Return	Description
returnedItem	The retrieved item, which is an instance of a `com.gensym.classes.Item`.

Exception	Description
G2AccessException	An error occurred during communication or no such item exists.

This method returns a handle to the named item in G2. To obtain the initial values of the item, as well as notification of item events, such as when the item is modified or deleted, your application must call the JavaLink `com.gensym.classes.Item.addItemListener` method. See the JavaLink API documentation for details.

Subscribing to ItemRetriever Events

The `ItemRetriever` notifies registered listeners of these events:

Event	Description
itemRetrieved	The item was retrieved successfully.
itemRetrievalFailed	A `G2AccessException` occurred when attempting to retrieve the item.

Because the ItemRetriever component does not attempt a G2 connection until the ItemRetriever.retreiveItem method is called, you must subscribe to ItemRetriever events *before* retrieving the item by adding the ItemRetriever as an ItemRetrievalListener.

The original example added the listener after the item was retrieved. Moving this before the code that retrieves the item causes a compile error: this not defined.

To subscribe to ItemRetriever events:

```
private static ItemRetriever retriever;
private static KbWorkspace kbWorkspace;

//Create retriever
retriever = new ItemRetriever();

//Add retriever as a listener
retriever.addItemRetrievalListener(this);
```

Components that Implement the ItemRetrievalListener Interface

Two TW2 Toolkit components implement the ItemRetrievalListener interface:

- com.gensym.wksp.WorkspaceView
- com.gensym.controls.ItemProxy

If your application needs to listen for item retrieval events, you must implement the ItemRetrievalListener in your own listener class.

Informing an ItemRetrievalListener of Events

If you create a class that subscribes to ItemRetriever events as an ItemRetrievalListener, your class is informed the first time the ItemRetriever retrieves an item. If the ItemRetriever has already retrieved its item, your class is informed of the event when it subscribes as a listener.

A component can be informed more than once of the same item being retrieved, because of the way in which an ItemRetriever component generates events. The ItemRetriever:

- Generates an event each time it retrieves an item from G2.
- Retrieves an item each time a component property changes and each time its retrieveItem method is called.

As an example, this sequence of events can cause an ItemRetrievalListener to be informed twice about the same item being retrieved:

1. An application App creates an ItemRetriever component and calls its retrieveItem method to get an item. The item named Foo is returned.
2. ItemRetrievalListener Bar subscribes to ItemRetriever events. Because Foo has already been retrieved, Bar receives an itemRetrieved event immediately.
3. The itemClassName property of the ItemRetriever component is changed from ITEM to KB-WORKSPACE.
4. The next call to the retrieveItem method detects that a component property has changed and retrieves Foo again.
5. Listener Bar receives a second itemRetrieved event.

Conversely, the ItemRetriever component does not retrieve the same item more than once unless changes to its properties have been made. Calling the retrieveItem method multiple times without changing existing component properties neither gets the item, nor generates itemRetrieved events.

Handling Connection Exceptions

The ItemRetriever component can generate any of the TwGateway exceptions, described in "Handling Connection Exceptions" on page 79.

In addition, it can generate an exception when retrieving an item, as described in "Retrieving an Item" on page 38.

Within an IDE, no formal technique exists for handling exceptions. By default, if an exception occurs during a connection attempt from ItemRetriever, an exception message is printed on the command window. If you want to handle exceptions in a different way, you can create custom hookups that add exception handling code. For an example of using custom hookups, see "Using G2 Item Components in Dialogs" on page 226.

Closing a Connection

When a successful connection is made to G2 to retrieve an item, that connection remains open. Currently, no method exists for closing a connection that you make by using an ItemRetriever.

Example

This example retrieves two KB workspace items, using two ItemRetriever components. The first retriever uses the default constructor and sets the connection properties explicitly, and the second retriever uses a TwConnectionInfo object.

```
import com.gensym.controls.ItemRetriever;
import com.gensym.ntw.TwConnectionInfo;
import com.gensym.classes.KbWorkspace;

public class MyItemRetriever {
  private static ItemRetriever retriever;
  private static KbWorkspace kbWorkspace;
  private static TwConnectionInfo connectionInfo;

   //Retrieve an item using default constructor
  public static void getWorkspace1 () {
    //Create retriever
    retriever = new ItemRetriever();
    //Set class and item information
    retriever.setItemClassName("KB-WORKSPACE");
    retriever.setItemName("WKSP-1");
    retriever.setHost("localhost");
    retriever.setPort("1112");
    try {
      kbWorkspace = (KbWorkspace)retriever.retrieveItem();
      System.out.println(kbWorkspace);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  //Retrieve an item, using connection info object
  public static void getWorkspace2 () {
    //Create a connection info object
    TwConnectionInfo connectionInfo = new TwConnectionInfo
       ("localhost", "1112");
    //Create retriever
    retriever = new ItemRetriever(connectionInfo, "KB-WORKSPACE",
                                  "WKSP-2");
```

```
    try {
      //Retrieve item and cast result
      kbWorkspace = (KbWorkspace)retriever.retrieveItem();
      System.out.println(kbWorkspace);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  // MAIN
  public static void main(String[] args){
    System.out.println("Got to main!");
    getWorkspace1();
    getWorkspace2();
    System.out.println ("End of Main!");
    System.exit (0);
  }
}
```

Running this example produces these messages in the command window, assuming you have a G2 running on your local machine on port 1112:

```
Got to main!
com.gensym.classes.KbWorkspaceImpl@1ed9b0(KB-WORKSPACE, # = 3v0, ok)
com.gensym.classes.KbWorkspaceImpl@1f0778(KB-WORKSPACE, # = 3v0, ok)
End of Main!
```

Note Because `ItemRetriever` does not provide a method for closing the connection, this example generates an error in the G2 Operator Logbook when it finished.

ItemRetriever Reference

The `ItemRetriever` is an invisible component in a dialog, though it uses this representation within a visual programming environment:

`com.gensym.controls.ItemRetriever`

Properties

These are the properties and associated accessor methods of an `ItemRetriever` component:

Property Get Property Set Property	Type	Description

`brokerURL`
`getBrokerURL`
`setBrokerURL`
	`String`	The URL of a middle-tier server. This field is required only for connections to G2 that use a middle-tier server.
		Default = `null`

`connectionClassName`
`getConnectionClassName`
`setConnectionClassName`
	`String`	The name of the connection class on which this connection is made.
		Default = `com.gensym.ntw.TwGateway`

`connectionInfo`
`getConnectionInfo`
`setConnectionInfo`
	`G2ConnectionInfo`	The `com.gensym.jgi.G2ConnectionInfo` object that the `ItemRetriever` will use the next time it retrieves an item.

`gsiInterfaceClassName`
`getGsiInterfaceClassName`
`setGsiInterfaceCkassName`
	`String`	The name of the GSI Interface class in G2 that holds the connection information.
		Default = `UI-CLIENT-INTERFACE`

Property Get Property Set Property	Type	Description
host getHostName setHostName	String	The name of the host on which G2 is running. Default = localhost
itemClassName getItemClassName setItemClassName	String	The name of the G2 class for which the ItemRetriever is seeking information. Default = ITEM
itemName getItemName setItemName	String	The name of the item that the ItemRetriever is to retrieve. Default = null
port getPort setPort	String	The port number to which G2 is listening. Default = 1111
userMode getUserMode setUserMode	String	The G2 user mode of the user. The default value of null logs in to G2 in Administrator mode. Default = null
userName getUserName setUserName	String	The G2 user name. The default value of null logs in to G2 as the currently logged in G2 user. Default = null

For information on...	See...
Using advanced connectivity properties	Chapter 5, “Using Connection Information Objects.”
Connecting to a secure G2	Chapter 7, “Establishing a G2 Login Session.”
Connecting to G2 through a middle-tier server	Chapter 8, “” on page 117.

Events and Listeners

Components can register as a listener to receive notification of these `ItemRetriever` events:

Listener (Package) Registration Methods Events	Description
`ItemRetrievalListener (com.gensym.controls)`	
`addItemRetrievalListener` `removeItemRetrievalListener`	
`itemRetrieved`	Dispatched when the specified item was retrieved from G2.
`itemRetrievalFailed`	Dispatched when the component was unable to retrieve the specified item.

Methods

These are the methods of an `ItemRetriever` component, other than its accessor methods:

Method	Description
`retrieveItem`	Calls G2 synchronously to connect to G2, using the current information in the `G2ConnectionInfo` object, and retrieves the unique named item.
`setUserPassword`	While no property exists for editing the user password in a visual programming environment, you can set the user password programmatically by using an accessor method. For security reasons, no accessor method exists for getting the user password. The `setUserPassword` method only works if the connection is to a `com.gensym.ntw.TwAccess`.
`toString`	Overrides the `Object` class `toString` method for providing a description of the object.

4

Using TwConnector

Describes how to use the TwConnector component to connect to G2.

Introduction

The `TwConnector` component incorporates the functionality of the `com.gensym.ntw.TwGateway` class in a Java Bean for use within a JavaBeans-compliant visual programming environment, where it is not possible to call the `TwGateway.openConnection` static method.

The `TwConnector` component inherits its definition from `com.gensym.jgi.G2Connector`, which means your client application can:

- Connect to a single G2 process at a time.
- Send data to and receive data from a G2 process.
- Support 2- and 3-tier connections.
- Make RPC calls.
- Subscribe to G2 state events.

In addition, `TwConnector` lets your application perform these tasks:

- Establish a login session with G2.
- Get and set user modes.

- Get user menu choices.
- View workspaces.
- Subscribe to workspace show and hide events and KB events.

By default, a `TwConnector` creates a Telewindows2 (TW2) Toolkit connection type, which is an unshared connection. This means that each time you launch a dialog that contains a `TwConnector` component, the component creates a new connection to G2.

Note When creating dialogs that use a `TwConnector`, you should always close the connection when the dialog closes, unless the dialog provides an explicit way to close the connection; otherwise, you might run out of TW2 Toolkit licenses for connecting to G2. Alternatively, during development, you can create a shared connection, as described in "Sharing a Connection" on page 69.

In addition to creating a connection, a `Twconnector` uses default login information to log in to a non-secure G2. You can set the login information explicitly through the properties dialog or by using accessor methods to log in to a secure G2.

This chapter shows you how to use the `TwConnector` component to connect to G2 and includes these sections:

- Packages covered.
- Reference information for the `TwConnector` component's properties, events, and methods.

This chapter describes how to set basic connectivity properties only. For information on setting advanced connectivity properties, see:

- Chapter 5, "Using Connection Information Objects" on page 61.
- Chapter 7, "Establishing a G2 Login Session" on page 101.

The `TwConnector` component is part of the `com.gensym.controls` package. However, because it is a Java Beans version of the `TwGateway` class, which supplies all of its connectivity functionality, the advanced features of the `TwConnector` component are described fully in Chapter 6, "Using TwGateway" on page 73.

This chapter does not include a section on using the `TwConnector` component programmatically, because no benefit exists in doing so. While developers can, of course, use this component outside of a visual programming environment within an integrated development environment (IDE) or pure Java development environment, we recommend that you instead use the `TwGateway` class to connect to G2.

Packages Covered

com.gensym.controls

TWConnector

com.gensym.jgi

Exceptions

ConnectionTimedOutException
G2AccessInitiationException
G2AccessException

TwConnector Reference

The `TwConnector` component is a visual component that uses this representation in a visual Java programming environment:

`com.gensym.controls.TwConnector`

If you use the `TwConnector` component without specifying host and port properties, an attempt is made to connect to host `localhost` at port `1111`.

Properties

These are the properties and associated accessor methods of a `TwConnector` component:

Property **Get Property** **Set Property**	**Type**	**Description**

`brokerURL`
`getBrokerURL`
`setBrokerURL`
	`String`	The URL of a middle-tier server. This field is required only for connections to G2 that are using a middle-tier server.
		Default = `null`

`connectionClassName`
`getConnectionClassName`
`setConnectionClassName`
	`String`	The name of the connection class on which this connection is made.
		Default = `com.gensym.ntw.TwGateway`

`forceNew`
`getForceNew`
`setForceNew`
	`boolean`	Specifies whether a connection is created each time the `openConnection` method is called.
		Default = `false`

`gsiInterfaceClassName`
`getGsiInterfaceClassName`
`setGsiInterfaceClassName`
	`String`	The class of interface object that G2 creates in the KB to represent a client connection.
		Default = `UI-CLIENT-INTERFACE`

Property Get Property Set Property	Type	Description
gsiInterfaceName getGsiInterfaceName setGsiInterfaceName	String	The name of the interface instance that G2 creates in the KB to represent this connection. Default = TW-CONNECTOR-BEAN-INTERFACE
host getHost setHost	String	The name of the host on which G2 is running. Default = localhost
logicalName getLogicalName setLogicalName	String	A logical name for a shared G2 connection. Once the connection exists, other TwConnector components can refer to the logical name to share that connection. Default = null
permanent isPermanent setPermanent	Boolean	Specifies whether the connection is permanent when you reset G2. Default = true
port getPort setPort	String	The port number to which G2 is listening. Default = 1111
remoteProcessInitString getRemoteProcessInitString setRemoteProcessInitString	String	The string to pass during a connection attempt.
sharable isShared setShared	boolean	Specifies whether the connection can be shared. Default = false

Property Get Property Set Property	Type	Description

`userMode`
`getUserMode`
`setUserMode`
	`String`	The G2 user mode of the logged in user. The default value of `null` logs in to G2 in Administrator mode.
		Default = `null`

`userName`
`getUserName`
`setUserName`
	`String`	The user name of the logged in user. The default value of `null` logs in to G2 as the currently logged in G2 user.
		Default = `null`

For information on...	See...
Using advanced connectivity properties	Chapter 5, "Using Connection Information Objects" on page 61.
Connecting to a secure G2	Chapter 7, "Establishing a G2 Login Session" on page 101.
Connecting to G2 through a middle-tier server	Chapter 8, "Using a Middle-Tier Server" on page 117.

Events and Listeners

Components can receive notification of the following `TwConnector` events, which the following listeners implement:

Listener (Package) Registration Methods Events	Description
G2ConnectionListener (com.gensym.jgi)	
addG2ConnectionListener removeG2ConnectionListener	
g2IsPaused	Dispatched when the G2 process has been paused.
g2IsResumed	Dispatched when the G2 process has been resumed.
g2IsReset	Dispatched when the G2 process has been reset.
g2IsStarted	Dispatched when the G2 process has been started.
g2ConnectionClosed	Dispatched when the current connection has been closed.
g2ConnectionInitialized	Dispatched when the connection attempt has been initialized successfully.
g2MessageReceived	Dispatched when a message has been received from G2.
communicationError	Dispatched when an error occurred during communication between the `TwConnector` component and the G2 process.
readBlockage	Dispatched when an attempt to read data from G2 failed.
writeBlockage	Dispatched when an attempt to write data to G2 failed.

Listener (Package) Registration Methods Events	Description
TwConnectionListener (com.gensym.ntw)	
addTwConnectionListener removeTwConnectionListener	
loggedIn	Dispatched when the user has logged into G2.
loggedOut	Dispatched when the user has logged out of G2.
userModeChanged	Dispatched when the current user has changed the user mode.
WorkspaceShowingListener (com.gensym.ntw)	
addWorkspaceShowingListener removeWorkspaceShowingListener	
showWorkspace	Dispatched when G2 has programmatically shown a workspace.
hideWorkspace	Dispatched when G2 has programmatically hidden a workspace.

For details on using these events, as well as information on subscribing to KB module and message events, see "Handling Events" on page 79.

Methods

These are the methods of a `TwConnector` component, other than its accessor methods:

Method	Description
`createConnection`	Creates a connection to G2, using the current set of connectivity properties, then logs on to G2, using specified login properties.
`createItem`	Creates an item given a class name as a symbol.
`getDialogManager`	Returns the `com.gensym.dlgruntime.DialogManager` for the current connection.
`getKb`	Gets the KB object associated with this connection.
`getNamedWorkspaces`	Returns a sequence of named KB workspace from G2.
`getUserMenuChoice`	Gets a user menu choice from the current KB.
`setUserPassword`	While no property exists for editing the user password in a visual programming environment, you can set the user password programmatically by using an accessor method. For security reasons, no accessor method exists for getting the user password.

For details on using these methods, see "Working with Telewindows2 Toolkit Connections" on page 89.

5

Using Connection Information Objects

Describes the use and purpose of the classes of connection information objects that JavaLink and Telewindows2 Toolkit provide.

Introduction

The Telewindows2 (TW2) Toolkit connectivity components call one of two methods internally to connect to G2. Calls to both methods consist of a single argument that is a type of `G2ConnectionInfo` object:

This component...	Calls this method...	On this class...	Which takes this argument...
`ItemRetriever`	`getOrMakeConnection`	`G2Gateway`	`G2ConnectionInfo`
`TwConnector`	`openConnection`	`TwGateway`	`TwConnectionInfo`

To connect to G2 programmatically, you can call the following static method on `com.gensym.ntw.TwGateway`:

```
openConnection(TwConnectionInfo info)
```

This chapter describes the use of connection information objects and the properties they specify, presenting information about setting connection properties that generally applies to all of the connectivity classes.

If you are using one of the connectivity components in a JavaBeans-compliant visual programming environment, the component takes care of creating the connection information argument for you when you specify the properties of the component.

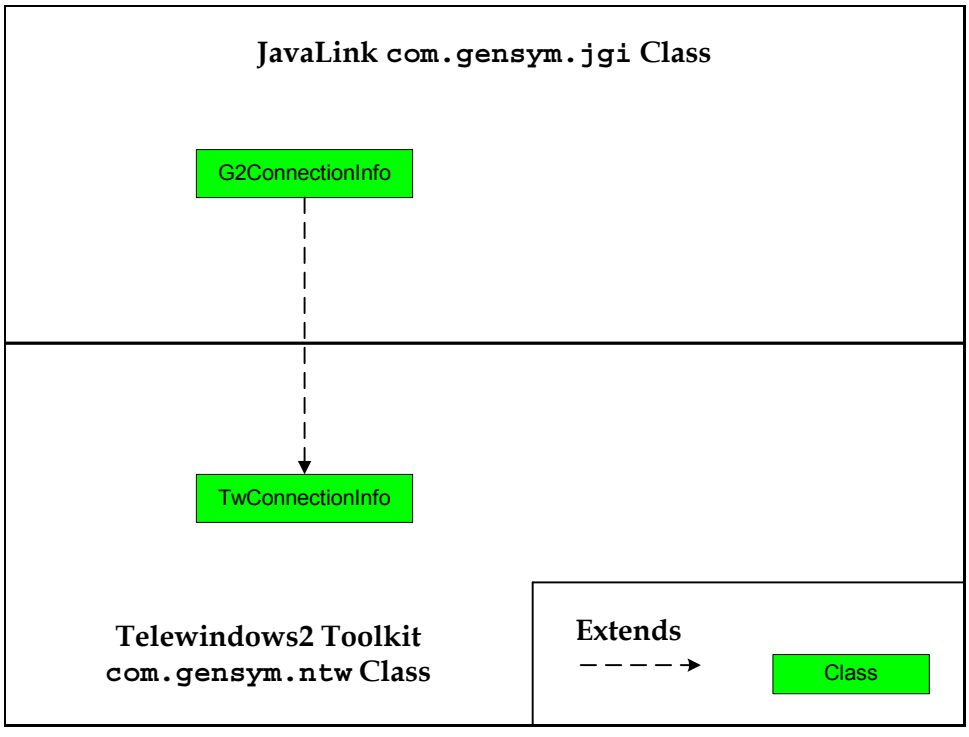
The term **connection information object** refers to both the `G2ConnectionInfo` and `TwConnectionInfo` object classes, unless noted otherwise. Examples throughout this chapter use mostly a `TwConnectionInfo` object.

The TwConnectionInfo Class Hierarchy

The two classes of connection information objects are:

- `com.gensym.jgi.G2ConnectionInfo`
- `com.gensym.ntw.TwConnectionInfo`

The `TwConnectionInfo` class extends the G2 JavaLink `G2ConnectionInfo` class:



Packages Covered

com.gensym.jgi

G2ConnectionInfo

com.gensym.ntw

TwConnectionInfo

Relevant Demos

The Telewindows2 Toolkit wkspapplet demo uses a TwConnectionInfo to:

- Connect to G2 through either a two- or three-tier connection.
- Create a sharable connection.
- Set the connection information for an ItemRetriever component.

The demo is located in this directory, depending on your platform:

NT: `%SEQUOIA_HOME%\classes\com\gensym\demos\wkspapplet\WkspApplet.java`

UNIX: `$SEQUOIA_HOME/classes/com/gensym/demos/wkspapplet/WkspApplet.java`

Using Connection Information Objects

A connection information object sets the properties that a G2 connection request requires. While you can specify one or more properties of a such an object to set the connection, no requirement exists to do so. Making a connection with any of the following methods implicitly constructs an instance of a G2ConnectionInfo object with all of its properties set to their default values:

Class	Method
G2Gateway	getOrMakeConnection (String host, String port)
G2Gateway	getOrMakeConnection (String brokerURL, String host, String port)
TwGateway	openConnection (String brokerURL, String host, String port)

When using either the `ItemRetriever` or `TwConnector` components, you can set the connectivity properties through a properties dialog in a visual Java programming environment. Alternatively, all of the connectivity classes include accessor methods to get and set the connection information object properties.

For examples of how to specify the basic properties required to create a connection, see "Using an ItemRetriever Programmatically" on page 37.

Regardless of the way in which an application gets and sets the connectivity properties, a connection information object always exists when a TW2 Toolkit client creates a connection to a G2 server.

Connection information objects are immutable, meaning you cannot set any of their properties once an instance has been created.

Creating a Connection Information Object

Both connection information object classes include constructors to specify between two and twelve available properties. The `G2ConnectionInfo` class includes a complete suite of constructors for every available permutation, whereas the `TwConnectionInfo` class does not.

Using a constructor with no arguments creates a connection information object with default values for every property.

For a complete description of the constructor methods, refer to the G2 JavaLink and Telewindows2 Toolkit API documentation for `com.gensym.jgi.G2ConnectionInfo` and `com.gensym.ntw.TwConnectionInfo`, respectively.

Basic and Advanced Properties

The following sections describe the following basic and advanced properties of a connection information object:

Basic Property	Type	Advanced Property	Type
brokerURL	String	connectionClassName	String
hostName	String	forceNew	boolean
port	String	gsiInterfaceClassName	String
		interfaceName	String
		isPerm	Boolean
		logicalName	String
		connectionClassName	String
		rpis	String
		sharable	boolean

Setting Basic Connectivity Properties

Basic properties are those that most likely require setting in any client application. You can specify the basic properties in all of the class constructors, except those with no arguments.

Setting the Host and Port

The `hostName` and `port` properties let you specify the host and port to which your application will connect. The default values for these properties are `localhost` and `1111`, respectively.

A `TwConnectionInfo` constructor exists specifically to set only the host and port properties.

To create a TwConnectionInfo object with host and port properties:

```
//Create a connection info object
TwConnectionInfo connectionInfo = new TwConnectionInfo
   ("myhost", "1114");
```

Specifying a Middle-Tier Server

The `brokerURL` property lets you connect to G2 through a middle-tier server. The default value of this property is `null`.

For a complete discussion of connecting this way, see Chapter 8, "Using a Middle-Tier Server" on page 117.

You can use a `TwConnectionInfo` constructor that lets your application connect to G2 through a middle-tier server, specifying the server machine and name as a string with this format:

//machine/ServerName

Argument	Description
//machine	The name of the machine on which the RMI registry has been started.
/ServerName	The name assigned to the server when it was started.

The following example creates a new `TwConnectionInfo` object that connects to G2 through a middle tier.

To create a TwConnectionInfo object that uses a middle tier:

```
//Create a connection info object with brokerURL:
TwConnectionInfo connectionInfo = new TwConnectionInfo
   ("//mynode/demoserver", "myhost", "1114");
```

Setting Advanced Connectivity Properties

This section describes the advanced properties of connection information objects and the results you can expect from changing their default values.

Properties are classified as advanced because, in many cases, applications do not need to change them, and not all constructors support setting them. Further, changing some of the advanced properties can affect fundamental behavior of your application.

Interrelated and Independent Properties

These advanced properties are interrelated:

If you change this property...	Then also change this property...
connectionClassName	gsiInterfaceClassName
forceNew	sharable
logicalName	forceNew sharable

The level of interrelatedness differs for these properties. For the connectionClassName or gsiInterfaceClassName, changing one property without changing the other produces unpredictable results. For the forceNew and sharable properties, changing one without changing the other does not make sense. For the logicalName, forceNew, and sharable properties, changing one without changing the other does not have the intended effect.

These properties work independently of others:

- interfaceName
- isPerm
- rpis

The following sections describe the advanced properties.

Connection and Interface Classes

The connectionClassName property specifies the connection class that is returned to the calling method upon a successful connection to G2. For TW2 Toolkit connection types, the default is com.gensym.ntw.TwGateway. All of the TW2 Toolkit connectivity components return this connection class, thus creating a TW2 Toolkit connection type, by default. The TwGateway class returns an implementation of the com.gensym.ntw.TwAccess interface upon successful connection to G2.

The gsiInterfaceClassName property specifies the interface class that G2 creates in the KB for a TW2 Toolkit client. All of the TW2 Toolkit connectivity classes cause G2 to create an instance of the ui-client-interface class.

While it is possible to change these properties when creating TW2 Toolkit connection types, we do not recommend it. In particular, the connectionClassName property must be com.gensym.ntw.TwGateway when creating TW2 Toolkit connection types by calling the openConnection static method on TwGateway. Similarly, when creating G2 JavaLink connection types by

calling getOrMakeConnection on a com.gensym.jgi.G2Gateway, the connectionClassName property must be com.gensym.jgi.G2Gateway.

Instead, to create a G2 JavaLink connection type, create a com.gensym.jgi.G2ConnectionInfo, using the default values for these properties, and pass this object to the getOrMakeConnection method of a com.gensym.jgi.G2Gateway.

The section "Choosing a Connection Type" on page 30 explains the two available connection types and the functionality that each provides.

Thus, the only time that you would edit the connection and interface classes of a connection information object is if you have done either of the following, which is rare:

- Provided your own implementation of either the TwAccess or G2Access interfaces.
- Subclassed either gsi-interface or ui-client-session.

To create a G2 JavaLink connection type:

```
//Create a G2 JavaLink connection type
G2ConnectionInfo connectionInfo = new G2ConnectionInfo
   (null, "myhost", "1114")
```

This constructor uses the following default values for the connection and interface class properties:

Property	Value
connectionClassName	com.gensym.jgi.G2Gateway
gsiInterfaceClassName	GSI-INTERFACE

Note The string value of the gsiInterfaceClassName property must always be uppercase, as must all class and attribute name specifications that you enter are arguments to TW2 Toolkit classes, unless an item name has been explicitly created in lowercase, using escape characters in G2.

Changing the Interface Name

The interfaceName property sets the name of the ui-client-interface instance that G2 creates at the time a TW2 Toolkit client makes a successful connection. By default, the TwConnector component creates an instance named tw-connector-bean-interface, and calls to the TwGateway.openConnection static method create instances named no-name.

If you want the interface item in G2 to have a particular name, other than the default, set this value as part of the constructor.

Sharing a Connection

The `sharable` property indicates whether the connection to G2 can be shared with other components. For Telewindows2 Toolkit connection types, the default for this property is `false`, which means connections are not shared. Contrast this with G2 JavaLink connection types, which are shared, by default.

When a connection is not shared, subsequent connection requests to the same host and port of the unshared connection cannot use the existing connection. This means that each client connects to G2 under a separate login session and is assumed to require full control over the connection.

The `forceNew` property specifies that each time a component attempts to connect to G2, a new connection is made, even if a connection already exists with the current connection information and even if that connection is specified as sharable. For TW2 Toolkit connections, the default of this property is `true`, which means that each TW2 Toolkit client always creates a new connection, regardless of the value of the `sharable` property. Contrast this with G2 JavaLink connection types, which do not force a new connection, by default.

Using a Shared Connection

Shared connections are useful in situations where applications:

- Do not require continuous access to G2, such as agents, which simply need to connect to G2 to obtain data, then continue processing.
- Adhere to some agreement about when and how to close a connection.

As mentioned above, G2 JavaLink connections are shared, which means they do not force a new connection to G2. Thus, multiple calls to the `getOrMakeConnection` method on a `G2Gateway` have access to the same connection, if one exists.

A shared connection might be applicable for a TW2 Toolkit application that connects to G2 with a single `TwConnector` component, then uses multiple `ItemRetriever` components to retrieve items. In such an application, the client could establish a shared connection through the `TwConnector`. Once a connection to G2 existed, each `ItemRetriever` whose host, port, and URL settings match those of the existing connection would share that connection.

Internally, the `ItemRetriever` calls the `getOrMakeConnection` method on a `G2Gateway` to create a connection. Each time the `ItemRetriever` attempts to retrieve an item, TW2 Toolkit compares all of the information contained in the connection information object that is passed to the calling method to see if its values match those of an existing `G2Gateway` object. If the values match, G2 JavaLink returns the connection object to the calling method. If no exact match exists between the connection information object passed to the method and the existing connection, G2 JavaLink establishes a new connection to G2.

When a client application creates a G2 connection, that application can close the connection at any time. If the connection is shared, other components requesting an identical G2 connection will share the existing connection and, in doing so, will also gain control over its existence. When one component closes a shared connection, all others sharing that connection are disconnected, too.

Creating a Shared Telewindows2 Toolkit Connection

If your application requires a shared connection, you must change both the `sharable` and the `forceNew` properties.

To create a shared TW2 Toolkit connection type:

➔ Create a `TwConnectionInfo` and set these properties:

Property	Value
`forceNew`	`false`
`sharable`	`true`

For example, this code example uses a constructor that lets you change both of these properties:

```
//Create a shared connection
TwConnectionInfo connectionInfo = new TwConnectionInfo
   (null, "myhost", "1114", false, true);
```

Understanding How These Properties Interact

Because you must set two properties to determine whether a connection is shared, situations can arise in which the `sharable` and `forceNew` properties seem to conflict. For example, you might make a connection request to an existing connection whose `sharable` property is `true`, using a connection information object that specifies its `forceNew` property as `true`. In this case, the connection request forces a new connection, regardless of the existence of the sharable connection with the same connection information. Similarly, you might make a connection request to an existing connection whose `sharable` property is `false`, using a connection information object that specifies its `forceNew` property as `false`. In this case, the connection request creates a new connection, regardless of the specified connection information. Thus, when setting these properties explicitly, keep in mind that:

- `forceNew = true` always overrides `sharable = true`.
- `sharable = false` always overrides `forceNew = false`.

Setting a Permanent Connection

The `permanent` property specifies that the connection survives a G2 reset. The default is `true` for both Telewindows2 Toolkit and G2 JavaLink connection types.

When the `permanent` property is set to `Boolean.TRUE`, if the KB is reset, G2:

- Keeps the client connection open.
- Keeps the ui-client-interface or gsi-interface instances.

Setting this property to `Boolean.FALSE` causes G2 to break the connection and delete the interface object during a KB reset.

Note This property maps to the G2 Gateway `gsi_initiate_connection` API function `keep_connection` argument, described in the *G2 Gateway Bridge Developer's Guide*.

Specifying a Logical Name

The `logicalName` property specifies an alias for a shared connection to G2. To use this feature, specify the `logicalName`, `host`, and `port` properties in one `TwConnector`, then specify the same `logicalName` for any other `TwConnector`.

Once a connection to the specified host and port exists, any additional `TwConnector` components uses the existing connection, ignoring their own `host` and `port` properties.

The use of logical names only works if the connection to the specified host and port is shared, as described in "Sharing a Connection" on page 69.

Setting a Remote Procedure Invocation String

The `rpis` property lets you specify a remote procedure initialization string.

The default value is `null`.

6

Using TwGateway

Describes how to use the TwGateway class to create a connection in a Java application.

Introduction

You use the `TwGateway` class for connecting a client application to G2 to gain access to Telewindows2 (TW2) Toolkit functionality. The `TwGateway` class includes a three versions of this static method for connecting to G2:

```
TwGateway.openConnection
```

The `TwGateway` class inherits its definition from `com.gensym.jgi.G2Gateway`, which means your client application can:

- Connect to a single G2 process at a time.
- Send data to and receive data from a G2 process.

- Support 2- and 3-tier connections.
- Make RPC calls.
- Subscribe to G2 state events.

In addition, `TwGateway` lets your application perform these tasks:

- Establish a login session with G2.
- Get and set user modes.
- Get user menu choices.
- View workspaces.
- Subscribe to workspace show and hide events and KB events.

This chapter presents information about using the `TwGateway` class to connect to G2 and accomplish many TW2 Toolkit tasks. The chapter includes these major sections:

- Packages covered.
- Creating a connection.
- Handling connection events and working with the connection.
- Reference information for the `TwGateway` class's methods.

Packages Covered

com.gensym.ntw

Interfaces

```
TwAccess
TwCallbacks
TwConnection
TwConnectionListener
WorkspaceShowingListener
```

Classes

```
TwConnectionAdapter
TwConnectionEvent
TwGateway
WorkspaceShowingEvent
```

com.gensym.ntw.util

Interfaces

KbMessageListener
KbModuleListener

Classes

KbEvent
KbModuleAdapter
NtwEventMulticaster

com.gensym.jgi

Exceptions

G2AccessException
ConnectionTimedOutException
G2CommunicationException

Relevant Demos

The TW2 Toolkit ItemAccessDemo illustrates a simple use of the TwGateway class to connect to G2 to obtain an item. The demo is located in this directory, depending on your platform:

NT: `%SEQUOIA_HOME%\classes\com\gensym\demos\itemaccessdemo\ItemAccessDemo.java`

UNIX: `$SEQUOIA_HOME/classes/com/gensym/demos/itemaccessdemo/ItemAccessDemo.java`

In addition, the Java applications that this chapter uses are available online in this directory, depending on your platform:

NT: `%SEQUOIA_HOME%\classes\com\gensym\demos\docs\connectivitydemos\*.java`

UNIX: `$SEQUOIA_HOME/classes/com/gensym/demos/docs/connectivitydemos/*.java`

The filenames correspond to the class names in each example in this chapter.

To run these demos, you must load this KB file into G2, depending on your platform:

NT: `%SEQUOIA_HOME%\kbs\ connectivity-demos.kb`

UNIX: `$SEQUOIA_HOME/kbs/ connectivity-demos.kb`

Supporting a Middle-Tier Connection

Before establishing a connection to G2, you can set up and use a middle-tier server. For information about how to do this, see Chapter 8, "Using a Middle-Tier Server" on page 117.

Creating a G2 Connection

When you create a connection to G2, using the `TwGateway.openConnection` static method, the method returns a connection object. The class of the returned object is a `TwAccess`, which is an interface.

To create a single G2 connection, you must:

- Open the connection to G2.
- Log in to G2.
- Handle exceptions.

Your application might also need to get or set the current user mode. For information about how to do this, see "Working with User Modes in a TwGateway Connection" on page 112.

The following sections explain these steps in detail and provide examples.

Opening and Closing a Connection

You can call any of these static methods on a `TwGateway` to open a connection:

- `openConnection(String host, String port)`
- `openConnection(String brokerURL, String host, String port)`
- `openConnection(TwConnectionInfo info)`

Typically, when opening a G2 connection, you declare a private variable that holds the connection to be a `TwAccess`. If the methods you are calling on the returned connection are not defined by `TwAccess`, you must declare the variable to be a `TwGateway` and cast the result of the `openConnection` call to a `TwGateway`.

To create a connection, you must also log in to G2 by calling the `login` on the returned connection. Logging in is required regardless of whether you are

connecting to a non-secure or secure G2. For more information about login sessions, including how to establish a login to a secure G2, see Chapter 7, "Establishing a G2 Login Session" on page 101.

To log out and close a connection, you call this G2 JavaLink method on a `G2Gateway`:

```
closeConnection
```

You can also log out and leave the connection open by calling the `logout` method on a `TwGateway`. To do this, you must declare the connection variable to be a type of `G2Gateway` and cast the return value of the `openConnection` method to a `G2Gateway`, because `logout` is defined on `TwGateway`, not `TwAccess`. For more information, see "Logging Out From G2" on page 113.

The following code fragments open a connection to a particular host and port, then log out and close the connection. For information on creating a connection information object as the argument to the `openConnection` method, see Chapter 5, "Using Connection Information Objects" on page 61.

To open a connection to G2:

```
//Declare connection variable
private static myConnection = TwAccess;

//Call static method on TwGateway to open connection
myConnection = TwGateway.openConnection("myhost", "1234");

//log in to G2
myConnection.login();
```

To close a connection:

```
//Declare connection variable
private static myConnection = TwAccess;

//Call static method on TwGateway to open connection
myConnection = TwGateway.openConnection("myhost","1234");

//log in to G2
myConnection.login();

//Close connection
myConnection.closeConnection();
```

The following example defines a connection class and the `runMyConnection` method, which creates a connection, logs in to G2, sends a message to the G2 Message Board, and handles exceptions. The `closeConnection` method logs out and closes the connection. The example prints messages to the command window to verify the code.

```
package com.gensym.demos.docs.connectivitydemos;

import com.gensym.ntw.TwGateway;
import com.gensym.ntw.TwAccess;
import com.gensym.jgi.G2AccessException;
import com.gensym.jgi.ConnectionTimedOutException;
import com.gensym.jgi.G2CommunicationException;

public class MyConnection {
  private static TwAccess myConnection;

  //Create a connection
  public static void runMyConnection () {
    try{
      //Establish a connection to G2
      myConnection = TwGateway.openConnection("localhost", "1111");
      myConnection.login();
      System.out.println("Connected to G2!");
      myConnection.returnMessage("Connected to G2!");
    } catch (G2AccessException e){
      e.printStackTrace();
      System.exit (-1);
    }
  }

  // MAIN
  public static void main(String[] args){
    System.out.println("Got to main!");
    runMyConnection();
    myConnection.closeConnection();
    System.out.println("Connection closed!");
    System.out.println ("End of Main!");
    System.exit (0);
  }
}
```

Running this example produces these messages in the command window, assuming you have a G2 running on your local machine on port 1111:

```
Got to main!
Connected to G2!
Connection closed!
End of Main!
```

Handling Connection Exceptions

Using the `TwGateway.openConnection` static method to connect to G2 can generate these exceptions:

Exception	Description
`G2AccessException`	An error occurred during the connection attempt.
`ConnectionTimedOutException`	An error occurred during the connection initialization phase of connecting to G2.
`G2CommunicationException`	Signals that a G2 communication problem has occurred.

All of the connection exceptions are generated by `G2Gateway`. For more information, refer to the G2 JavaLink API documentation for the `com.gensym.jgi` package.

Handling Events

Once a connection to G2 exists, you can subscribe to the following connection and KB events:

To subscribe to...	Call this method...	On an implementation of this interface...
Connection events	`addTwConnectionListener`	`com.gensym.ntw.TwAccess`
Programmatic show and hide workspace events	`addWorkspaceShowingListener`	`com.gensym.ntw.TwAccess`
KB module events	`addKbModuleListener`	`com.gensym.classes.Kb`
KB message events	`addKbMessageListener`	`com.gensym.classes.Kb`

Subscribing to Connection Events

You subscribe to connection events by adding a `TwGateway` as a `com.gensym.ntw.TwConnectionListener` or using a `TwConnectionAdapter`.

Both the `G2Gateway` and `TwGateway` classes generate connection events.

These are the events about which you can receive notification through the `addTwConnectionListener` subscription method:

Event	G2Gateway	TwGateway
`communicationError`	✓	
`g2ConnectionClosed`	✓	
`g2ConnectionInitialized`	✓	
`g2IsPaused`	✓	
`g2IsResumed`	✓	
`g2IsReset`	✓	
`g2IsRestarted`	✓	
`g2MessageReceived`	✓	
`loggedIn`		✓
`loggedOut`		✓
`readBlockage`	✓	
`userModeChanged`		✓
`writeBlocakge`	✓	

The following code fragments show the basic steps for subscribing to connection events.

To subscribe to connection events:

```
//Declare connection variable
private static TwAccess myConnection;

//Open connection
myConnection = TwGateway.openConnection ("myhost", "1234");

//Add connection as a listener for connection events
myConnection.addTwConnectionListener(this);
```

This example adds the application as a listener for connection events in the constructor and prints a message to the command window when the user logs in. The `main` method opens the connection, calls the constructor, which adds the application as a listener, logs in, then closes the connection.

Note The `g2ConnectionClosed` event serves as notification for both closing the connection and logging out; thus, the `loggedOut` event is generated only when you explicitly log out without closing the connection.

```
package com.gensym.demos.docs.connectivitydemos;

import com.gensym.ntw.TwGateway;
import com.gensym.ntw.TwAccess;
import com.gensym.ntw.TwConnectionListener;
import com.gensym.ntw.TwConnectionEvent;
import com.gensym.jgi.G2CommunicationErrorEvent;
import com.gensym.jgi.G2ConnectionEvent;

public class MyConnectionListener implements TwConnectionListener {
  private static TwAccess myConnection;

  public MyConnectionListener () {
    //Add connection as a listener for connection events
    myConnection.addTwConnectionListener(this);
    System.out.println ("Added as listener!");
  }

  public static void openConnection () {
    try{
      myConnection = TwGateway.openConnection ("localhost", "1111");
    } catch (Exception e){
      e.printStackTrace();
    }
  }

  public static void loginConnection () {
    try{
      myConnection.login();
    } catch (Exception e){
      e.printStackTrace();
    }
  }

  //TwConnectionListener Methods
  public void loggedIn (TwConnectionEvent e){
    System.out.println ("Logged in event detected!");
  }

  public void loggedOut (TwConnectionEvent e) {}
  public void userModeChanged (TwConnectionEvent e) {}

  //G2ConnectionListener Methods
  public void communicationError(G2CommunicationErrorEvent e){}
  public void g2ConnectionClosed(G2ConnectionEvent e) {
    System.out.println ("Connection closed!");
  }
  public void g2ConnectionInitialized(G2ConnectionEvent e) {}
  public void g2IsPaused(G2ConnectionEvent e) {}
  public void g2IsReset(G2ConnectionEvent e) {}
  public void g2IsResumed(G2ConnectionEvent e) {}
```

```
  public void g2IsStarted(G2ConnectionEvent e) {}
  public void g2MessageReceived(G2ConnectionEvent e) {}
  public void readBlockage(G2ConnectionEvent e) {}
  public void writeBlockage(G2ConnectionEvent e) {}

  public static void main(String[] args){
    System.out.println("Got to main!");
    openConnection();
    MyConnectionListener app = new MyConnectionListener();
    loginConnection();
    myConnection.closeConnection();
    System.out.println ("End of Main!");
    System.exit (0);
  }
}
```

Running this example produces these messages in the command window, assuming you have a G2 running on your local machine on port 1111:

```
Got to main!
Added as listener!
Logged in event detected!
Connection closed!
End of Main!
```

Subscribing to Workspace Show and Hide Events

When G2 programmatically shows or hides a KB workspace, the connection generates an event. You can subscribe to these events by adding a `TwGateway` as a `com.gensym.ntw.WorkspaceShowingListener`. Only `TwGateway` connections generate these events.

These are the events about which you can receive notification through the `addWorkspaceShowingListener` subscription method:

- `hideWorkspace`
- `showWorkspace`

To subscribe to programmatic workspace show and hide events:

```
//Declare connection variable
private static TwAccess myConnection;

//Open connection
myConnection = TwGateway.openConnection ("myhost", "1234");

//Add connection as a listener for workspace showing events
myConnection.addTwConnectionListener(this);
```

This example makes RPC calls to two user-defined G2 procedures named show-wksp and hide-wksp, which show and hide a named workspace in the `connectivity.kb`. The `main` method opens the connection, calls the constructor, which adds the application as a listener for workspace showing events, then calls each method that makes an RPC call to show and hide the workspace. The `main` method must wait before closing the connection in order to detect the show and hide workspace events.

```
package com.gensym.demos.docs.connectivitydemos;

import com.gensym.ntw.TwGateway;
import com.gensym.ntw.TwAccess;
import com.gensym.ntw.WorkspaceShowingListener;
import com.gensym.ntw.WorkspaceShowingEvent;
import com.gensym.util.Symbol;
import com.gensym.classes.Object;

public class MyWorkspaceShowingListener implements
WorkspaceShowingListener {
  private static TwAccess myConnection;
  private static Symbol SHOW_WKSP_ = Symbol.intern("SHOW-WKSP");
  private static Symbol HIDE_WKSP_ = Symbol.intern("HIDE-WKSP");
  private static Object[] args = new Object[]{};

  public MyWorkspaceShowingListener() {
    try{
      //Add connection as a workspace showing listener
      myConnection.addWorkspaceShowingListener(this);
      System.out.println ("Added as listener!");
    } catch (Exception e){
      e.printStackTrace();
    }
  }

  public static void runMyConnection () {
    try{
      myConnection = TwGateway.openConnection ("localhost", "1111");
      myConnection.login();
    } catch (Exception e){
      e.printStackTrace();
    }
  }

  public static void showWksp () {
    try {
      myConnection.callRPC(SHOW_WKSP_, args);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
```

```
  public static void hideWksp () {
    try {
      myConnection.callRPC(HIDE_WKSP_, args);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  //WorkspaceShowingListener methods
  public void hideWorkspace (WorkspaceShowingEvent e){
    System.out.println ("Workspace hidden!");
  }
  public void showWorkspace (WorkspaceShowingEvent e){
    System.out.println ("Workspace shown!");
  }

  public static void main(String[] args){
    System.out.println("Got to main!");
    runMyConnection();
    MyWorkspaceShowingListener app = new MyWorkspaceShowingListener();
    try {
      showWksp();
      hideWksp();
      Thread.sleep(500);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
    myConnection.closeConnection();
    System.out.println ("Connection closed!");
    System.out.println ("End of Main!");
    System.exit (0);
  }
}
```

Running this example produces these messages in the command window, assuming you have a G2 running on your local machine on port 1111:

```
Got to main!
Added as listener!
Workspace shown!
Workspace hidden!
Connection closed!
End of Main!
```

Subscribing to KB Module Events

G2 JavaLink provides the `com.gensym.classes.Kb` interface to represent a G2 KB. The KB generates various module events to which you can subscribe by adding the downloaded `Kb` item as a `com.gensym.ntw.util.KbModuleListener` or by using a `KbModuleAdapter`. You get the KB by calling the `getKb` method on a `TwGateway`.

These are the events about which you can receive notification through the `addKbModuleListener` subscription method:

- `kbCleared`
- `moduleCreated`
- `moduleDeleted`
- `moduleDependencyChanged`
- `moduleNameChanged`
- `receivedInitialModules`
- `topLevelWorkspaceAdded`
- `topLevelWorkspaceDeleted`

To subscribe to KB module events:

```
//Declare connection and KB variables
private static TwAccess myConnection;
private static Kb myKb;

//Open connection
myConnection = TwGateway.openConnection ("myhost", "1234");

//Get KB from connection
Kb myKb = myConnection.getKb();

//Add KB as a listener for module events
myKb.addKbModuleListener(this)
```

The following example adds the application as a listener for KB module events in the constructor and prints a message to the command window when the listener is initially added.

```
package com.gensym.demos.docs.connectivitydemos;

import com.gensym.ntw.TwGateway;
import com.gensym.ntw.TwAccess;
import com.gensym.classes.Kb;
import com.gensym.ntw.util.KbModuleListener;
import com.gensym.ntw.util.KbEvent;

public class MyKbModuleListener implements KbModuleListener {
  private static TwAccess myConnection;

  //Variable for KB
  private static Kb myKb;

  private MyKbModuleListener () {
    //Get KB associated with connection
    Kb myKb = myConnection.getKb();
```

```
    try {
      //Add KB as a listener for module events
      myKb.addKbModuleListener(this);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public static void runMyConnection () {
    try{
      myConnection = TwGateway.openConnection ("localhost", "1111");
      myConnection.login();
    } catch (Exception e){
      e.printStackTrace();
    }
  }

  //KbModuleListener Methods

  public void receivedInitialModules(KbEvent e) {
    //Invoked when KbModuleListener added
    System.out.println ("Received initial modules!");
  }

  public void kbCleared (KbEvent e) {}
  public void moduleCreated(KbEvent e) {}
  public void moduleDeleted(KbEvent e) {}
  public void moduleDependencyChanged(KbEvent e) {}
  public void moduleNameChanged(KbEvent e) {}
  public void topLevelWorkspaceAdded(KbEvent e) {}
  public void topLevelWorkspaceDeleted(KbEvent e) {}

  public static void main(String[] args){
    System.out.println("Got to main!");
    runMyConnection();
    MyKbModuleListener app = new MyKbModuleListener();
    myConnection.closeConnection();
    System.out.println ("Connection closed!");
    System.out.println ("End of Main!");
    System.exit (0);
  }
}
```

Running this example produces these messages in the command window, assuming you have a G2 running on your local machine on port 1111:

```
Got to main!
Received initial modules!
Connection closed!
End of Main!
```

Subscribing to KB Message Events

Implementations of the G2 JavaLink `com.gensym.classes.Kb` interface generate events when the G2 KB generates an Operator Logbook or Message Board message. You can subscribe to these events by adding the downloaded `Kb` item as a `com.gensym.ntw.util.KbMessageListener`. You get the KB by calling the `getKb` method on a `TwGateway`.

These are the events about which you can receive notification through the `addKbMessageListener` subscription method:

- `kbMessageAdded`
- `kbMessageDeleted`
- `receivedInitialContents`

To subscribe to KB message events:

```
//Declare connection and KB variables
private static TwAccess myConnection;
private static Kb myKb;

//Open connection
myConnection = TwGateway.openConnection ("myhost", "1234");

//Get KB from connection
Kb myKb = myConnection.getKb();

//Add KB as a listener for module events
myKb.addKbMMessageistener(this)
```

The following example adds the application as a listener for KB message events in the constructor and prints a message to the command window when the listener is initially added.

```
package com.gensym.demos.docs.connectivitydemos;

import com.gensym.ntw.TwGateway;
import com.gensym.ntw.TwAccess;
import com.gensym.classes.Kb;
import com.gensym.ntw.util.KbMessageListener;
import com.gensym.ntw.util.KbEvent;

public class MyKbMessageListener implements KbMessageListener {
  private static TwAccess myConnection;

  //Variable for KB
  private static Kb myKb;

  private MyKbMessageListener () {
    //Get KB associated with connection
    Kb myKb = myConnection.getKb();
```

```
    try {
      //Add KB as a listener for module events
      myKb.addKbMessageListener(this);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public static void runMyConnection () {
    try{
      myConnection = TwGateway.openConnection ("localhost", "1111");
      myConnection.login();
    } catch (Exception e){
      e.printStackTrace();
    }
  }

  //KbMessageListener Methods

  public void receivedInitialContents(KbEvent e) {
    //Invoked when KbMessageListener added
    System.out.println ("Received initial contents!");
  }

  public void kbMessageAdded (KbEvent e) {}
  public void kbMessageDeleted (KbEvent e) {}

  public static void main(String[] args){
    System.out.println("Got to main!");
    runMyConnection();
    MyKbMessageListener app = new MyKbMessageListener();
    myConnection.closeConnection();
    System.out.println ("Connection closed!");
    System.out.println ("End of Main!");
    System.exit (0);
  }
}
```

Running this example produces these messages in the command window, assuming you have a G2 running on your local machine on port 1111:

```
Got to main!
Received initial contents!
Connection closed!
End of Main!
```

Working with Telewindows2 Toolkit Connections

Once your application creates a Telewindows2 Toolkit connection and establishes a login session with the G2 process, you can perform numerous tasks available through that connection. The G2 JavaLink and TW2 Toolkit API documentation presents all the methods you can call. This section describes how to perform the following common tasks:

- Getting the KB.
- Getting a list of named workspaces.
- Getting the `DialogManager` associated with the connection.
- Getting and invoking user menu choices.
- Sending a message to G2.
- Getting and setting attributes of user-defined items.

As mentioned earlier, the `TwGateway.openConnection` static method returns a `TwAccess`. If the methods you are calling on the returned connection are defined on `TwAccess`, then you do not need to cast the result to an instance of a `TwGateway`. However, if the methods calls are on `TwGateway` only, then you must cast the connection object to `TwGateway`.

Getting the KB

To receive notification of KB module and message events, you must first get the `com.gensym.classes.Kb` associated with the connection. For information on subscribing to these events, see "Subscribing to KB Module Events" on page 84 and "Subscribing to KB Message Events" on page 87.

To get the KB associated with a connection:

```
//Declare connection and KB variables
private static TwAccess myConnection;
private static Kb myKb;

//Open connection
myConnection = TwGateway.openConnection ("myhost", "1234");

//Get KB from connection
Kb myKb = myConnection.getKb();
```

Getting a List of Named Workspaces

You can use a current connection to obtain a list of named KB workspaces. The returned value is a `com.gensym.util.Sequence`.

When you get a list of named workspaces from a connection, you need to handle this exception:

```
com.gensym.jgi.G2AccessException
```

To get a list of workspaces:

```
//Declare connection and workspace variables
private static TwAccess myConnection;
private static Sequence workspaces;

//Open connection
myConnection = TwGateway.openConnection ("myhost", "1234");

//Get named workspaces from connection
workspaces = myConnection.getNamedWorkspaces();
```

For example, this example prints the current named workspaces to the command window:

```
package com.gensym.demos.docs.connectivitydemos;

import com.gensym.ntw.TwGateway;
import com.gensym.ntw.TwAccess;
import com.gensym.util.Sequence;
import com.gensym.jgi.G2AccessException;

public class MyWorkspaces {
  private static TwAccess myConnection;

  //Variable for workspaces
  private static Sequence workspaces;

  public static void runMyConnection () {
    try{
      myConnection = TwGateway.openConnection ("localhost", "1111");
      myConnection.login();
    } catch (Exception e){
      e.printStackTrace();
    }
  }
```

```
  public static void getNamedWorkspaces() {
    try {
      //Get named workspaces from connection
      workspaces = myConnection.getNamedWorkspaces();
      System.out.println (workspaces);
    } catch (G2AccessException e){
      e.printStackTrace();
    }
  }

  public static void main(String[] args){
    System.out.println("Got to main!");
    runMyConnection();
    getNamedWorkspaces();
    myConnection.closeConnection();
    System.out.println ("Connection closed!");
    System.out.println ("End of Main!");
    System.exit (0);
  }
}
```

Running this example produces these messages in the command window, assuming you have a G2 running on your local machine on port 1111 with the `connectivity-demos.kb` loaded:

```
Got to main!
[WKSP-1, WKSP-2, INVOKE-USER-MENU-CHOICE, WORKSPACE-SHOWING-LISTENER]
Connection closed!
End of Main!
```

Getting the Current DialogManager

Every TW2 Toolkit connection has an associated `com.gensym.dlgruntime.DialogManager`, which is responsible for managing all the item properties dialogs in the KB.

The connection might also have an associated `com.gensym.dlgruntime.DialogManagerFactory`, which you can implement to generate subclasses of `DialogManager` for managing different types of dialog resources.

To get the `DialogManagerFactory`, you must cast the connection to a `TwGateway` because `getDialogManagerFactory` is not defined on `TwAccess`.

For more information about `DialogManager` and `DialogManagerFactory`, see Chapter 16, "Launching Custom Item Properties Dialogs" on page 305.

To obtain the current `DialogManager`:

```
//Declare connection and DialogManager variables
private static TwAccess myConnection;
private static DialogManager dialogManager;

//Open connection
myConnection = TwGateway.openConnection ("myhost", "1234");

//Get DialogManager
dialogManager = myConnection.getDialogManager();
```

For example, this example prints the current `DialogManager` to the command window:

```
package com.gensym.demos.docs.connectivitydemos;

import com.gensym.ntw.TwGateway;
import com.gensym.ntw.TwAccess;
import com.gensym.dlgruntime.DialogManager;

public class MyDialogManager {
  private static TwAccess myConnection;

  //Variable for DialogManager
  private static DialogManager dialogManager;

  public static void runMyConnection () {
    try{
      myConnection = TwGateway.openConnection ("localhost", "1111");
      myConnection.login();
    } catch (Exception e){
      e.printStackTrace();
    }
  }

  public static void getDialogManager () {
    //Get DialogManager
    dialogManager = myConnection.getDialogManager();
    System.out.println (dialogManager);
  }

  public static void main(String[] args){
    System.out.println("Got to main!");
    runMyConnection();
    getDialogManager ();
    myConnection.closeConnection();
    System.out.println ("Connection closed!");
    System.out.println ("End of Main!");
    System.exit (0);
  }
}
```

Running this example produces these messages in the command window, assuming you have a G2 running on your local machine on port 1111 :

```
Got to main!
com.gensym.dlgruntime.DialogManager@191e74
Connection closed!
End of Main!
```

Invoking a User Menu Choice

You can get a user menu choice associated with a particular class from the current connection. Once you have a handle on the user menu choice, you can call the `invoke` method on a `com.gensym.classes.UserMenuChoice` to invoke the user menu choice across the connection.

When you get a user menu choice from a connection, you must handle this exception:

```
com.gensym.jgi.G2AccessException
```

To get a user menu choice:

1. Create an upper-case symbol to represent the class name of the user menu choice.

   To do this, we recommend that you create a `private static final` variable that is a `com.gensym.util.Symbol`, then call the `intern` method on the `Symbol` to convert the G2 class name as an upper-case string to a symbol. For example:

   ```
   private static final Symbol PLANET_ = Symbol.intern("PLANET");
   ```

2. Create an upper-case symbol to represent the label of the user menu choice.

   For example:

   ```
   private static final Symbol COMPUTE_ORBIT_ =

      Symbol.intern("COMPUTE-ORBIT");
   ```

3. Establish a connection to G2 and make a login request.

4. Call this method on the connection to get the user menu choice:

   ```
   getUserMenuChoice(Symbol label, Symbol applicableClass)
   ```

For example, this example gets and invokes the user menu choice named compute-orbit on the planet class:

```
package com.gensym.demos.docs.connectivitydemos;

import com.gensym.ntw.TwGateway;
import com.gensym.ntw.TwAccess;
import com.gensym.classes.UserMenuChoice;
import com.gensym.util.Symbol;
import com.gensym.classes.Item;

public class MyUserMenuChoice {
  private static TwAccess myConnection;

  //Variable for UserMenuChoice
  private static UserMenuChoice userMenuChoice;

  //Variable for user menu choice label
  private static final Symbol COMPUTE_ORBIT_ =
       Symbol.intern("COMPUTE-ORBIT");

  //Variable for user menu choice applicable class
  private static final Symbol PLANET_ = Symbol.intern("PLANET");

  //Variable for Planet item as a symbol
  private static final Symbol MY_PLANET_ = Symbol.intern("MY-PLANET");

  //Variable for Planet item
  private static Item myPlanet;

 public static void runMyConnection () {
    try{
      myConnection = TwGateway.openConnection ("localhost", "1111");
      myConnection.login();
    } catch (Exception e){
      e.printStackTrace();
    }
 }

  public static void runUserMenuChoice () {
    try{
      //Get UserMenuChoice
      userMenuChoice = myConnection.getUserMenuChoice(COMPUTE_ORBIT_,
                                                      PLANET_);
      System.out.println ("Got user menu choice!");

      //Get Planet item
      myPlanet = myConnection.getUniqueNamedItem(PLANET_, MY_PLANET_);
      System.out.println ("Got unique named item!");
```

```
//Invoke user menu choice on item
      userMenuChoice.invoke(myPlanet);
      System.out.println ("Invoked user menu choice!");

    } catch (Exception e){
      e.printStackTrace();
    }
  }

  public static void main(String[] args){
    System.out.println("Got to main!");
    runMyConnection();
    runUserMenuChoice();
    myConnection.closeConnection();
    System.out.println ("Connection closed!");
    System.out.println ("End of Main!");
    System.exit (0);
  }
}
```

Running this example produces these messages in the command window, assuming you have a G2 running on your local machine on port 1111 :

```
Got to main!
Got user menu choice!
Got unique named item!
Invoked user menu choice!
Connection closed!
End of Main!
```

Sending a Message

You can send messages to the G2 Message Board across a connection by calling the `returnMessage` method on a `com.gensym.jgi.G2Gateway`.

To receive messages, you register as a `KbMessageListener`, as described in "Subscribing to KB Message Events" on page 87.

To send a message to the G2 Message Board:

```
//Declare connection variable
private static TwAccess myConnection;

//Open connection
myConnection = TwGateway.openConnection ("myhost", "1234");

//Send message
myConnection.returnMessage("Connected to G2!");
```

For a complete example, see "Opening and Closing a Connection" on page 76.

Getting and Setting Attributes of User-Defined Items

You can use the G2 JavaLink Download Interfaces wizard to generate a Java class representation of any G2 item. When you download interfaces, G2 JavaLink generates two classes:

- An interface that defines accessor methods for getting and setting all attributes of the class, plus methods for all the G2 methods defined for the class.
- An implementation of the downloaded interface that represents the instance.

Once you have downloaded interfaces, you can get a handle on the instance, and use the accessor methods to get and set attributes of the item. To do this, you establish a connection and call this method on a `com.gensym.jgi.G2Gateway`:

```
getUniqueNamedItem (Symbol itmClass, Symbol itmName)
```

To download interfaces, you run `DownloadInterfaces` from the `bin` directory of your G2 JavaLink product directory. If you are running G2 JavaLink on an NT platform, you can also run Download Interfaces from the Start menu. For details, see the *G2 DownloadInterfaces User's Guide*.

To get and set attributes of a user-defined G2 item:

1. Download interfaces for the item whose attributes you wish to get or set.
2. Import the downloaded interface into your application.
3. Create an upper-case symbol to represent the class name of the downloaded interface.

   To do this, we recommend that you create a `private static final` variable that is a `com.gensym.util.Symbol`, then call the `intern` method on the `Symbol` to convert the G2 class name as an upper-case string to a symbol. For example:

   ```
   private static final Symbol PLANET_ = Symbol.intern("PLANET");
   ```

4. Establish a connection to G2.

   Note that in this example, it is not necessary to log in to G2 because you are only getting and setting attributes of an item; you are not displaying the item in a workspace view.

5. Call the `getUniqueNamedItem` method on a `G2Gateway` to get the item from the connection, providing as arguments the class name and the item name as symbols.
6. Cast the result of the unique named item to the appropriate downloaded interface to make its accessor methods available.
7. Call the accessor methods on the result of the cast to get and set attribute values of the item.

The following example shows how code fragments for getting and setting values for the number-of-moons attribute of a user-defined planet class. To do this, you must first download interfaces for this class. For the location of the complete demo, see "Relevant Demos" on page 75.

```
//Import downloaded interface
import com.gensym.demos.democlasses.Planet;

//Connection variable
private static TwAccess myConnection;

//Variable for downloaded interface
private Planet planet;

//Variable for item name
private Symbol planetName_;

//Variable for class name
private static final Symbol PLANET_ = Symbol.intern("PLANET");

//Establish connection and make login request
myConnection = TwGateway.openConnection("localhost", "1111");
myConnection.login();

//Obtain a stub for the G2 instance
Object result = myConnection.getUniqueNamedItem(PLANET_, planetName_);

//Cast stub to appropriate interface
planet = (Planet)result;

//Use accessor method on interface to get attribute value
int numberOfMoons = planet.getNumberOfMoons();

//Use accessor method on interface to set attribute value
planet.setNumberOfMoons(numberOfMoons);
```

TwGateway Reference

`TwGateway` extends `com.gensym.jgi.G2Gateway`. Refer to the G2 JavaLink API documentation for that class for additional methods.

Because the `TwGateway.openConnection` static method returns a `com.gensym.ntw.TwAccess`, and `TwGateway` implements `TwAccess` through the `TwConnection` interface, this section separates the `TwGateway` methods into:

- The abstract methods on `TwAccess`, all of which `TwGateway` implements.
- The static methods on `TwGateway`.

For a diagram of the class hierarchy of `TwGateway`, see "Class Hierarchy of Connectivity Classes" on page 28.

Abstract Methods on TwAccess

These are the abstract methods, including accessor methods, on `TwAccess`, which `TwGateway` implements:

Method	Description
`login`	Creates a login session to G2, using default login information, unless the method is called with a `com.gensym.ntw.LoginRequest` object. See "Logging in to G2" on page 105.
`logout`	Logs out a currently logged in user and leaves the connection open. See "Logging Out From G2" on page 113.
`addTwConnectionListener` `removeTwConnectionListener`	Adds or removes a `com.gensym.ntw.TwConnectionListener` that is informed when the user logs in, logs out, or changes the user mode.
`addWorkspaceShowingListener` `removeWorkspaceShowingListener`	Adds or removes a `com.gensym.ntw.WorkspaceShowingListener` that is informed when G2 programmatic shows or hides a KB workspace.
`getDialogManager` `setDialogManager`	Gets or sets the `com.gensym.dlgruntime.DialogManager` for the current connection. See Chapter 16, "Launching Custom Item Properties Dialogs" on page 305.
`getKb`	Returns the `com.gensym.classes.Kb` associated with the current KB for this connection.
`getMinimumVersion`	Returns a `com.gensym.jgi.G2Version` that is the earliest G2 version with which this connection is expected to communicate.

Method	Description
`getNamedWorkspaces`	Returns a `com.gensym.util.Sequence` containing the names of all named KB workspaces in the current connection.
`getUserMenuChoice`	Returns a `com.gensym.classes.UserMenuChoice` by specifying a label and applicable class as symbols.
`changeUserMode`	Dynamically changes the user mode of the currently logged in user in G2. Provide a `com.gensym.util.Symbol` as the argument.
`isLoggedIn`	Returns a `boolean` indicating whether the current connection has an active login session with G2. Certain methods on `TwGateway` require that you test whether the connection is currently logged in, such as `changeUserMode`.
`retrieveSession`	Returns the `com.gensym.classes.UiClientSession` for this connection, which has a one-to-one relationship with the connection. For details, see "Creating a Telewindows2 Toolkit Connection" on page 33.
`retrieveUserMode`	Gets the user mode of a currently logged in user.
`toString`	Returns a `java.lang.String` that is a description of this connection.

Static Methods on TwGateway

These are the static methods on `TwGateway`:

Method	Description
`openConnection`	Attempts to open a G2 connection, and if successful, returns a `com.gensym.ntw.TwAccess` interface. You can provide as arguments one of the following: • A `host` and `port` • A `brokerURL`, `host`, and `port` • A `com.gensym.ntw.TwConnectionInfo`
`getDialogManagerFactory` `setDialogManagerFactory`	Gets or sets the `com.gensym.dlgruntime.DialogManagerFactory` for this connection. See Chapter 16, "Launching Custom Item Properties Dialogs" on page 305.

Protected Methods on TwGateway

These are the protected methods of a `TwGateway`, which only subclasses of `TwGateway` can call:

- `initializeLocalRPCs`
- `initializeConnectionRPCs`
- `dispatchTwConnectionEvent`

For details, see the API documentation.

7

Establishing a G2 Login Session

Describes how to establish a login session with G2 after successfully establishing a connection.

gensym

Introduction

After using one of the connectivity classes to establish a connection to a G2 server process, each Telewindows2 (TW2) Toolkit client must establish a login session with G2.

Establishing a login session does two things:

- Creates an instance of a ui-client-session object in G2 that represents the login session and consumes one Telewindows2 Toolkit floating license.
- Makes available the complete functionality of Telewindows2 Toolkit.

You establish login sessions differently depending on the connection class you use:

If you connect to G2 by using a...	Then...
`com.gensym.TwGateway`	You establish a login session with G2 by calling `TwGateway.login` on the returned connection object.
`com.gensym.controls.ItemRetriever` `com.gensym.controls.TwConnector`	TW2 Toolkit calls the `login` method on a `TwGateway` automatically when a connection is made and uses specified login parameters.

The `login` method accepts a `com.gensym.ntw.LoginRequest` object as its argument. This chapter describes the use and purpose of establishing login sessions, and the exceptions that support them.

Packages Covered

com.gensym.ntw

Classes

`LoginRequest`

Exceptions

`AlreadyLoggedInException`
`InvalidUserModeException`
`LoginFailedException`
`NotLoggedInException`
`OkException`
`TooManyLoginAttemptsException`

Relevant Demos

The Java applications that this chapter uses are available online in this directory, depending on your platform:

NT: `%SEQUOIA_HOME%\classes\com\gensym\demos\docs\connectivitydemos\*.java`

UNIX: `$SEQUOIA_HOME/classes/com/gensym/demos/docs/connectivitydemos/*.java`

The filenames correspond to the class names in each example in this chapter.

To run these demos, you must load this KB file into G2, depending on your platform:

NT: `%SEQUOIA_HOME%\kbs\connectivity-demos.kb`

UNIX: `$SEQUOIA_HOME/kbs/connectivity-demos.kb`

Establishing a Login Session

Creating a login session permits a client to access the full functionality of Telewindows2 Toolkit.

Without a valid login session, a client can connect to G2, but it cannot populate a workspace view. This is because displaying a workspace view requires the use of one or more restricted RPC calls that TW2 Toolkit uses internally. For a list of restricted RPC calls and their corresponding public APIs, see Appendix A, "Restricted Remote Procedure Calls" on page 351.

Attempting to display a workspace without first creating a login session results in a `NotLoggedInException` exception, as described in the "Handling Login Exceptions" on page 112.

Managing Clients and Security in G2

While TW2 Toolkit clients must call the `login` method on a `com.gensym.ntw.TwGateway` after a successful connection, G2 itself need not be secure. The reason is that you call the `login` method without any arguments, which uses a `LoginRequest` object with default property values.

Requiring a client login session to connect to a non-secure G2 might seem anomalous, but the TW2 Toolkit behavior corresponds to the way in which G2 handles client connections and their security.

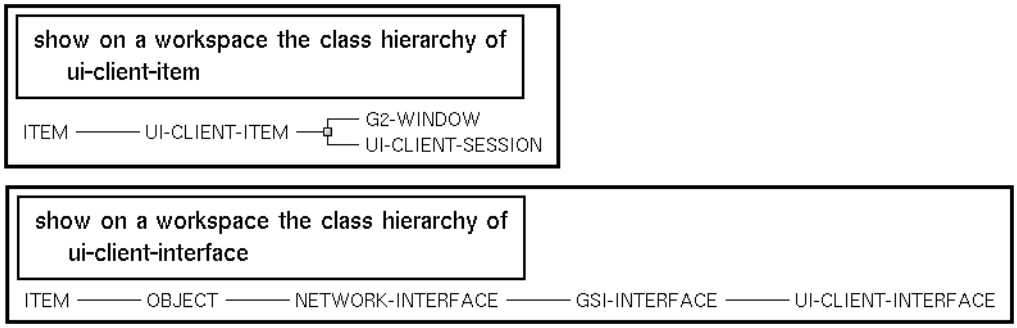
G2 manages security through the OK file with which it is launched. The OK file specifies whether a G2 is secure or not. For a secure G2, the OK file includes user elements consisting of user name, password, and permitted user modes. G2 verifies the user login information from a client against the user elements in the OK file. For a non-secure G2, the OK file specifies that G2 is not secure and contains no user elements.

Representing a Login Session in G2

G2 represents every client connection with one or more object classes, as this table describes:

This type of client...	Creates instances of these classes...
Classic Telewindows	g2-window
Telewindows2 Toolkit	ui-client-interface ui-client-session

g2-window and ui-client-session are both subclasses of ui-client-item, whereas ui-client-interface is a subclass of gsi-interface, as this diagram shows:



As described in "Creating a Telewindows2 Toolkit Connection" on page 33, creating a TW2 Toolkit connection creates an instance of a ui-client-interface.

When a TW2 Toolkit client requests a login session to a non-secure G2, using default user element values, G2 creates a ui-client-session object to represent the user login. That object includes these attributes:

This attribute...	Has this value...
ui-client-session-user-mode	administrator
ui-client-session-user-name	The symbol version of the ui-client-session-user-name-in-operating-system

For a TW2 Toolkit client login session request to a secure G2, these attributes would consist of the user mode, user name, and password properties that the `LoginRequest` object includes.

Two-Tier Connections

In a two-tier connection, a one-to-one correspondence exists between a ui-client-interface object representing the connection and a ui-client-session object representing the user logged into G2 across that connection.

Three-Tier Connections

In a three-tier connection, a single ui-client-interface object can represent multiple ui-client-session objects, since the middle tier creates a single connection, through which multiple users can log in to G2.

Logging in to G2

A `LoginRequest` consists of three properties, each representing OK file user elements that the client uses during the G2 login request:

- `userMode`
- `userName`
- `userPassword`

A `LoginRequest` is immutable. Typically, a client application creates a `LoginRequest`, uses it as the argument to the `login` method on a `TwGateway`, then constructs a new object for the next login session.

Using Accessor Methods

The `ItemRetriever` and `TwConnector` connectivity components provide the following methods for accessing the properties of a `LoginRequest`:

- `getUserName`
- `setUserName`
- `getUserMode`
- `setUserMode`
- `setUserPassword`

Note For security reasons, no accessor method exists to obtain a user password.

A `LoginRequest` is analogous to a `TwConnectionInfo` in that you typically do not edit the properties of either object after it has been created. However, unlike a connection information object, a login request allows you to edit the user mode of a login session once it has been created.

The `ItemRetriever` and `TwConnector` components provide accessor methods that you can call to set the properties of a `LoginRequest` object. These methods have different effects depending on whether a current connection exists, as follows:

If a connection...	This set method...	Has this effect...
Exists	`setUserMode`	Sets the user mode of the current connection.
	`setUserName` `setUserPassword`	Generates a runtime exception; you cannot set the user name and password when a connection exists.

Does not exist	`setUserMode`
	`setUserName`
	`setUserPassword`
		Sets the user mode, user name, and password that the `LoginRequest` uses the next time the `login` method is called.

Using LoginRequest Constructors

A `LoginRequest` provides three constructors for creating a login session to a non-secure or secure G2, depending on the information that is available at compile time:

- `LoginRequest()`
- `LoginRequest(Symbol userMode)`
- `LoginRequest(Symbol userMode, Symbol userName, Symbol userPassword)`

For most applications, login information will be unavailable until run time, at which point the client will obtain one or more user elements with which to populate the `LoginRequest` object.

In all constructors, you must create an upper-case symbol to represent each property. We recommend that you do this by creating a `private static final` variable that is a `com.gensym.util.Symbol`, then calling the `intern` method on the `Symbol` to convert the property as an upper-case string to a symbol.

For example, here is a static final variable that defines the `userMode` argument:

```
private static final Symbol DEVELOPER_ = Symbol.intern("DEVELOPER");
```

A constructor also exists that takes an OK file as an argument. For details, see the API documentation for `LoginRequest`.

Creating a Login Session to a Non-Secure G2

You can call the `login` method on a `TwGateway` without an argument to establish a login session to a non-secure G2. TW2 Toolkit supplies a `LoginRequest` object with default values, which logs in to G2 in Administrator mode, using the user name of the currently logged in user in G2.

You might also want to login to a non-secure G2 as a particular user in a particular user mode. To do this, you create a `LoginRequest` with user mode and user name elements only, then pass this object as the argument to the `login` method.

To create a login session to a non-secure G2, using a default LoginRequest:

```
private static TwAccess myConnection;

try{
    //Establish a connection to G2
    myConnection = TwGateway.openConnection("myhost","1234");

    //log in to non-secure G2 with default LoginRequest
    myConnection.login();

    System.out.println("Connected to G2!");
}
```

To create a login session to a non-secure G2, specifying a user name and mode:

```
package com.gensym.demos.docs.connectivitydemos;

import com.gensym.ntw.TwGateway;
import com.gensym.ntw.TwAccess;
import com.gensym.ntw.LoginRequest;
import com.gensym.jgi.G2AccessException;
import com.gensym.jgi.ConnectionTimedOutException;
import com.gensym.jgi.G2CommunicationException;
import com.gensym.util.Symbol;

public class MyNonSecureLoginRequest {
  private static TwAccess myConnection;
  private static LoginRequest loginRequest;
  private static final Symbol DEVELOPER_ = Symbol.intern("DEVELOPER");
  private static final Symbol TASHA_ = Symbol.intern("TASHA");

  //Create a connection
  public static void runMyConnection () {
    try{
      //Establish a connection to G2
      myConnection = TwGateway.openConnection("localhost", "1111");
      //Create a LoginRequest with user mode and user name only
      loginRequest = new LoginRequest(DEVELOPER_, TASHA_, null);
      //Log in to non-secure G2 with LoginRequest
      myConnection.login(loginRequest);
      System.out.println("Connected to G2!");
    } catch (G2AccessException e){
      e.printStackTrace();
      System.exit (-1);
    }
  }

  // MAIN
  public static void main(String[] args){
    System.out.println("Got to main!");
    runMyConnection();
    myConnection.closeConnection();
    System.out.println("Connection closed!");
    System.out.println ("End of Main!");
    System.exit (0);
  }
}
```

Running this example produces these messages in the command window, assuming you have a G2 running on your local machine on port 1111 with the `connectivity-demos.kb` loaded:

```
Got to main!
Connected to G2!
Connection closed!
End of Main!
```

Creating a Login Session to a Secure G2

For a secure G2, every user must have a valid user mode, user name, and password in the OK file with which G2 was launched. You request a login session to a secure G2 process by first creating a `LoginRequest` object with valid user elements, then passing it as the argument to the `login` method of a `TwGateway`.

Typically, you set at least the `userPassword` property at runtime, which typically requires launching some kind of dialog.

The following example launches an `InputDialog` to obtain the user name, user mode, and password from the user, and passing those values as the arguments to a `LoginRequest` before logging in. You launch the dialog from a demo frame.

The example also prints the current user name and user mode to the command window. In this example, the connection is declared to be a `TwGateway`, and the result of the call to `openConnection` is cast to a `TwGateway`, because the `retrieveUserMode` method is not defined on `TwAccess`. For a discussion of obtaining the user mode, see "Working with User Modes in a TwGateway Connection" on page 112.

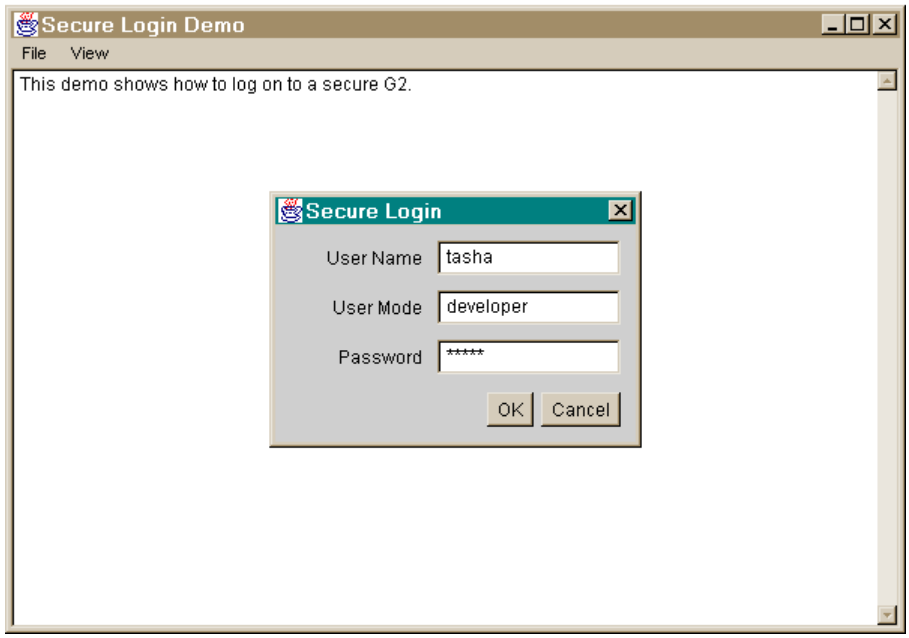
To create a login session for a secure G2:

```
package com.gensym.demos.docs.connectivitydemos;

import com.gensym.ntw.TwGateway;
import com.gensym.demos.Demo;
import com.gensym.demos.DemoShell;
import com.gensym.ntw.TwAccess;
import com.gensym.ntw.LoginRequest;
import com.gensym.util.Symbol;
import com.gensym.jgi.G2AccessException;
import com.gensym.jgi.ConnectionTimedOutException;
import com.gensym.jgi.G2CommunicationException;
import com.gensym.dlg.StandardDialog;
import com.gensym.dlg.StandardDialogClient;
import com.gensym.dlg.InputDialog;
import java.awt.Frame;
```

```
public class MySecureLoginRequest implements Demo, StandardDialogClient
{
  private static TwAccess myConnection;
  private static LoginRequest myLogin;
  private static DemoShell demoShell;

  //Make frame visible and create dialog
  public void runDemo (Frame frame) {
    frame.setVisible(true);
    String[] textFieldLabels = new String[]{"User Name", "User Mode",
                                            "Password"};
    String[] initialValues = new String[]{"", "", ""};
    InputDialog id = new InputDialog(frame, "Secure Login", false,
       textFieldLabels, initialValues, this);
    id.setVisible(true);
    id.setNoEcho(2);
  }

  //Handle dialog events
  public void dialogDismissed (StandardDialog d, int code) {
    InputDialog id = (InputDialog)d;
    if (id.wasCancelled()) return;
    String[] results = id.getResults();
    Symbol userName = Symbol.intern(results[0]);
    Symbol userMode = Symbol.intern(results[1]);
    Symbol userPassword = Symbol.intern(results[2]);
    try{
      //Establish a connection to G2
      myConnection = TwGateway.openConnection("localhost", "1111");
      //Create LoginRequest
      myLogin = new LoginRequest(userName, userMode, userPassword);
      myConnection.login(myLogin);
      System.out.println("Connected to G2!");
      myConnection.returnMessage("Connected to G2!");
      //Get user name and user mode
      Symbol g2UserName = myLogin.getUserName();
      Symbol g2UserMode = myConnection.getUserMode();
      System.out.println(g2UserName);
      System.out.println(g2UserMode);
    } catch (G2AccessException e){
      e.printStackTrace();
      System.exit (-1);
    }
  }
```

```
// MAIN
  public static void main(String[] args){
    System.out.println("Got to main!");
    MySecureLoginRequest demo = new MySecureLoginRequest();
    String msg = new String("This demo shows how to log in to a secure
G2.");
    demo.demoShell = new DemoShell ("Secure Login Demo", msg,
       (Demo)demo, "MySecureLoginRequest.java");
    demo.demoShell.setSize(600,400);
    demo.demoShell.setVisible(true);
  }
}
```

When you run this demo and choose Run Demo, you see a login dialog for entering the user information for the login request, as follows:



Running this example produces these messages in the command window, assuming you have a G2 running on your local machine on port 1111 with the `connectivity-demos.kb` loaded, and you have entered the user information specified above:

```
Got to main!
Connected to G2!
developer
tasha
```

Working with User Modes in a TwGateway Connection

When a client is connected to a secure G2, the application might wish to get the user mode of the current user, and the current user might wish to change his or her user mode. A TW2 Toolkit client can call these methods on a `com.gensym.ntw.TwGateway` to accomplish these tasks:

To...	Use this method...
Get the current user mode from G2	`retrieveUserMode`
Set the current user mode in G2	`changeUserMode`

Note In `TwGateway`, the `setUserMode` and `getUserMode` methods have been deprecated and are replaced by the `changeUserMode` and `retrieveUserMode` methods.

Handling Login Exceptions

These are the exceptions that can occur while creating a login session to either a secure or non-secure G2:

Exception	Description
`InvalidUserModeException`	G2 rejects an attempt to log in, in a particular user mode.
`LoginFailedException`	No more floating licenses are available or an attempt is made to log in to a secure G2 with invalid login information. Each logged in user consumes a floating Telewindows2 Toolkit license.
`NotLoggedInException`	An attempt is made to call a restricted RPC without first logging in. For example, you cannot display a workspace view without first logging in.
`OkException`	A problem occurred while analyzing an OK file associated with the login attempt or while checking its authorization.

Exception	Description
`TooManyLoginAttemptsException`	Too many login attempts have been made consecutively.
`AlreadyLoggedInException`	G2 already includes a ui-client-session object representing the user name, user mode, and password of the user information that the `LoginRequest` object contains.

A client application using the `TwGateway` class can catch all of these login exceptions in the `catch` statement following a connection attempt by catching the `com.gensym.jgi.G2AccessException`.

For example, this code fragment catches connection and login exceptions:

```
catch (G2AccessException e){
   e.printStackTrace();
   System.exit (-1);
}
```

Logging Out From G2

Once you have a connection and a login session, the connection automatically logs you out when you close the connection.

Note Closing the connection does not generate a `loggedOut` event to registered `TwConnectionListeners`. Instead, the `g2ConnectionClosed` event indicates that user has logged out because the connection no longer exists.

Your application might also wish to supply a way for a current user to log out from G2 but leave the connection open. You use these two methods for this purpose:

Class	Method	Description
`G2Gateway`	`closeConnection`	Logs out and closes the connection.
`TwGateway`	`logout`	Logs out and leaves the connection open.

Logging out:

- Logs out the current user from G2.
- Frees the Telewindows2 Toolkit floating license for use by another user.

If you are using the `TwConnector` component, call the `closeConnection` method on a `com.gensym.jgi.G2Gateway` to log out and close the connection.

Logging Out and Closing the Connection

When you close a secure or non-secure connection, you must also log out.

To log out from a secure or non-secure G2 and close the current connection:

```
private static TwAccess myConnection;

try{
   //Establish a connection to G2
   myConnection = TwGateway.openConnection("mynode","1111");

   //Log in to non-secure G2 with default LoginRequest
   myConnection.login();

   //log out and close connection
   myConnection.closeConnection();
}
```

Logging Out and Leaving the Connection Open

You might want to log out and leave the connection open to allow another user to log in to the current connection. When you log out and leave the connection open, you must declare the connection variable to be a `TwGateway` and cast the result of the `openConnection` method call to a `TwGatweway`. This is because the `logout` method is not defined on `TwAccess`.

To log out from a secure or non-secure G2 and leave the connection open:

```
private static TwGateway myConnection;

try{
   //Establish a connection to G2
   myConnection = (TwGateway)TwGateway.openConnection("mynode","1111");

   //Log in to non-secure G2 with default LoginRequest
   myConnection.login();

   //log out and leave connection open
   myConnection.logout();
}
```

The following example creates a connection, logs in as one user, logs out, then logs in as another user:

```
package com.gensym.demos.docs.connectivitydemos;

import com.gensym.ntw.TwGateway;
import com.gensym.ntw.TwAccess;
import com.gensym.ntw.LoginRequest;
import com.gensym.jgi.G2AccessException;
import com.gensym.jgi.ConnectionTimedOutException;
import com.gensym.jgi.G2CommunicationException;
import com.gensym.util.Symbol;

public class MyNonSecureLoginRequest2 {
  private static TwGateway myConnection;
  private static LoginRequest loginRequest;
  private static final Symbol DEVELOPER_ = Symbol.intern("DEVELOPER");
  private static final Symbol KANTI_ = Symbol.intern("KANTI");
  private static final Symbol JOHN_ = Symbol.intern("JOHN");

  //Create a connection
  public static void createConnection () {
    try{
      //Establish a connection to G2
      myConnection = (TwGateway)TwGateway.openConnection("localhost",
                                                         "1111");
      System.out.println ("Connection created!");
    } catch (G2AccessException e){
      e.printStackTrace();
    }
  }

  public static void loginKanti () {
    try {
      //Create a LoginRequest with user mode and user name only
      loginRequest = new LoginRequest(DEVELOPER_, KANTI_, null);
      //Log in to non-secure G2 with LoginRequest
      myConnection.login(loginRequest);
      System.out.println("Logged in as Kanti!");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
```

```
public static void loginJohn () {
    try {
      //Create a LoginRequest with user mode and user name only
      loginRequest = new LoginRequest(DEVELOPER_, JOHN_, null);
      //Log in to non-secure G2 with LoginRequest
      myConnection.login(loginRequest);
      System.out.println("Logged in as John!");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public static void logoutUser () {
    try {
      //Logout currently logged in user
      myConnection.logout();
      System.out.println ("Current user logged off!");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  // MAIN
  public static void main(String[] args){
    System.out.println("Got to main!");
    createConnection ();
    loginKanti();
    logoutUser ();
    loginJohn();
    myConnection.closeConnection();
    System.out.println("Connection closed!");
    System.out.println ("End of Main!");
    System.exit (0);
  }
}
```

Running this example produces these messages in the command window, assuming you have a G2 running on your local machine on port 1111 with the `connectivity-demos.kb` loaded:

```
Got to main!
Connection created!
Logged in as Kanti!
Current user logged off!
Logged in as John!
Connection closed!
End of Main!
```

8

Using a Middle-Tier Server

Provides an overview of the two- and three-tier communication models and describes how to start, configure, and connect to a middle tier.

gensym

Introduction

Telewindows2 (TW2) Toolkit includes two connection configurations:

- Two-tier
- Three-tier

Whenever a TW2 Toolkit client application connects to G2, by default, it connects directly to G2 by using a two-tier configuration. You can optionally direct your client to connect to G2 through a middle-tier server. The available middle tier that TW2 Toolkit supports is currently an RMI server.

This chapter describes the purpose and use of a middle-tier server, and describes how to create such a connection that your client application can use.

Telewindows2 Toolkit Communication Support

All communication between a TW2 Toolkit client and a G2 server occurs through G2 JavaLink, as described in Chapter 2, "Overview of Connectivity" on page 25. The two configurations that TW2 Toolkit provides are made possible by different implementations of one of the fundamental G2 JavaLink connectivity classes.

G2 JavaLink supplies the `com.gensym.jgi.G2Access` interface to provide basic connectivity behavior. By making this important class an interface, other classes can and do implement it's methods for different purposes. In this case, `com.gensym.jgi.G2Gateway` implements `G2Access` for two-tier communication, while a private G2 JavaLink class implements the interface for three-tier communication.

Prerequisites

This chapter requires an understanding of the information presented in the previous chapters of this part:

- Chapter 2, "Overview of Connectivity" on page 25.
- Chapter 5, "Using Connection Information Objects" on page 61.

Packages Covered

com.gensym.jgi.rmi

Interfaces

`G2RMIAccessBroker`

Classes

`G2RMIAccessBrokerImpl`

Relevant Demos

All of the TW2 Toolkit demos that extend `com.gensym.core.GensymApplication` support three-tier communication. To run these demos in three-tier, run them from the command line, using the `-brokerURL` command-line argument, providing the name of the RMI server as the argument.

The demos that support three-tier communication are:

- `wksppanel`
- `singlecxnsdiapp`
- `singlecxnmdiapp`
- `multiplecxnsdiapp`
- `multiplecxnmdiapp`

The demos are located in this directory, depending on your platform:

NT:	`%SEQUOIA_HOME%\classes\com\gensym\demos\`
UNIX:	`$SEQUOIA_HOME/classes/com/gensym/demos/`

Using a Two-Tier Configuration

In a two-tier configuration, the TW2 Toolkit client loads G2 Gateway into the client's Java virtual machine (VM) to handle communication to the server, using Gensym's proprietary protocol.

Each client requires the G2 JavaLink `JgiInterface` shared library file. Thus, the client process is not pure Java.

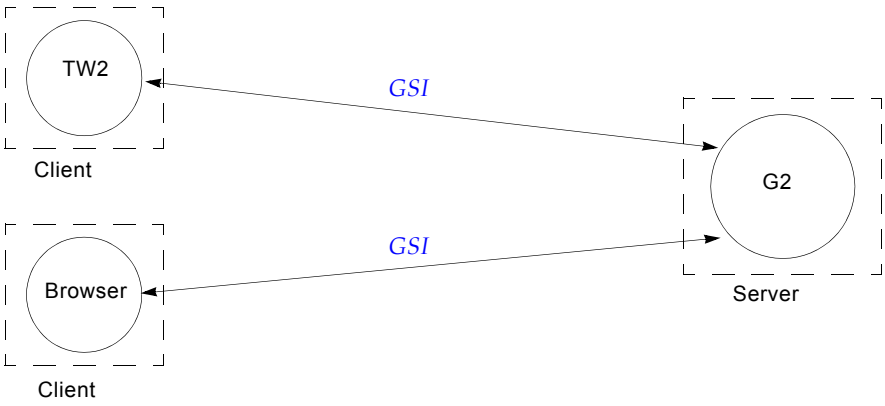
Establishing a Two-Tier Connection

In a two-tier configuration, each TW2 Toolkit client attempts to establish a direct connection with a G2 server. Upon a successful connection, G2 creates two items that each exist in the KB for each TW2 Toolkit client:

- An instance of a ui-client-interface object to represent each client.
- An instance of a ui-client-session item to represent the user login when a successful login is made through the client.

The G2 server has the additional processing load of handling each request from every TW2 Toolkit client connection.

The following figure illustrates two clients communicating with G2 through a two-tier connection:



When to Use Two-Tier Connections

Two-tier connections to G2 are useful when:

- A small number of client connections exist.
- The G2 server has a limited amount of processing activity.
- Network limitations exist such that not having a direct connection to the G2 server could severely impact client performance.

Using a Three-Tier Configuration

In a three-tier communication configuration, each TW2 Toolkit client communicates with G2 through a middle tier, using the Java Remote Method Invocation (RMI) API. A **middle tier** consists of the two processes that use the RMI API, an RMI registry and an RMI server. The system on which the RMI server is running requires the G2 JavaLink `JgiInterface` shared library.

For information on the Java Remote Method Invocation system, see this Web site:

`http://java.sun.com/products/jdk/1.2/docs/guide/rmi/index.html`

Establishing a Three-Tier Connection

When a TW2 Toolkit client attempts to connect to G2 through a middle tier, its connectivity properties must include a Uniform Resource Locator (URL) for the RMI server, in addition to the G2 host and port designators.

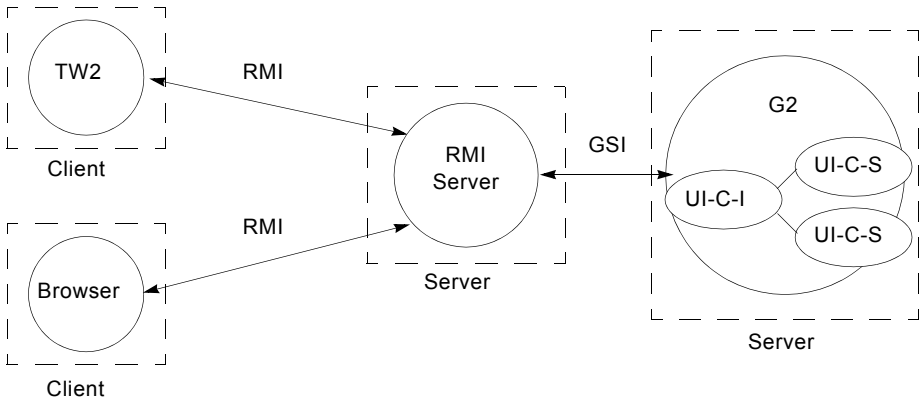
Upon receiving a client connection request, the middle tier uses G2 Gateway and Gensym's proprietary communications protocol to establish a connection with the

G2 server. For more information about connectivity properties, see Chapter 5, "Using Connection Information Objects" on page 61.

For a three-tier configuration, G2 maintains one item to represent the middle tier and one item for each user login, as follows:

- When a TW2 Toolkit client connection is made, the G2 server creates an instance of a ui-client-interface object to represent the middle tier, which G2 views as a single GSI connection.
- Each time a TW2 Toolkit client establishes a login session, G2 creates an instance of a ui-client-session item to represent each user login made through the middle tier.

The following figure illustrates two clients communicating with G2, through an RMI server using a single, shared connection to G2:



As other TW2 Toolkit clients attempt to connect to the current G2 server, the middle tier registers each client as being actively connected and uses the existing connection to communicate client requests to and from the G2 server.

The middle tier funnels all client requests to and from one or more clients and the G2 server process. The middle tier is capable of distributing a single G2 update, such as a show workspace command, to multiple clients. G2 handles calls from multiple clients through one or more middle-tier processes. Off-loading client processing from the G2 server can significantly reduce the G2 process load in a three-tier configuration.

Development Considerations

For developers, connecting a TW2 Toolkit client through a middle tier is entirely transparent and requires no additional coding or development. You can develop a client application without any consideration for the connection configuration it will use.

For your application to run in either two- or three-tier configurations, the client must be capable of specifying a URL when making a connection.

Note The method `registerJavaMethod` on `G2Connection` is not currently supported when called from a RMI Java client.

For information on G2 JavaLink, see the *G2 JavaLink User's Guide*.

When to Use Three-Tier Connections

Three-tier connections to G2 are preferred when:

- A large number of client connections to G2 exist.
- Heavy processing activity exists in G2.
- Multi-platform access to G2 is required on non-supported platforms.
- Sufficient network resources exist for clients to communicate with G2 indirectly, without a performance loss.

Setting Up a Three-Tier Configuration

The three steps for creating a connection to G2 through a three-tier server are:

- Starting an RMI registry.
- Starting an RMI server.
- Specifying the URL for establishing a connection.

You can start the middle-tier server on the same machine as the one on which G2 is running or on an entirely separate system on the network, but it must exist before a TW2 Toolkit client attempts a G2 connection.

Starting an RMI Registry

The **RMI registry** is a naming service tool of the Java Remote Method Invocation system. RMI servers use the registry to bind remote objects to names. A TW2 Toolkit client can look up remote objects in the registry and, after locating the object, invoke its methods.

The `rmiregistry` command creates and starts a remote object registry. The RMI registry should run on the same machine as the RMI server and must exist before starting the server.

To start an RMI registry:

➔ Execute this command in a command window on the machine on which the registry and the server will reside:

```
rmiregistry
```

Starting an RMI Server

An **RMI server** is a process for handling requests from clients to the server, and from the server to its clients.

TW2 Toolkit supplies its own RMI server for your use, which is an instance of the G2 JavaLink `com.gensym.jgi.rmi.G2RMIAccessBrokerImpl` class. This class implements the `G2RMIAccessBroker` interface and supplies the functionality required for connecting to G2 through RMI.

After starting a TW2 Toolkit RMI server, the server exists as a separate process and waits for TW2 Toolkit client connection requests.

To start a G2 RMI server:

➔ Execute this Java command in a command window:

```
java -Djava.rmi.server.hostname=machine
 com.gensym.jgi.rmi.G2RMIAccessBrokerImpl -tsName rmi-server
```

Argument	Description
machine	The fully qualified name of the machine on which you locate the RMI server.
rmi-server	The name of this particular RMI server.

For example:

```
java -Djava.rmi.server.hostname= mynode.gensym.com
com com.gensym.jgi.rmi.G2RMIAccessBrokerImpl -tsName demoserver
```

Together, the *machine* and *rmi-server* arguments specify the URL of the RMI server.

Executing this command registers the RMI server with the RMI registry. Upon registration, the RMI server displays a message similar to the following:

```
rmi://mynode/demoserver bound in registry
```

Tip The designator `rmi://mynode/demoserver` is the format of the URL for this RMI server. Users use this URL when connecting to G2.

The middle tier needs to be able to locate the class definition for the classes that it will create. One way of doing this is to use the `-classpath` command-line option that starts the RMI server.

Connecting to G2 Through a Middle Tier

Once the RMI registry and server have been started, a TW2 Toolkit client need only specify the correct URL, host, and port to connect to G2 through a middle tier

In many cases, a client application will gather connection information from a user and pass it to the appropriate connectivity class to create a G2 connection through a middle tier. For example, if your client application displays a dialog requesting URL, host, and port information, or if it accepts those values as arguments on the command line, the user would enter the *machine* and *rmi-server* arguments that were used to start the RMI server, as described in "Starting an RMI Server" on page 123.

For example, if `mynode` is the machine on which an RMI server named `demoserver` is running, the user would enter this value as the URL:

```
rmi://mynode/demoserver
```

The following table summarizes the minimum connectivity properties required for both two-tier and three-tier connections:

This connection configuration...	Requires these connectivity properties...
Two-tier	Host Port
Three-tier	URL Host Port

For more information about connectivity properties, see Chapter 5, "Using Connection Information Objects" on page 61.

After creating a successful connection to G2, a TW2 Toolkit client communicates with the middle-tier server, using RMI.

A single middle-tier server can manage connections to multiple G2 processes. Multiple TW2 Toolkit clients can use a single middle-tier server to communicate with one or more G2 processes.

Part III

Viewing Workspaces

9

Workspace Views Terms and Concepts

Introduces the terms and concepts that you must understand to work with Telewindows2 Toolkit workspace view components for viewing KB workspaces in Java applications, Web browsers, and other contexts.

Introduction

This chapter introduces the essential workspace view terms and concepts. The definitions in this chapter are used in the subsequent chapters on workspace views.

This and the following chapters show partial examples of workspace views. For complete examples of workspace views in Java applications and applets, see the *Telewindows2 Toolkit Java Demos Guide*.

Workspace View Terminology

Telewindows2 (TW2) Toolkit provides three closely related components that can display G2 KB workspaces in a Java container, such as a TW2 Toolkit application shell or an applet running in a Web browser. Such components are called **workspace view components**. All workspace view components display views of

instances of the G2 kb-workspace class, referred to as a **KB workspace** or simply a workspace when the meaning is clear in the context.

The three types of workspace view components are:

- **Single workspace view —** Displays a view of a single KB workspace that exists in a G2 application. A single workspace view supports scaling of the view along the x and y axes.
- **Multiple workspace view —** Displays any of several KB workspaces. The workspaces can exist in one or more G2 applications. The workspace that is visible in the view is called the **current workspace**.
- **Multiple workspace panel —** A multiple workspace view that has scrollbars and that can listen for programmatic show and hide workspace events in G2.

TW2 Toolkit uses the term **workspace view** in various ways. Depending on the context, it can refer to:

- Any display of a G2 KB workspace in a Java container.
- Any of the three types of workspace view components.

Multiple workspace views and panels have much in common. The term **multiple workspace display** refers generically to a multiple workspace view or a multiple workspace panel.

Programming Workspace Views

From the TW2 Toolkit programmer's perspective, a workspace view is a Java object that is an instance of one of these classes in the `com.gensym.wksp` package:

- `ScalableWorkspaceView`
- `MultipleWorkspaceView`
- `MultipleWorkspacePanel`

Note While a `WorkspaceView` also exists as a separate type of workspace view component, typically you use one of the subclasses of `WorkspaceView`.

A `ScalableWorkspaceView` is associated with at most one KB workspace at a time. A `MultipleWorkspaceView` or `MultipleWorkspacePanel` can be associated with any number of KB workspaces, but it can display only one of them at a time. Any others are hidden and can be displayed in place of the currently visible KB workspace. Displaying more than one KB workspace simultaneously requires using more than one workspace view component.

A `MultipleWorkspaceView` or `MultipleWorkspacePanel` that is associated with only one KB workspace is effectively a `ScalableWorkspaceView`. The ability to display multiple KB workspaces adds overhead, so a single

`ScalableWorksapceView` is more efficient when you only need to display one KB workspace.

You can use the three types of workspace views together, as needed, in an application. Thus, an application that needs to display several KB workspaces could display each in a separate `ScalableWorkspaceView`, or all of them in one `MultipleWorkspaceView` or `MultipleWorkspacePanel`, or it could distribute them into a combination of single and multiple workspace views.

To use a workspace view, you must place it in a visible container, such as:

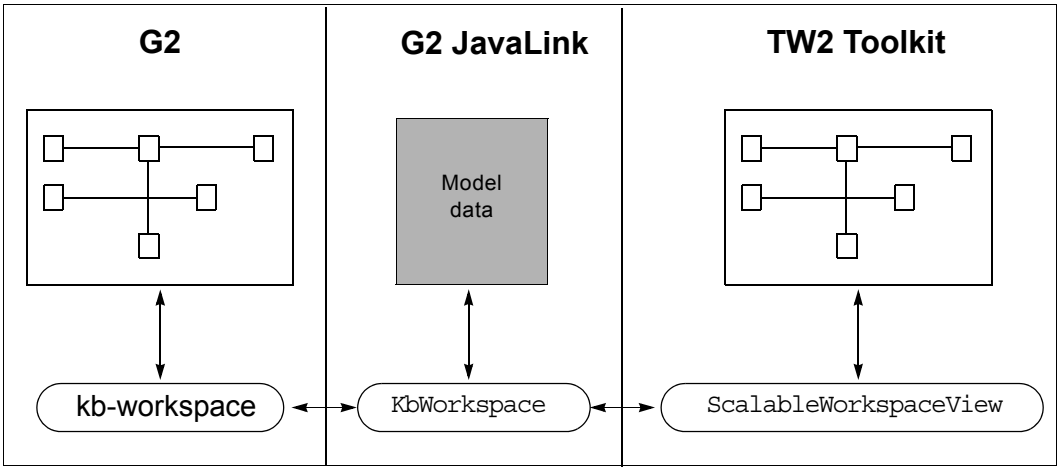
- A Telewindows2 Toolkit application, as described in Chapter 9 "Creating Telewindows2 Toolkit Applications" in the *Telewindows2 Toolkit Java Developer's Guide: Application Classes*.
- Any type of Java container, such as the example shown in "Workspace View Example" on page 177.

You can then invoke methods on the view to elicit the desired behavior.

Workspace views communicate with G2 via G2 JavaLink, as described in Part II, "Connecting to G2," and the *G2 JavaLink User's Guide*.

KB Workspaces vs. Workspace Views

To use workspace views effectively, you must clearly understand the relationship between the kb-workspace item in the G2 server, the G2 JavaLink representation of the KB workspace item as data, and the view of that model data in the TW2 Toolkit client. This figure shows that relationship:



A kb-workspace item in G2 is analogous to a `KbWorkspace` in G2 JavaLink in that both are representations of the same model data. The difference between a kb-workspace in G2 and a `KbWorkspace` in G2 JavaLink is merely how you access

the data — using the G2 language or using Java methods. However, fundamentally, these two data representations are the same.

The more significant difference between a kb-workspace in G2 and a `KbWorkspace` in G2 JavaLink arises when representing KB workspace model data in a client by rendering its data to the screen, as follows:

- In classic Telewindows, the rendering mechanism is private and is not abstracted.
- In a Telewindows2 Toolkit client, the rendering mechanism is completely open and standards-based, and is encapsulated into a single `ScalableWorkspaceView` component, which is capable of painting directly to a `java.awt.Graphics` object through its `paint` method.

Thus, in G2, the view of a kb-workspace is tightly bound to its data representation. For this reason, the term "workspace" conventionally describes both the G2 item and the screen display, and the term kb-workspace is used when necessary to distinguish the item from its display.

By contrast, in a TW2 Toolkit client, a `ScalableWorkspaceView` in a TW2 Toolkit client is separate from its data representation as a `KbWorkspace` in G2 JavaLink. Both the view and its data representation are more accessible in the client. In addition, a `ScalableWorkspaceView` can handle gestures on the client without contacting the server, which means the client uses the network more efficiently.

The difference in how each type of client renders KB workspace data depends on whether you are a user or a developer, as the following sections explain.

User's Perspective

From the user's perspective, a "workspace" is a graphical representation of some information that exists in a G2 knowledge base. This information looks the same whether it appears as a KB workspace in G2 or classic Telewindows, or as a workspace view in a TW2 Toolkit client. The difference exists only in the container in which the workspace is displayed; each of these displays is as much a "view" of the workspace as any other.

KB workspaces and workspace views support virtually all of the same operations. Some differences exist, but these are of interest primarily to users accustomed to the G2 classic interface.

For more information on...	See...
The user interface characteristics of a workspace view and the behavioral differences between workspaces views and KB workspaces.	Chapter 10, "The Workspace View User Interface" on page 133.
How the user interacts with the text editor, which appears whenever the user edits a textual attribute of an item on a workspace from a TW2 Toolkit client	Chapter 11, "Using the Text Editor" on page 147.

Developer's Perspective

As a developer, you need to be aware of the differences between how a classic Telewindows client interacts with KB workspace data and how a TW2 Toolkit client interacts. In particular, you need to understand that when users interact with a "workspace" in a TW2 Toolkit client, they are interacting directly with the `ScalableWorkspaceView`; they are interacting only indirectly with the `KbWorkspace`. For example, it is the `ScalableWorkspaceView`, not the `KbWorkspace`, that determines how the user drags or resizes an item, and even how that item is drawn to the screen.

Because the user is interacting with a view of the KB workspace and not the KB workspace itself, the model data in the server does not get updated until the user has concluded the interaction. For example, when the user moves an object in a workspace view, the object does not move in the G2 server until the user interaction is complete. This means a TW2 Toolkit client uses the network more efficiently.

By contrast, when the user interacts with a KB workspace either in G2 or classic Telewindows, the object it represents gets updated immediately. For example, in classic Telewindows, when the user moves an object on a KB workspace, the object moves simultaneously in the Telewindows client and the G2 server. This means classic Telewindows uses the network heavily each time the user interacts in any way with the KB workspace.

In addition, because you have much more control over both the data representation of a KB workspace and its view in a TW2 Toolkit client, you need to be aware of how you can interact with and manipulate a `ScalableWorkspaceView` in the client.

For more information on...	See...
The most common features that are available to you when programming the three types of workspace views.	Chapter 12, "Using Workspace View Components" on page 159.
The data representation of a workspace view	`com.gensym.classes.KbWorkspace`
Displaying workspace views in a document window when building multiple document interface (MDI) applications	Chapter 8, "Using Telewindows2 Toolkit MDI Documents" in the *Telewindows2 Toolkit Java Developer's Guide: Application Classes*.

10

The Workspace View User Interface

Describes the user interface for Telewindows2 Toolkit workspace view components.

gensym

Introduction

This chapter describes the default behavior of a Telewindows2 (TW2) Toolkit workspace view and the G2 objects that appear in it. This behavior is the same for all three classes of workspace views.

This chapter contains various examples that show workspace view capabilities. For a guided tour of using a workspace view in a Java application, see Chapter 2

"Guided Tour of the Telewindows2 Toolkit Shell" in the *Telewindows2 Toolkit Java Developer's Guide: Application Classes*.

This chapter assumes that you have read Chapter 9, "Workspace Views Terms and Concepts" on page 127, which defines the essential workspace view terms and concepts.

For information on the programmatic features of workspace views, see Chapter 12, "Using Workspace View Components" on page 159.

Relevant Demos

The following demos use workspace views:

- `wkspdemo`
- `wkspapplet`
- `wksppanel`

The demos are located in this directory, depending on your platform:

NT:	`%SEQUOIA_HOME%\classes\com\gensym\demos\`
UNIX:	`$SEQUOIA_HOME/classes/com/gensym/demos/`

Workspace View Appearance

A KB workspace is a locally managed rectangle within the G2 window. It has no title bar, scrollbars, or other window accoutrements.

A workspace view is a Java component that provides a view of a KB workspace in a TW2 Toolkit client. Items in a workspace view appear exactly the same as items on a KB workspace in G2. This identity extends even to very complex items, such as trend charts.

Some aspects of workspace view appearance, however, such as scroll position and item selection, are independent of G2 and have no meaning to it; they exist only in the workspace view and can differ in different views of the same KB workspace.

Whereas in G2, a KB workspace can exist as a separate entity within the G2 window, in a TW2 Toolkit client, a workspace view must appear within a container of some kind, such as a `java.awt.Frame`.

If you are creating a Java application that supports multiple document windows within a single application frame, you can use `com.gensym.shell.util.WorkspaceDocument` to display a workspace view within a child document of the parent frame. When embedded within such a workspace document, the

workspace view behaves like a window, with a title bar, minimize, resize, and close buttons, and scroll bars. An example of an application that uses workspace documents to display workspace views is the TW2 Toolkit default application shell.

For information on using workspace documents, see Chapter 8 "Using Telewindows2 Toolkit MDI Documents" in the *Telewindows2 Toolkit Java Developer's Guide: Application Classes*.

Workspace View Behavior

All user interface items that are intrinsic to G2 have the same effect in a workspace view that they have in a KB workspace. For example, clicking an action button in either context when G2 is running executes the action of the button. TW2 Toolkit renders GUIDE/UIL dialogs as workspace views so they have the same behavior in the client that they have in G2. You can also create native palettes directly from GFR palettes.

TW2 Toolkit does not support a migration path for all G2 utilities. In such cases, you can substitute equivalent TW2 Toolkit techniques to produce the desired effects. For information on which KB utilities TW2 Toolkit supports, see Chapter 3 "Compatibility with Existing KBs and G2 Utilities" in the *Telewindows2 Toolkit Release Notes*.

Workspace views honor most item configurations that affect KB workspaces.

Note If you edit the item configuration of an item in the G2 server, you must redownload the workspace view to see the changes.

For details on features and configurations that workspace views do not support, see "Unsupported Features of Workspace Views" on page 145.

Synchronizing KB Workspaces and Workspace Views

When something changes a KB workspace in G2, the change propagates to all associated workspace views. When a TW2 Toolkit client requests a change to a workspace view, the request is first transmitted to G2, which changes the underlying KB workspace item. The change then propagates to all associated workspace views, including the view in which the change was requested.

This two-way communication occurs automatically via the capabilities described in Part II, "Connecting to G2." The communication is typically very fast, giving the impression that the change occurs directly in the client, rather than being routed through the G2 server.

To reduce the amount of network traffic between KB workspaces and workspace views, the two communicate only when necessary; thus, much of the computation is handled locally. For example, once a workspace view has communicated with G2 to determine the contents of a popup menu for an item on a workspace view, that information is cached on the client and is used the next time the user displays the popup.

For an example of the difference between the local and remote aspects of changing a workspace view, see "Moving and Reshaping Objects" on page 138.

Differences between KB Workspaces and Workspace Views

The independence of workspace views from G2 KB workspaces allows a workspace view to follow modern GUI conventions, which the classic G2 interface generally does not do. Consequently, the workspace view user interface differs in various ways from the classic G2 interface.

Few of the differences between KB workspaces and workspace views have functional significance; they are just differences of convention and appearance. However, some functional differences do exist because:

- Workspace views run in clients that are independent of G2 and communicate with it only when necessary.
- Not all G2 item configurations are applicable to workspace views.

The following sections describe workspace view behavior and note differences between it and the corresponding behavior in G2.

Changing Workspace View Appearance

Many changes that you make to the appearance of a workspace view within its container do not affect the underlying KB workspace item but only the overall appearance of the view. For example, when you reshape the container in which a workspace view appears, such as a workspace document, such changes are local to the particular view and have no effect on any other view or on the underlying KB workspace item in G2.

Similarly, scrolling either the container in which a workspace view appears or the workspace view itself, which you can also do, does not effect any other view or the underlying KB workspace in G2.

This independence is similar to that provided by classic Telewindows, which allows the same workspace to appear at different positions and scales in different G2 windows.

Changing Objects in a Workspace View

This section describes the gestures that operate on the objects in a workspace view. All modern GUIs provide essentially the same set of gestures for manipulating graphical objects, and workspace views use these exclusively. Your platform might differ slightly or use different mouse-button mappings, but an obvious equivalent gesture should exist.

Selecting and Deselecting Objects

In a workspace view, you can select and deselect individual objects or groups of objects. The selected status persists until a subsequent gesture explicitly changes it.

The selection status of an object in a workspace view is local to the view. It has no effect on that object in any other view or on the KB workspace item in G2.

To select an object:

➔ Click the object.

Selection occurs as soon as you press the mouse button. The selected object becomes outlined, and handles appear on it. Any previously selected objects become deselected.

To select several objects at once:

➔ Click and hold on the workspace view background, drag to create a selection rectangle that completely includes the objects, then release the mouse button.

Each of the selected objects becomes outlined and displays handles. Any previously selected objects become deselected.

To add to an existing selection:

➔ Hold down the Shift key, then click each object to be added.

or:

➔ Click and drag to draw a selection rectangle that includes all item in the new selection.

Any previously selected objects remain selected, and the newly specified object(s) become selected also.

To deselect a selected object:

➔ Hold down the Shift key, then click the selected object.

The object is deselected. Any other selected objects remain selected.

To deselect all selected objects:

➔ Click somewhere on the workspace view background away from any object.

Moving and Reshaping Objects

To minimize processing and network overhead, moving and reshaping objects initially changes only an empty outline that appears around the object(s) when the operation begins. Such an outline is called a **ghost**.

Only the local workspace view is aware of any moving or reshaping in progress. After the change is complete, the client notifies G2, which then changes the underlying KB workspace item and all views of it, including the one on which the change was requested.

To move an object:

➔ Click an object and drag it to a new location.

If the object was not previously selected, it becomes so when you press the mouse. A ghost of the object moves with the mouse. When you release the mouse button, the client notifies G2 of the change, and the object moves to its new location.

To move a group of objects:

1 Select the objects to be moved.

2 Click and drag any of the objects to a new location.

Ghosts of all selected objects move with the mouse. When you release the mouse, G2 is notified of the change, and all objects move to the new location. The objects retain their relative positions.

Note If moving an object is limited by a move-object or move-object-beyond-workspace-margin item configuration in a G2 configuration statement, moving any group that contains the object is limited by that configuration.

To reshape an object:

1 Select the object to be reshaped.

2 Drag one of the handles of the selected object until the ghost is the desired size and shape.

Note This feature of workspace views replaces the change size menu choice for items on a KB workspace.

When you release the mouse button, the object resizes to fill the outline indicated by the ghost.

To reshape a group of objects:

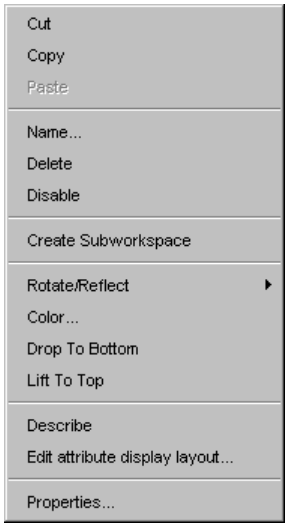
1. Select the objects to be reshaped.
2. Reshape any one of the objects as previously described.

All of the selected objects reshape in the same proportions as the one you reshaped. For example, if you reshape one selected object to be half as tall and twice as wide as it was, every other selected object also reshapes to be half as tall and twice as wide as it was.

The relative positions of the reshaped objects do not change. Thus, reshaping a group of objects can cause objects to overlap or cease to overlap.

Using Workspace View Item Popup Menus

Each item in a workspace view has a popup menu. A standard item popup menu in a workspace view looks like this:



Workspace views handle the invocation and display of popup menu choices locally, using data obtained from G2 as needed to populate the menu.

Comparison with Item Popup Menus in KB Workspaces

The system menu choices in a item popup menu in a workspace view are equivalent to the choices of the same name in the item popup in G2, except for the following menu choices:

This popup menu choice in a workspace view...	Is equivalent to this popup menu choice in a KB workspace...
Cut/Copy/Paste	clone/transfer
Name	names
Properties	table

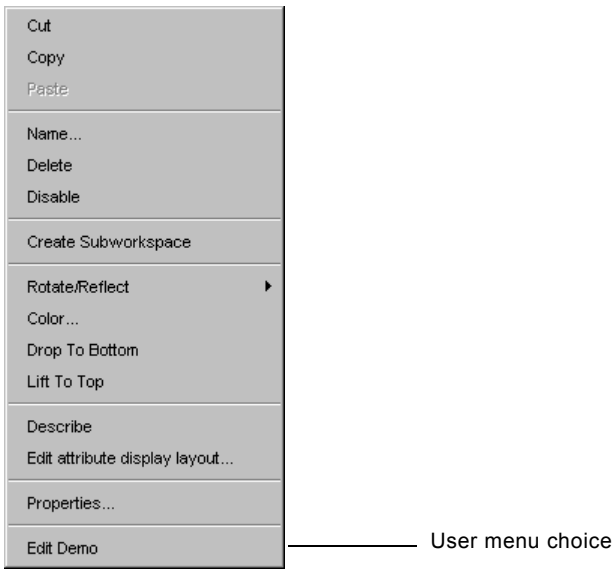
The G2 change size menu choice is implemented by reshaping the object as described in "Moving and Reshaping Objects" on page 138.

For information on using the Properties menu choice, see "Using Workspace View Item Properties Dialogs" on page 141.

User Menu Choices in Item Popup Menus

If an object has any G2 user menu choices, they appear at the bottom of the item popup menu below Properties. User menu choices appear in an item popup menu in a workspace view only when G2 is running.

Here is a popup menu for an item that defines a G2 user menu choice named edit-demo:



Choosing a user menu choice executes the choice in the G2 server. If the menu choice changes the underlying KB workspace item, the change propagates to all workspace views, including the one in which the choice was made.

Interacting with Item Popup Menus

To display an item's popup menu:

➔ Right-click the item.

To execute a menu choice from a popup menu:

➔ Click the menu choice.

To dismiss a popup menu without choosing:

➔ Click somewhere on the workspace view away from the popup.

The popup disappears, and the environment remains unchanged.

Using Workspace View Item Properties Dialogs

The Properties menu choice on an item popup menu in a workspace view is equivalent to the table menu choice in an item popup on a KB workspace. However, choosing Properties in a workspace view displays a free-floating tabbed dialog rather than a table.

By default, the workspace view automatically generates Properties dialogs for items, as described in "Item Properties Dialogs" on page 205. You can also create and register custom item properties dialogs for individual items or entire classes of items, as described in Chapter 16, "Launching Custom Item Properties Dialogs" on page 305.

To display or edit an item's properties:

➔ Choose Properties from the item's popup menu.

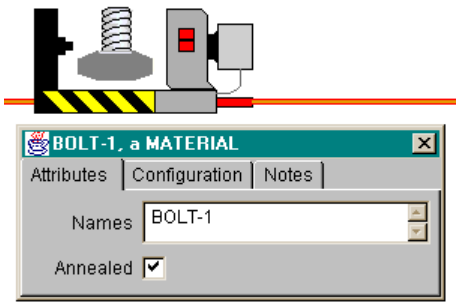
or

➔ Double-click the item.

A dialog appears that has one or more tabs. The possible tabs are:

- Attributes
- Configuration
- Notes

For example, the figure below shows the Properties dialog for a material named bolt-1, with the Attributes tab selected. In this example, the Names attribute is textual and the Annealed attribute is a truth-value that currently is true:



A workspace view obtains from G2 the values shown in a Properties dialog just before it launches the dialog. If G2 changes an item attribute while a Properties dialog displays it, the dialog changes to display the new value.

Attributes Tab

The Attribute tab always appears and is selected when the dialog opens. This tab page lists the names and values of all attributes other than notes and item-configurations that appear in the item's G2 table. Attributes that are invisible in G2 due to item configuration statements do not appear on the Attributes page.

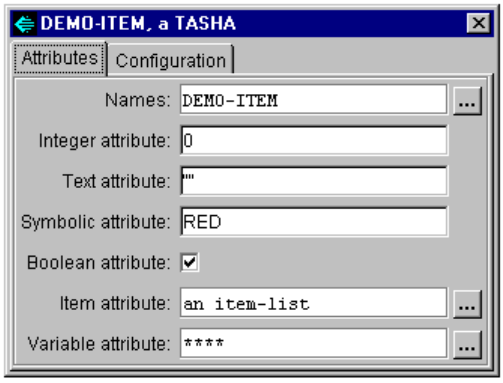
The values on the Attributes page appear using a variety of dialog controls, depending on the data type of the attribute. You can edit the value of any attribute that is not read-only.

To edit an attribute value:

➔ Proceed as shown in the following table:

This G2 data type...	Uses this dialog control...	Which you edit by doing this...
Numeric	Text box	Click the value and edit it.
Text	Text box	Click the value and edit it.
Symbolic	Text box	Click the value and edit it.
Truth-value	Check box	Click to toggle the check.
Subobject	Text area	Click the button next to the text area to launch a subdialog.

This figure shows an item Properties dialog for a class that defines the following class-specific attributes:



TASHA

integer-attribute is an integer, initially is 0;
text-attribute is a text, initially is "";
symbolic-attribute is a symbol, has values
red, blue, or green, initially is red;
boolean-attribute is a truth-value, initially is
true;
item-attribute is an instance of an item-list,
initially is an instance of an item-list;
variable-attribute is given by a float-variable,
initially is given by a float-variable

Note Because G2 does not enforce type checking for user-defined enumerated attributes, such attributes cannot appear as dropdown choices in the client. However, system-defined classes that use enumerated attributes, such as the type-of-relation of a relation item, do represent these attributes as dropdown choices.

Editing System-Defined Attributes with a Grammar

Some system-defined attributes have an associated grammar that controls the values for the attribute. Such attributes appear with a button to the right of the attribute field. Clicking the button displays a native, syntax-guided text editor with language prompts, described in Chapter 11, “Using the Text Editor” on page 147.

Editing Typed Attributes

Some system-defined attributes and all user-defined attributes have only a type; they have no associated grammar. The following principles govern changes to such attributes:

- When you edit a typed attribute, G2 does not track the intermediate steps of the change.
- When you complete an edit, G2 receives notification of the requested change, as follows:
  - If the new value is valid, G2 makes the change. Any resulting changes to the KB workspace propagate immediately to all managed dialogs.
  - If the value is invalid, a dialog describing the error appears and the value in G2 remains unchanged.
- The last edit that completes overrides all other edits.

Configuration Tab

The Configuration tab appears unless a configuration is in effect that hides the item-configurations attribute for the item. The corresponding page describes any configurations that are in effect for the item.

To change an item's configuration:

1. Click the Configuration tab, then click the current value of the configuration.

   The Text Editor appears.

2. Edit the configuration as described in Chapter 11, "Using the Text Editor" on page 147.

Notes Tab

The Notes tab appears only if the item has G2 notes that describe some problem or special condition. The corresponding page contains the notes and is read-only.

Scaling Workspace Views

You can scale a workspace view by a specified percentage, to fit the workspace document, or in and out by a standard percentage.

To make these commands available, use `com.gensym.shell.commands.ZoomCommands`, described in Chapter 11, "Using Shell Commands" in the *Telewindows2 Toolkit Java Developer's Guide: Application Classes*.

Unsupported Features of Workspace Views

Most G2 item configurations on KB workspaces have the same effect in workspace views that they have on KB workspaces. However, some item configurations that affect KB workspaces, as well as certain other features, are either obsolete or have not been implemented in TW2 Toolkit.

For information on unsupported item configurations, see Appendix B, "Compatibility Issues."

For information on additional compatibility with G2 in general, see Chapter 3 "Compatibility with Existing KBs and G2 Utilities" in the *Telewindows2 Toolkit Release Notes*.

11

Using the Text Editor

Describes the text editor that workspace views use for changing attributes with a grammar. Comparisons and contrasts to the native G2 text editor appear as needed.

Introduction

When you edit a system-defined attribute through an item Properties dialog in a workspace view and that attribute has an associated G2 grammar, the workspace view launches the Telewindows2 (TW2) Toolkit text editor. The text editor provides the same capabilities as the G2 Text Editor, but it packages them in a dialog that provides greater convenience and conforms to the native GUI.

This chapter describes the essential techniques for using the TW2 Toolkit text editor, the text editor shortcuts, and the capabilities of the View menu. It also provides a reference for the text editor menus.

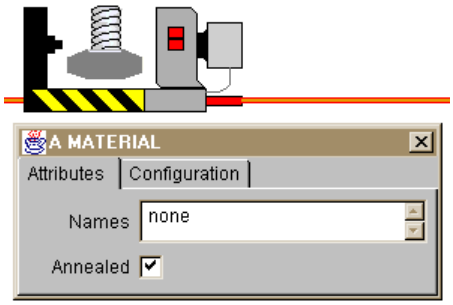
This chapter assumes you are familiar with:

- The G2 Text Editor.
- The standard mouse and keyboard gestures for your platform.

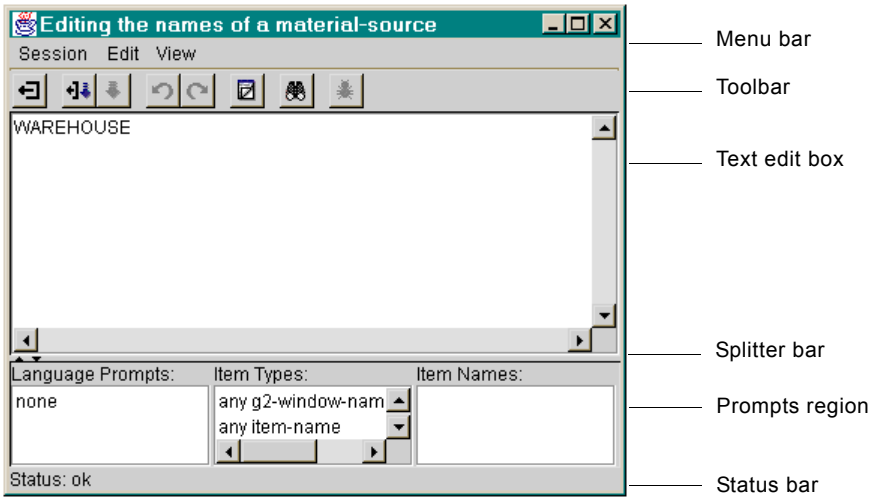
Using the Telewindows2 Toolkit Text Editor

The Telewindows2 Toolkit text editor appears whenever you click a value in an item's Properties table that has an associated G2 grammar, as described in "Using Workspace View Item Properties Dialogs" on page 141.

In the following figure, the bolt in the workspace view has two attributes: Names and Annealed. Choosing Properties from the item's popup menu displays this dialog with the Attributes page selected by default:



No name has been specified for the bolt, so the value of the Names attribute is none. Because Names is a grammatical attribute, clicking the value displays the text editor. This figure shows the text editor and labels its parts:



The text editor is not modal — more than one editor can be active at one time, editing the same or different items in the same or different G2 applications. When concurrent editors are editing the same value, the change that is applied last overrides all others.

Editing Text

The top region of the text editor is a scrollable text edit box. The text in this box is initially the value that you clicked to display the editor. The text appears in a fixed-width font.

To enter text in the edit box:

➔ Use standard mouse and keyboard gestures to enter the desired value.

For details, see "Text Editor Shortcuts" on page 153.

G2 parses the text that you enter to check for grammatical correctness. To avoid excessive network traffic, parsing occurs only when you cease typing for one half second. G2 parses only the text between the beginning of the text to the cursor position; text after that position is ignored. Text already known to be correct is not rechecked.

After G2 parses the text, the editor's prompt region and status bar change to reflect the results, as described under "Using Grammar Prompts" on page 150 and "Detecting Syntax Errors" on page 151.

To undo/redo a change:

➔ Choose Edit > Undo to undo the last change or Edit > Redo to undo the undo.

To delete all text in the edit box:

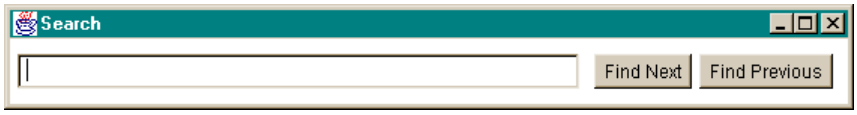
➔ Choose Edit > Clear.

Searching for Text

To search for text in the edit box:

1 Choose Edit > Search.

The search dialog appears:



2 Enter the string to be searched for.

The search is case sensitive. Use standard mouse gestures as needed to edit the search string.

The search proceeds incrementally as you specify the search string. The first instance of the string after the current cursor position becomes selected. If no instance exists, no text is selected and the cursor position does not change.

3 Click Find Next and Find Previous to search forwards or backwards for additional instances of the search string.

   The next/previous instance of the search string becomes selected. When no further instances exist before/after the cursor, the button that searches in that direction is unavailable.

4 Close the dialog when you no longer need it.

The search dialog is not modal, so you can leave it open and use it whenever you need to. When you exit the text editor, any open search dialog disappears.

Using Grammar Prompts

The prompts region displays information that guides you through the grammar of the attribute you are editing. The information consists of prompts that appear in three scrollable read-only list boxes.

To select a prompt from any prompts region box:

1 Scroll to the desired prompt.

2 Click the prompt.

The effect varies with the box, as follows.

Language Prompts

The Language Prompts box displays strings that you can legally enter in the edit box at the cursor position. Clicking any string enters it into the text at that position, followed by a blank. The cursor moves to the position after the blank. If you enter part of a prompt, then click that prompt in the Language Prompts box, the text editor inserts the rest of the prompt into the text box at the cursor position.

Item Types

The Item Types box displays descriptions of the kinds of items (if any) that you may enter at the cursor position, such as any class or any symbol. Clicking any description displays all instances of the item type in the Item Names box.

Item Names

The Item Names box displays the names of all items that match the description currently selected in the Item Types box. Clicking any name enters it into the text at the cursor position, followed by a blank. The cursor moves to the position after the blank. If you enter part of a name, then click that name in the Item names box, the editor inserts the rest of it into the text box at the cursor position.

Detecting Syntax Errors

While you edit text, the status bar displays the grammatical status of the text that G2 has checked. If the text is grammatically correct, the status bar displays OK; if not, the status bar displays Bad, followed by a brief description of the error.

Because parsing occurs only when you have ceased typing for a half second, you might have typed beyond the location of the error by the time the editor reports it.

To determine the location of a syntax error:

➔ Choose Edit > Goto Error.

The cursor moves to point in the text at which the first error occurs. This capability replaces the ellipses (...) that would appear at the beginning of an error in the G2 Text Editor.

Applying Changes

While editing is underway, the value of the attribute being edited remains unchanged in G2. To change it, you must explicitly apply changes. When you apply changes, the editor remains open, allowing you to make further changes.

You can also apply changes and exit in a single gesture, as described in "Toolbar Buttons" on page 155.

To apply changes made in the text editor:

➔ Choose Session > Apply Changes.

G2 parses any unparsed text. If any syntax error exists, no change is applied, and the status bar displays Bad, followed by a brief description of the error.

If no syntax error exists, G2 compiles the text. If the compilation fails, no change is applied, and the status bar displays Compiler, followed by a description of the error.

If the compilation succeeds, the text in the text box becomes the value of the edited attribute in G2. Any resulting changes to KB workspace appearance propagate immediately to all workspace views.

Exiting the Editor

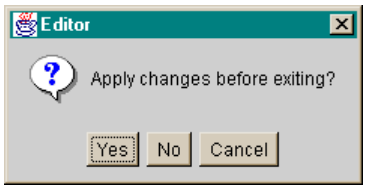
The text editor remains until you explicitly exit from it.

You can apply changes and exit in a single gesture, as described in "Toolbar Buttons" on page 155.

To exit the text editor:

➔ Choose Session > Exit.

If changes exist that have not been applied, the editor closes. If changes exist that have not been applied, the editor displays a confirmation dialog:



Click...	To...
Yes	Apply changes, as described under "Applying Changes" on page 151, then exit. If changes cannot be applied due to an error, the Exit command is cancelled.
No	Discard changes and exit.
Cancel	Cancel the Exit command itself, which allows editing to continue as if the exit had not been attempted.

Entering Native Language Texts

The text editor supports entering texts in languages other than English by using one of two techniques:

- For users running localized versions of NT that provide native input methods for languages other than English, such as Japanese or Hebrew, you can use those input methods to enter text directly in the native language.
- You can enter four-digit Unicodes to represent native characters.

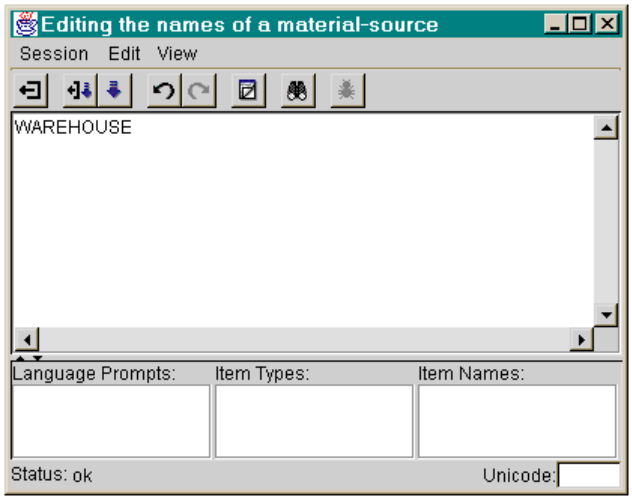
To support the rest of the Unicode character set beyond the ASCII characters, you must:

- Make the required fonts available on your computer.
- Set up the font preferences file as described in Appendix A: "Supporting Languages Other than English" in the *Telewindows2 Toolkit Installation Guide*.

To enter Unicode characters:

1 Choose View > Unicode Insertion to enable the Unicode insertion box.

The text editor displays a Unicode insertion box on the status bar, as follows:



2 Enter a four-digit unicode character to represent the native character.

The character that the Unicode represents appears in the text area in the currently loaded font.

Text Editor Shortcuts

The text editor provides several types of shortcuts:

- Keyboard accelerators
- Text editor popup menu
- Toolbar buttons

Keyboard Accelerators

The text editor supports the following keyboard accelerators, some of which might exist in the native GUI independently of TW2 Toolkit:

To move the cursor...	Use...
Right one character	Right Arrow key
Left one character	Left Arrow key
Up one line	Up Arrow key

To move the cursor...	Use...
Down one line	Down Arrow key
Right one word	Control + Right Arrow key
Left one word	Control + Left Arrow key
To the start of the line	Home key
To the end of the line	End key
To the start of the text	Control + Home key
To the end of the text	Control + End key

To delete...	Use...
Character left	Backspace Key
Character right	Delete Key

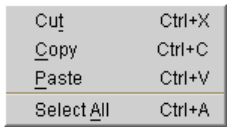
To insert a...	Use...
Tab	Tab key
Line break	Enter key

To...	Use...
Cut text to the clipboard	Control + x
Copy text to the clipboard	Control + c
Paste text from the clipboard	Control + v

To complete...	Use...
The first language prompt that matches the entered text	Ctrl + Space
The last language prompt that matches the entered text	Ctrl + Shift + Space

The Text Editor Popup Menu

Clicking the right mouse button in the text box when text is currently selected displays the following popup menu:



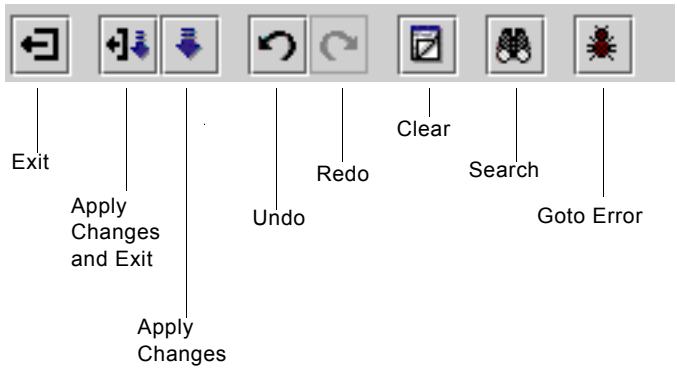
Menu choices that are inapplicable under current conditions are unavailable.

To dismiss the menu without choosing, click anywhere away from it.

For information about undoing an operation, see “Editing Text” on page 149.

Toolbar Buttons

The tool bar contains a set of icons representing frequently used commands. Buttons whose commands are inapplicable under current conditions are unavailable. The next figure shows the command that each button executes:



Note that the Apply Changes and Exit button provides additional functionality over the menu choices by performing two actions in a single button.

For information about...	See...
Exit	“Exiting the Editor” on page 151.
Apply Changes	“Applying Changes” on page 151.
Undo, Redo, Clear, and Search	“Editing Text” on page 149.
Goto Error	“Detecting Syntax Errors” on page 151.

Text Editor Menu Reference

This section provides a quick reference to the commands in the Telewindows2 Toolkit text editor menu bar. The menu bar provides three pulldown choices:

- Session
- Edit
- View

Commands that are inapplicable under current conditions are unavailable. The toolbar provides equivalents for some menu bar choices. See "Toolbar Buttons" on page 155 for details.

Session Menu

The Session menu provides commands that apply changes and/or exit the editor:

Choose...	To...	See...
Apply Changes	Apply the changes you have made and continue the edit session.	"Applying Changes" on page 151.
Exit	Exit the editor. You will be prompted to apply any changes.	"Exiting the Editor" on page 151.

Edit Menu

The Edit menu provides commands that operate on the text in the Text Box:

Choose...	To...	See...
Undo	Reverse the preceding change to the text.	"Editing Text" on page 149.
Redo	Reverse the preceding Undo command. The text then appears as it did before the Undo.	"Editing Text" on page 149.
Clear	Delete all text in the Text Box.	"Editing Text" on page 149.

Choose...	To...	See...
Search	Search for a character or string of characters anywhere in the text.	"Editing Text" on page 149.
Goto Error	Move the cursor to the position in the text at which the first error begins.	"Detecting Syntax Errors" on page 151 and "Applying Changes" on page 151.

View Menu

The View menu provides commands that toggle the display of certain text editor capabilities:

Choose...	To toggle the...	See...
Unicode Insertion	Unicode insertion box	"Entering Native Language Texts" on page 152.
Tool Bar	Toolbar	"Toolbar Buttons" on page 155.
Language Prompts	Language Prompts box	"Using Grammar Prompts" on page 150.
Item Names	Item Names box	"Using Grammar Prompts" on page 150.
Item Types	Item Types box	"Using Grammar Prompts" on page 150.
Status	Status bar	"Detecting Syntax Errors" on page 151 and "Applying Changes" on page 151.

12

Using Workspace View Components

Describes techniques and methods that you typically use when using any type of workspace view in any programming environment.

Introduction

The three types of workspace views provide somewhat different capabilities, and different types of views sometimes implement the same capability in different ways. These variations exist because the three types of views have different purposes:

- `ScalableWorkspaceView` — Provides essential capabilities for representing a KB workspace in the client as a single workspace view, including scaling.
- `MultipleWorkspaceView` — Provides a convenient way to manage a collection of workspace views.
- `MultipleWorkspacePanel` — Adds frequently needed higher-level capabilities to a multiple workspace view, but some lower-level capabilities become inaccessible.

Note `ScalableWorkspaceView` extends `WorkspaceView`; however, typically you use one of the subclasses of `WorkspaceView`.

All workspace views share many features. This chapter distinguishes between the three types of views only where they differ. Wherever two or more types of views are the same, this chapter refers to them generically as a workspace view or multiple workspace display, as described in "Workspace View Terminology" on page 127.

This chapter assumes that you:

- Have read Chapter 9, "Workspace Views Terms and Concepts," which defines the essential workspace view terms and concepts.
- Understand the use of events and methods in defining user interfaces.
- Have some familiarity with Java programming.

This chapter describes common capabilities for workspace views, providing the syntax and a description for each method described. For each workspace view capability, this chapter:

- Tells which type(s) of workspace views offer the capability.
- Shows how to invoke the capability in each type of view that offers it.

Additional methods also exist that are not described in this chapter. For complete information on all workspace view methods, see the Telewindows2 (TW2) Toolkit API documentation.

For a complete example of programming a workspace view, see "Workspace View Example" on page 177.

Packages Covered

com.gensym.wksp

Interfaces

ItemView
WorkspaceElement

Classes

ScalableWorkspaceView
MultipleWorkspaceView
MultipleWorkspacePanel
WorkspaceView
WorkspaceViewScrollbar

com.gensym.ntw.util

Interfaces

CollectionListener
SelectionListener

com.gensym.util

Interfaces

ItemListener

Relevant Demos

The following demos use workspace views:

- wkspdemo
- wkspapplet
- wksppanel

The demos are located in this directory, depending on your platform:

NT:	%SEQUOIA_HOME%\classes\com\gensym\demos\
UNIX:	$SEQUOIA_HOME/classes/com/gensym/demos/

Creating Workspace Views

For each of the following types of workspace view, a constructor with no arguments exists:

- `ScalableWorkspaceView`
- `MultipleWorkspaceView`
- `MultipleWorkspacePanel`

For a `ScalableWorkspaceView`, you can create and populate the workspace view in one call, by providing a `com.gensym.classes.KbWorkspace` as an argument to the constructor.

Populating Workspace Views

You populate workspace views as follows:

- For a single workspace view, you set a KB workspace in the view explicitly or when you create the view.
- For a multiple workspace display, you explicitly add a KB workspace to the view.
- A multiple workspace panel can automatically populate the view by adding the panel as a `com.gensym.ntw.WorkspaceShowingListener`.

Populating a Single Workspace View

You can populate a single workspace view in the same call that creates it or at any time thereafter.

To create and populate a single workspace view:

➔ `ScalableWorkspaceView(KbWorkspace` *`workspace`*`)`

To create and populate a scalable workspace view:

➔ `ScalableWorkspaceView(KbWorkspace` *`workspace`*`)`

To populate an existing single workspace view:

➔ `setWorkspace(KbWorkspace` *`workspace`*`)`

The KB workspace is immediately visible in the view, which loses all knowledge of the workspace previously contained in it.

Populating a Multiple Workspace Display

You can add a KB workspace to a multiple workspace display at any time. The workspace is added to the end of a list of KB workspaces that the display contains. This list is called the **workspace list**. You can add the same KB workspace more than once to a multiple workspace display, which causes it to appear more than once in the workspace list.

A workspace view must generate information from a `KbWorkspace` to display it graphically. For each KB workspace in a multiple workspace display, you can specify whether the display caches this information or re-creates it every time that KB workspace becomes current.

To populate a multiple workspace display:

➔ `addWorkspace(KbWorkspace` *`workspace`*`, boolean` *`keepHistory`*`)`

If *`keepHistory`* is `true`, the workspace view caches the information it creates to display the KB workspace, and it reuses that information each time that workspace becomes the current workspace in the view. If *`keepHistory`* is `false`, the view recreates that information each time that workspace becomes current.

In a multiple workspace panel, a newly added KB workspace becomes the current workspace by default. You can toggle this default as needed.

To set whether a MultipleWorkspacePanel makes a new workspace current:

➔ `setShowWorkspace(boolean` *`showWorkspace`*`)`

If *`showWorkspace`* is `true`, a KB workspace subsequently added to the panel automatically becomes current and, therefore, visible. If the argument is `false`, subsequently adding a KB workspace does not make it current. The default is to make a newly added KB workspace current.

To obtain the current value of `setShowWorkspace`:

➔ `getShowWorkspace(boolean` *`showWorkspace`*`)`

If `setShowWorkspace()` has never been called, the value returned is `true`.

Automatically Populating a Multiple Workspace Panel

A `MultipleWorkspacePanel` implements the `com.gensym.ntw.WorkspaceShowingListener` interface. Therefore, by registering the panel as a listener, once it is connected to a `TwConnector` or `TwGateway`:

- A show workspace action in G2 automatically adds the KB workspace to the panel.
- A hide workspace action in G2 automatically removes the KB workspace from the panel.

For information on `WorkspaceShowingListener`, see "Subscribing to Workspace Show and Hide Events" on page 82.

When a show workspace action causes a KB workspace to be added to a multiple workspace panel, the action cannot specify whether the panel should cache or re-create the information it needs to display that workspace, as described under "Populating a Multiple Workspace Display" on page 163.

To specify display information caching for automatically added workspaces:

➔ `setKeepInHistory(boolean keepInHistory)`

If *`keepInHistory`* is `true`, the multiple workspace panel caches display information for any KB workspace that is added automatically thereafter. If the argument is `false` at the time the workspace was added, the panel re-creates display information each time the KB workspace becomes current.

In effect, *`keepInHistory`* supplies globally the same information that the *`keepHistory`* argument to the `addWorkspace` method supplies for individual workspace views. You can toggle *`keepInHistory`* at any time, but the change does not affect any KB workspace that is already in the panel.

To obtain the current value of `setKeepInHistory`:

➔ `getKeepInHistory()`
`–> boolean` *`keepInHistory`*

If `setKeepInHistory()` has never been called, the value returned is `true`.

Removing a KB Workspace from a Workspace View

The KB workspace in a single workspace view workspace view is automatically removed when a new workspace is added to the view. You can remove it explicitly, if needed. You can also remove a KB workspace from a multiple workspace display.

To remove explicitly a KB workspace from a single workspace view:

➔ `setWorkspace(null)`

To remove a KB workspace from a multiple workspace display:

➔ `removeWorkspace(KbWorkspace` *`workspace`*`)`
`–> boolean` *`workspaceRemoved`*

If *`workspace`* exists in the display's workspace list, it is removed and the method returns `true`. If *`workspace`* exists more than once in the list, only the first instance is removed. If *`workspace`* does not exist in the list, the method has no effect on the view and returns `false`.

Obtaining KB Workspaces

You can obtain a KB workspace from a connection or from a workspace view. You can also test whether a workspace view contains a KB workspace.

Obtaining a KB Workspace from a Connection

Workspace views display objects of class `com.gensym.classes.KbWorkspace`, the Java equivalent of a G2 kb-workspace.

One way of obtaining a KB workspace is through a `com.gensym.jgi.G2Gateway` connection. When obtaining a KB workspace through a connection, you cast the return value, which is a `com.gensym.classes.Item`, to a `KbWorkspace`.

You can also use a connection to obtain a list of named KB workspaces, as described in "Getting a List of Named Workspaces" on page 90.

For information on creating a connection, see Part II, "Connecting to G2" on page 23.

To obtain a KB workspace from a connection:

➔ Call this method on a `G2Gateway`:

```
getUniqueNamedItem(Symbol itmClass, Symbol itmName)
throws G2AccessException
–> Item item
```

Obtaining a KB Workspace(s) from a Workspace View

You can obtain the KB workspace in a single workspace view, or an array containing every KB workspace in a multiple workspace display.

To obtain the KB workspace contained in a single workspace view:

➔ `getWorkspace()`
`–> KbWorkspace` *`workspace`*

To obtain all KB workspaces contained in a multiple workspace display:

➔ `getWorkspaces()`
`–> KbWorkspace` *`workspaces`*`[]`

The returned array contains the KB workspaces in the same order that they were added to the display. If the display contains no workspaces, the array is empty.

Obtaining the Current KB Workspace from a Multiple Workspace Display

You cannot directly obtain the current KB workspace from a multiple workspace display, but you can obtain it indirectly.

To obtain the current KB workspace from a multiple workspace display:

1. Use `getCurrentView()` on the view to obtain a single workspace view that contains the workspace, as described under "Obtaining a Single Workspace View from a Multiple Workspace View" on page 167.
2. Use `getWorkspace()` on the single workspace view to obtain the KB workspace itself, as described under "Obtaining a KB Workspace(s) from a Workspace View" on page 166.

Polling a Multiple Workspace Display for a Named KB Workspace

You can interrogate a multiple workspace display to see whether it already contains a specified named KB workspace.

To determine whether a multiple workspace display contains a named KB workspace:

➔ `contains(KbWorkspace` *`workspace`*`)`
`–> boolean` *`workspaceExists`*

The returned value is `true` if *`workspace`* exists in the display, and `false` otherwise.

Obtaining a Single Workspace View from a Multiple Workspace View

You can obtain the current KB workspace from a multiple workspace view as a single workspace view. This view is functionally identical to the view you could have created by instantiating `ScalableWorkspaceView` and using `setWorkspace()` to populate it with that `KbWorkspace`.

To obtain a single workspace view from a multiple workspace view:

➔ `getCurrentView()`
`–> ScalableWorkspaceView` *`singleWorkspaceView`*

Be careful not to confuse this method with `getWorkspace()` or `getWorkspaces()`, which are described under "Obtaining a KB Workspace(s) from a Workspace View" on page 166. Those methods return KB workspaces, not workspace views.

Controlling KB Workspace Visibility

The KB workspace in a single workspace view is necessarily current and therefore always visible, so no methods exist that control its visibility.

You can set the current KB workspace in a multiple workspace display by specifying the desired workspace, or by traversing the display's workspace list. You can also set a newly added KB workspace to become current automatically, as described under "Populating a Multiple Workspace Display" on page 163.

To set the current KB workspace in a multiple workspace display:

➔ `setCurrentWorkspace(KbWorkspace` *`workspace`*`)`

The specified *`workspace`* becomes current in the view. If *`workspace`* is not contained in the view, the call throws a `com.gensym.wksp.WorkspaceNotAddedException`

To make the next KB workspace current in a multiple workspace display:

➔ `nextWorkspace()`

The next KB workspace in the workspace list becomes current. The method cycles through the list; if the last KB workspace was current, the first becomes current. If the display is empty, the call has no effect.

To make the previous KB workspace current in a multiple workspace display:

➔ `previousWorkspace()`

The previous KB workspace in the workspace list becomes current. The method cycles through the list; if the first workspace was current, the last becomes current. If the display is empty, the call has no effect.

Scrolling Workspace Views

You can scroll all workspace views, whether or not they include scrollbars; scrollbars serve only to provide an interactive interface to the scroll methods.

Adding and Removing Scrollbars

Single and multiple workspace views lack scrollbars, by default. Multiple workspace panels have scroll bars, by default, which you cannot hide or remove.

To add a scrollbar to a single or multiple workspace view:

1 Create a `com.gensym.wksp.WorkspaceViewScrollbar`:

```
WorkspaceViewScrollbar
    (ScalableWorkspaceView parent,
     int orientation,
     int value,
     int visible,
     int minimum,
     int maximum)
```

Argument	Description
`parent`	The workspace view that the scrollbars should control.
`orientation`	`Scrollbar.HORIZONTAL` or `Scrollbar.VERTICAL`
`value`	Initial scroll position.
`visible`	The visible amount of the scrollbar, which is the range of values represented by the width of the scroll bar's bubble. This value is used to determine the scrollbar's block increment.
`minimum`	The minimum value of this scrollbar.
`maximum`	The maximum value of this scrollbar.

2 Add the scrollbar to the workspace view:

```
addScrollbar(viewScrollbar scrollbar, boolean isHorizontal)
```

If *`isHorizontal`* is `true`, the scrollbar becomes a horizontal scrollbar; if `false`, it becomes a vertical scrollbar. If you need to remove the scrollbar later, keep a pointer to it.

For example:

```
// Create the ScalableWorkspaceView
wkspView = new ScalableWorkspaceView (wksp);

// Create the scrollbars
Rectangle initialBounds = wkspView.getBounds();
int initialLeft = initialBounds.x;
int initialTop = initialBounds.y;
int initialWidth = initialBounds.width;
int initialHeight = initialBounds.height;
hscroll = new WorkspaceViewScrollbar (wkspView,
   Scrollbar.HORIZONTAL, 0, 1, initialLeft,
   initialLeft + initialWidth);
vscroll = new WorkspaceViewScrollbar (wkspView,
   Scrollbar.VERTICAL, 0, 1,initialTop,
   initialTop + initialHeight);

// Associate the scrollbars with the view
wkspView.addScrollbar(vscroll, false);
wkspView.addScrollbar(hscroll, true);

// Add all UI components to the Frame
add (wkspView, BorderLayout.CENTER);
add (vscroll, BorderLayout.EAST);
add (hscroll, BorderLayout.SOUTH);
```

To remove a scrollbar from a multiple workspace view:

1. Keep a pointer to the scrollbar that you added with `addScrollbar()`.
2. Call this method:

   `removeScrollbar(viewScrollbar` *`scrollbar`*`, boolean` *`isHorizontal`*`)`

Set *`isHorizontal`* to `true` for a horizontal scrollbar, or `false` for a vertical scrollbar.

Setting Scrolling Increments

All types of workspace views provide scrolling in two granularities: **scroll unit** and **scroll block**. Both of these are measured in pixels. By convention, a scroll unit is smaller than a scroll block, but nothing requires this. The default values are:

- Scroll Unit: 10
- Scroll Block: 40

You can set or retrieve the value of either type of scroll increment.

To set the scroll unit increment of a workspace view:

➔ `setUnitIncrement(integer` *`increment`*`)`

To obtain the current scroll unit increment of a workspace view:

➔ `getUnitIncrement()`
`–> integer` *`increment`*

To set the scroll block increment of a workspace view:

➔ `setBlockIncrement(integer` *`increment`*`)`

To obtain the current scroll block increment of a workspace view:

➔ `getBlockIncrement()`
`–> integer` *`increment`*

Incrementally Scrolling a KB Workspace

You can scroll the KB workspace that is visible in any workspace view:

- By a unit or a block.
- Up, down, left, or right.

Eight scroll methods provide the eight permutations of increment and direction. Their names follow a simple grammar.

To scroll a KB workspace incrementally:

➔ `scroll{Unit|Block}{Up|Down|Left|Right}()`

For example, to scroll one unit up:

```
scrollUnitUp()
```

To scroll one block left:

```
scrollBlockLeft()
```

Note Scrolling a multiple workspace display scrolls only the current workspace view; the scroll position of other views of the same KB workspace is unaffected. Similarly, the scroll position of the underlying KB workspace is unaffected.

Working with Workspace View Elements

When a KB workspace is displayed in a workspace view, each item on the underlying kb-workspace is represented by an object that implements this interface:

```
com.gensym.wksp.WorkspaceElement
```

An implementation of this interface is called a **workspace view element**, or, for brevity, a **workspace element**.

The `WorkspaceElement` interface extends these two interfaces:

- `com.gensym.wksp.ItemView` — Provides a method for getting the `com.gensym.classes.Item` that the workspace element represents.
- `com.gensym.util.ItemListener` — A listener for receiving notification when a G2 item is modified or deleted, or when the listener interface is added.

Thus, adding, changing, or deleting an item on the underlying KB workspace in G2 automatically creates, changes, or deletes a corresponding element in every workspace view. Handshaking between G2 and the Telewindows2 Toolkit client ensures that the KB workspace and its views remain synchronized.

You can obtain the elements in the current KB workspace of a workspace view, then select and change some or all of them in various ways. Obtaining and selecting workspace view elements does not affect G2. Changing them transmits a request to G2, which makes the change to the corresponding items and propagates it to all workspace views.

Operations on elements in a multiple workspace display affect only the current KB workspace. If that workspace ceases to be current, any selected elements on it cease to be selected. Making the KB workspace current again does not restore the selection.

Obtaining All Workspace View Elements

You can obtain a list of all elements in the KB workspace of a single workspace view, or the current KB workspace of a multiple workspace view. You cannot obtain a list of all elements from a multiple workspace panel.

To obtain the elements in a single workspace view:

➔ `getElements()`
`–> workspaceElement` *`java.util.Enumeration`*

To obtain the elements in the current workspace of a multiple workspace view:

1. Extract the current KB workspace as a single workspace view, as described under "Obtaining a Single Workspace View from a Multiple Workspace View" on page 167.
2. Proceed as described for a single workspace view.

Obtaining the Workspace Element for an Item

You can obtain the `WorkspaceElement` associated with a particular `com.gensym.classes.Item` on a workspace view or a multiple workspace panel.

To obtain the workspace element for an item:

➔ `findElement(Item` *`item`*`)`
`–> WorkspaceElement` *`workspaceElement`*

Obtaining the Item Associated with a Workspace Element

You can obtain the item associated with a workspace view element and, thereby, have access to all methods for the particular class of item. For example, if the workspace element is an instance of a `com.gensym.classes.TrendChart`, you can:

- Get or set any of the attributes of the trend chart from the client.
- Call any of the methods on its superior class, which is `com.gensym.classes.Item`, such as `move`, `delete`, `enable`, or `disable`.
- Receive notification of `itemModified`, `itemDeleted`, and `receivedInitialValues` events.

For information on item events, see `com.gensym.util.ItemListener`, which is part of G2 JavaLink.

To obtain the item associated with a workspace element:

1. Get the list of workspace elements from a single workspace view.
2. Call this method on a single workspace element:

```
getItem()
```

For example, the following method calls `enable` on the `com.gensym.classes.Item` that is currently selected in a workspace view:

```
private ScalableWorkspaceView workspaceView = null;
private WorkspaceElement[] currentSelection = null;

//Create workspace view
ScalableWorkspaceView workspaceView =
   new ScalableWorkspaceView();

//Enable currently selected item
private void handleEnableSelectionCommand() {
    try {
      if (currentSelection.length == 0)
         ScalableWorkspaceView.getWorkspace().enable();
      else {
         for (int i=0; i<currentSelection.length; i++)
           currentSelection[i].getItem().enable();
      }
    } catch (G2AccessException gae) {
    }
  }
```

Working with Selections

A single workspace view maintains a list of currently selected items called a **selection**. You can select individual or all workspace view elements, add workspace view elements to the selection, and remove individual or all workspace view element from the selection.

You can obtain the list of selected workspace view elements, move the selected elements, or delete the selected elements.

Clients can add themselves as listeners to receive notification when the selection changes.

Selecting Workspace View Elements

You can select and deselect workspace view elements with the mouse, as described under "Selecting and Deselecting Objects" on page 137. You can also select and deselect elements in any workspace view by calling methods that indicate the element or elements affected. These methods operate on the current KB workspace in a multiple workspace display. They have no effect on any other workspaces.

Selecting an element that was already selected or deselecting an element that was not selected has no effect.

To select a specified workspace element:

➔ `addElementToSelection(WorkspaceElement element)`

The specified element becomes selected. The selection status of other elements is unaffected.

To select several workspace elements:

➔ `addElementsToSelection(WorkspaceElement element[])`

The specified elements become selected. The selection status of other elements is unaffected.

To select several workspace elements and deselect all others:

➔ `selectElements(WorkspaceElement element[])`

Any currently selected elements cease to be selected, and the specified elements become selected. The effect is the same as calling `clearSelection()` before calling `addElementsToSelection()`.

To select all workspace elements:

➔ `selectAll()`

All elements become selected.

To deselect a selected workspace element:

➔ `removeElementFromSelection(WorkspaceElement element)`

The specified element becomes unselected. The selection status of other elements is unaffected.

To deselect a list of workspace elements:

➔ `removeElementsFromSelection(WorkspaceElement element[])`

The specified elements become unselected. The selection status of other elements is unaffected.

To deselect all selected workspace elements:

➔ `clearSelection()`

All elements become unselected.

Obtaining Selected Elements

You can obtain a list of all selected elements in the KB workspace in a single workspace view, or the current KB workspace in a multiple workspace display.

To obtain the selected elements in a single workspace view:

➔ `getSelection()`
`–> WorkspaceElement` *`elements`*`[]`

Manipulating Selected Elements

When one or more elements are selected on the current KB workspace, you can:

- Move the selection.
- Delete the selection.

To move the selected element(s):

➔ `moveSelection(int deltaX, int deltaY, boolean enlargeWorkspace)`

This method moves the selection by `deltaX` pixels horizontally and `deltaY` pixels vertically. Use a positive integer to specify motion right or down, and use a negative integer to specify motion left or up.

If `enlargeWorkspace` is `false`, the movement is constrained by the borders of the KB workspace. If it is `true`, the KB workspace expands as needed to allow the move.

To delete the selected element(s):

➔ `deleteSelection()`

This method deletes all currently selected items.

Handling Selection Events

Clients can receive notification when the list of selected workspace view elements changes. To do this, they must add themselves as a `com.gensym.ntw.util.SelectionListener`.

To listen for selection events in a single workspace view:

➔ `addSelectionListener(SelectionListener selectionListener)`

Working with Collections

A multiple workspace panel maintains a list of KB workspaces that it can show, called a **collection**. Clients can receive notification when the collection changes by adding themselves as a `com.gensym.ntw.util.CollectionListener`.

To listen for collection events in a multiple workspace panel:

➔ `addCollectionListener(CollectionListener collectionListener)`

Scaling Workspace Views

A `ScalableWorkspaceView` supports scaling by setting the scale of the x and y axes or scaling the view to fit the current workspace document. Once you have scaled the workspace view, you can also set the size of the workspace view and scroll the workspace view to a particular x, y location.

To set the x and y scale of a scalable workspace view:

➔ `setScale(double` *`scaleX`*`, double` *`scale`*`-Y)`

or

➔ `setScaleX(double` *`scale`*`X)`
`setScaleY(double` *`scaleY`*`)`

To set the scalable workspace to fit the current workspace document:

➔ `setScaledToFit(boolean` *`scaledToFit`*`)`

To get the current scale of a scalable workspace view:

➔ `getScaleX()`
`getScaleY()`
`–> double` *`scaleFactor`*

To determine if the workspace view is scaled to fit:

➔ `isScaledToFit()`
`–> boolean` *`scaledToFit`*

Workspace View Example

You can add a workspace view to a container and display a KB workspace in the view with relatively little Java programming. The code on the following pages illustrates the essential techniques by showing the code in the wkspdemo.

```
package com.gensym.demos.wkspdemo;

import com.gensym.jgi.*;
import com.gensym.ntw.*;
import com.gensym.wksp.*;
import com.gensym.draw.*;
import com.gensym.util.*;
import com.gensym.classes.KbWorkspace;

import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

/**
 * This class creates and displays a frame that holds a workspace view.
 */
public class WorkspaceFrame extends Frame {

  private static final Symbol DEMO_ = Symbol.intern ("DEMO");
  private transient Image offscreen;
  private transient Graphics og;

  public static void main (String[] args) {
    // Create the frame with a ScalableWorkspaceView in it
    Frame f = new WorkspaceFrame ("Workspace Demo");
    // Resize the frame
    f.setBounds (100, 100, 500, 400);
    // Display it
    f.setVisible (true);
  }

  private String host = "localhost";
  private String port = "1111";
  private Symbol wkspName = DEMO_;
  private ScalableWorkspaceView wkspView;

  WorkspaceFrame (String title) {
    super (title);
    // Exit when the user clicks the close button
    addWindowListener (new WindowAdapter () {
      public void windowClosing (WindowEvent we) {
   // shut up shop
   System.exit (0);
      }
    });
    displayWorkspace (host, port, wkspName);
  }
```

```
/**
 * This class creates, customizes, and populates a Single Workspace View.
 */
private void displayWorkspace (String host, String port, Symbol wkspName_)
{
    ViewScrollbar vscroll, hscroll;
    try {
      // Make a connection and login
      TwAccess cxn = TwGateway.openConnection (host, port);
      cxn.login ();
      // Get the KB-WORKSPACE to display
      KbWorkspace wksp = (KbWorkspace)cxn.getUniqueNamedItem
              (com.gensym.util.symbol.G2ClassSymbols.ITEM_, wkspName_);
      // Listen for deletion of workspace
      wksp.addItemListener (new ViewDisposer ());

      // Create the ScalableWorkspaceView
      wkspView = new ScalableWorkspaceView (wksp);

      // Create the scrollbars
      Rectangle initialBounds = wkspView.getBounds();
      int initialLeft = initialBounds.x;
      int initialTop = initialBounds.y;
      int initialWidth = initialBounds.width;
      int initialHeight = initialBounds.height;
      hscroll = new ViewScrollbar (wkspView, LWScrollbar.HORIZONTAL, 0, 1,
                initialLeft, initialLeft + initialWidth);
      vscroll = new ViewScrollbar (wkspView, LWScrollbar.VERTICAL, 0, 1,
                initialTop, initialTop + initialHeight);

      // Associate the scrollbars with the view
      wkspView.addScrollbar(vscroll, false);
       wkspView.addScrollbar(hscroll, true);

      // Add all UI components to the Frame
      add (wkspView, BorderLayout.CENTER);
      add (vscroll, BorderLayout.EAST);
      add (hscroll, BorderLayout.SOUTH);
    } catch (Exception e)

      // Display an error dialog if something went wrong
      new com.gensym.dlg.MessageDialog (this,
              "Error!",
              false,
              e.getMessage (),
              (com.gensym.dlg.StandardDialogClient)null).setVisible (true);
    }
}
```

```
/**
   * Override update to *not* erase the background before painting
   * This reduces flicker.
   */
  public void update(Graphics g) {
    paint(g);
  }

  /**
   * Implement double buffering by painting children into an offscreen
   * image and then blast entire image at once.
   */
  public void paint(Graphics g) {
    if(offscreen == null) {
      offscreen = createImage(getSize().width, getSize().height);
      og = offscreen.getGraphics();
    }
    og.setClip(0,0,getSize().width, getSize().height);
    super.paint(og);
    g.drawImage(offscreen, 0, 0, null);
  }

  /**
   * Handle invalidation of double-buffering data
   */
  public void invalidate () {
    super.invalidate ();
    offscreen = null;
    og = null;
  }

/**
   * Provide the item listener
   */

class ViewDisposer implements com.gensym.util.ItemListener {
    public void receivedInitialValues (ItemEvent e) {
      // No action to take
    }
    public void itemModified (ItemEvent e) {
      // Don't care
    }
    public void itemDeleted (ItemEvent e) {
      // Dispose of the ScalableWorkspaceView
      remove (wkspView);
    }
  }
}
```

13

Customizing Popups for Selected Items

Describes how to customize the popup menu that gets created for selected items in a WorkspaceView.

Introduction

By default, when you right-click an item in a `ScalableWorkspaceView`, you get a popup menu that includes instances of these commands, which display these menu choices:

This command...	Contains these menu choices...
`com.gensym.wksp.SystemMenuChoiceCommands`	Cut Copy Paste Name Delete Disable Create Subworkspace Rotate/Reflect Color Drop to Bottom Lift to Top Describe Edit Attribute Display Layout Properties
`com.gensym.wksp.UserMenuChoiceCommands`	All user menu choices that have been defined for the class.

In addition, if you right-click the `ScalableWorkspaceView` item itself, the popup includes this command and associated menu choice:

This command...	Contains this menu choices...
`com.gensym.wskp.CreationCommands`	New Item

You can override the popup menu choices for selected items in all `ScalableWorkspaceViews` of an application or in a particular `ScalableWorkspaceView`. For example, you might want the item popup menu to show only user menu choices, or you might want to add a custom command to the popup, which appears only in the client or is invoked locally.

To do this, you use a `SelectionCommandGenerator` to generate popup menu choices of type `SelectionCommand`.

Customizing the item popup menu in this way represents an alternative to using G2 item configurations to restrict menu choices for items based on user mode.

SelectionCommandGenerator

You have two options for using a `SelectionCommandGenerator` to generate popup menu choices for items on workspaces:

To override the popup menu choices for items on...	Do this...
All workspaces in the application	Set the `SelectionCommandsFactory` and implement the factory to determine which commands should appear.
Specific workspaces in the application	Register the commands that should appear for a particular `ScalableWorkspaceView`.

SelectionCommand

The item popup commands that the factory generates or that you explicitly register must be implementations of the `SelectionCommand` interface. You can specify any of the commands that appear by default on an item popup menu, as described earlier, or you can implement your own `SelectionCommand`.

The `SelectionCommand` interface is responsible for setting the `ScalableWorkspaceView` of the selected items.

`SelectionCommand` is a subclass of the `com.gensym.ui.StructuredCommand` interface, which provides an interface for defining logical groupings of actions, a hierarchical structure of actions, or a dynamic number of actions.

For more information on the `StucturedCommand` interface, see Chapter 5 "Creating Menus and Toolbars" in the *Telewindows2 Toolkit Java Developer's Guide: Application Classes*.

MenuChoiceHandler

By default, a `ScalableWorkspaceView` invokes user menu choices remotely in the G2 server, which then updates the representation of the view in the client, as needed. You can provide local invocations of user menu choices so they are invoked only in a particular client. To do this, you implement a `MenuChoiceHandler`, which specifies the behavior of the user menu choice when it is invoked on the client.

Packages Covered

com.gensym.wksp

Interfaces

```
MenuChoiceHandler
SelectionCommand
SelectionCommandsFactory
```

Classes

```
CreationCommands
SelectionCommandGenerator
SystemMenuChoiceCommands
UserMenuChoiceCommands
```

Relevant Demos

The demos in the following directory, depending on your platform, show examples of customizing popup menus for selected items on workspaces:

NT: `%SEQUOIA_HOME%\classes\com\gensym\demos\custompopups\`

UNIX: `$SEQUOIA_HOME/classes/com/gensym/demos/custompopups/`

For information about these classes...	See...
`WorkspaceFrame` `CustomSelectionCommandsFactory`	"Displaying a Popup Menu with User Menu Choices Only" on page 185.

`WorkspaceFrame2`
`CustomSelectionCommandsFactory2`
`EditCommandsImpl`
	"Displaying Custom Commands in a Popup Menu" on page 188.

`SimpleWorkspaceApplication`	"Registering Popup Menu Choices for Individual Workspaces" on page 193.

`WorkspaceFrame3`
`CustomSelectionCommandsFactory3`
`LocalMenuChoiceHandler`
	"Invoking System-Defined User Menu Choices Locally in the Client" on page 196.

Displaying a Popup Menu with User Menu Choices Only

Suppose you want popups for selected items to display user menu choices only; you do not want them to display system menu choices. Furthermore, if the selected item is a `ScalableWorkspaceView`, you do not want the item popup menu to display commands for creating items.

To do this, set the `SelectionCommandsFactory` to generate the menu choices that appear in the popup menu for selected items in any `ScalableWorkspaceView` of your application.

To display a popup menu with user menu choices only:

1. Create a class that implements `SelectionCommandsFactory`, whose name corresponds to the factory you just set.

   In the previous example, you would create a class named `CustomSelectionCommandsFactory`.

2. Provide an implementation of the `createCommands` method of the `SelectionCommandsFactory` interface, which returns an array of `SelectionCommand` objects that contains a single element: an instance of a `UserMenuChoiceCommands`.

   The `createCommands` method takes as its argument a `ScalableWorkspaceView`, which is workspace view for which the factory generates item popup menus.

3. In the constructor for your application, call the `setSelectionCommandsFactory` static method on `com.gensym.wksp.SelectionCommandGenerator`, passing an implementation of the `com.gensym.wksp.SelectionCommandsFactory` interface as the argument.

Note Ensure that this method is called *before* the application downloads any workspaces; otherwise, the application will only use the factory to generate popup commands for selected items on subsequently downloaded workspaces.

For example, this method call sets the factory to an instance of a `CustomSelectionCommandsFactory`, an implementation of `SelectionCommandsFactory`:

```
SelectionCommandGenerator.setSelectionCommandsFactory
   (new CustomSelectionCommandsFactory());
```

Example

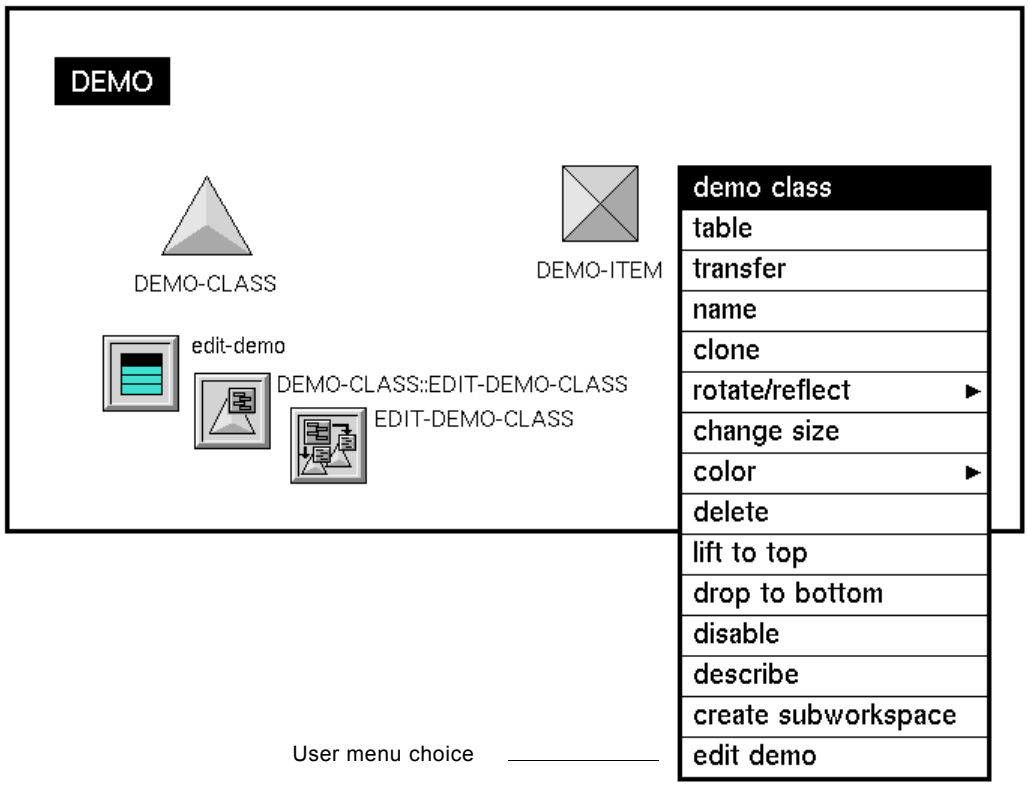
To see this example online, run `com.gensym.demos.docs.custompopups.WorkspaceFrame`, described in "Relevant Demos" on page 184.

To cause the factory to generate popup menus for selected items that contain only user menu choices, you would implement the `createCommands` method as follows:

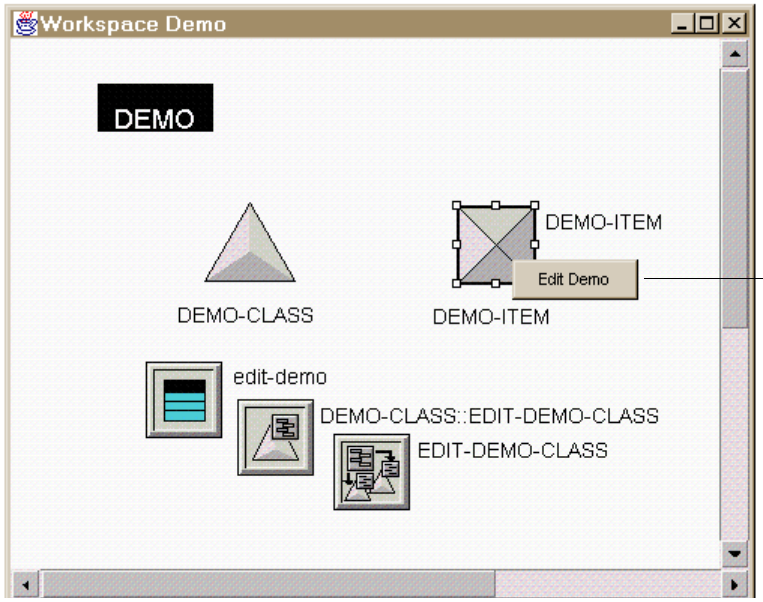
```
public SelectionCommand[] createCommands (ScalableWorkspaceView view) {
   return new SelectionCommand[] {
      new UserMenuChoiceCommands ()};
}
```

The resulting popup menu for selected items, as well as the `ScalableWorkspaceView` itself contains only user menu choices.

The following figure shows a KB workspace named demo, which contains an item named demo-item that defines a user-menu-choice named edit-demo. In G2, the item popup menu display all system-defined and user-defined user menu choices:



In the client, the item popup menu displays only the Edit Demo menu choice:



Choosing Edit Demo in the client invokes the user menu choice in G2, which updates the attributes of the item as the G2 attribute table shows:

DEMO-ITEM-1, a demo-class	
Notes	OK
Item configuration	none
Names	DEMO-ITEM-1
Integer attribute	10
Text attribute	"Hello"
Enumerated attribute	foo
Boolean attribute	false
Item list attribute	an item-list

Displaying Custom Commands in a Popup Menu

The `createCommands` method of the `SelectionCommandsFactory` interface returns an array of implementations of the `SelectionCommand` interface. This array can include any of the following commands or any other commands that you create:

- `com.gensym.wksp.SystemMenuChoiceCommands`
- `com.gensym.wksp.UserMenuChoiceCommands`
- `com.gensym.wksp.CreationCommands`

For information on the menu choices that these commands define, see "Introduction" on page 182.

The array can also contain any user-defined implementation of the `SelectionCommand` interface. This interface extends `com.gensym.ui.StructuredCommand` and provides one additional method, `setWorkspaceView`. This method takes as its argument the `ScalableWorkspaceView` in which the command is to be included.

Typically, a `SelectionCommand` defines actions that apply to the current selection, in which case, the command must also implement the `SelectionListener` interface or use a `SelectionAdapter`. However, this is not a requirement; the actions can be independent of the current selection, and the command need not implement this listener.

For example, suppose you want all items in the application to provide the Cut/Copy/Paste commands in their popup menu, and no others.

One way to do this is to create a command that extends `AbstractStructuredCommand`. By extending `AbstractStructuredCommand`, you can create the command structure in the constructor, and you can call methods to set the properties of each command key. Otherwise, you must implement these features yourself in the abstract methods of the `SelectionCommand` interface.

By default, popup menus represent command keys by using the textual description only; by default, they do not represent them by using the iconic description. However, to make the command more generic, we recommend that you provide an icon anyway.

Implementing a `SelectionCommand` is similar to implementing a `Command`. For more information, see these sections in Chapter 5 "Creating Menus and Toolbars" in the *Telewindows2 Toolkit Java Developer's Guide: Application Classes*:

- "Implementing the Command Interface"
- "Creating Commands with a Structure"

To display custom commands in a popup menu:

1. Create a class that:
   - Implements the `com.gensym.wksp.SelectionCommand` interface.
   - If the action of the command applies to the current selection, implements the `com.gensym.ntw.util.SelectionListener` interface or uses a `SelectionAdapter`.
   - Optionally, extends the `com.gensym.ui.AbstractStructuredCommand` interface, for convenience.
2. Define the `AbstractStructuredCommand` as follows:
   - Define command keys for each action of the command.
   - In the constructor, call the constructor for the superior class, passing in an array of `CommandInformation` objects for each command key.
   - Implement the `actionPerformed` method to specify the behavior of each command key.
3. Implement the `setWorkspaceView` method of the `SelectionCommand` interface to determine the behavior of the command when the `ScalableWorkspaceView` gets set.

   Typically, this method:
   - Sets the current `ScalableWorkspaceView` to workspace view that you pass in as the argument to the method.
   - Listens to the selection on the current `ScalableWorkspaceView` if the command applies to the current selection. For example, the cut command cuts the current selection.
   - If the action of the command applies to the current selection, gets the current selection from the `ScalableWorkspaceView`.
4. Implement the `selectionChanged` method of the `SelectionCommand` interface to determine the behavior of command when the selected item changes, or use a `SelectionAdapter`.

   Typically, this method:
   - Gets the current selection.
   - Updates the availability of the command keys.

5 Update the properties of each command key, such as availability, as needed.

   For example, if you extend `AbstractStructuredCommand`, you can call `setAvailable` to set the availability of each command key. Note that if you do this, you do not need to override `isAvailable`.

6 Follow the steps under "Displaying a Popup Menu with User Menu Choices Only" on page 185, substituting `UserMenuChoiceCommands` with your implementation of the `SelectionCommand` interface.

Example

To see this example online, run `com.gensym.demos.docs.custompopups.WorkspaceFrame2`, described in "Relevant Demos" on page 184.

This example creates a command named `EditCommandsImpl`, with command keys called `CUT_SELECTION`, `COPY_SELECTION`, and `PASTE_SELECTION`:

```
public final class EditCommandsImpl extends AbstractStructuredCommand
    implements SelectionCommand, SelectionListener {

    public static final String CUT_SELECTION = "cut";
    public static final String COPY_SELECTION = "copy";
    public static final String PASTE_SELECTION = "paste";

    private static final String shortResource = "ShortCommandLabels";
    private static final String longResource = "LongCommandLabels";
    private static Resource shortBundle =
        Resource.getBundle("com.gensym.demos.custompopups.
            ShortCommandLabels");
    private static Resource longBundle =
        Resource.getBundle("com.gensym.demos.custompopups.
            LongCommandLabels");
    private Resource i18n =
        Resource.getBundle("com.gensym.demos.custompopups.Errors");

    public Frame frame;
    private ScalableWorkspaceView workspaceView = null;
    private WorkspaceElement[] currentSelection = null;

    private static final int offset = 10;
    private static boolean clipboardDataWasCut;
```

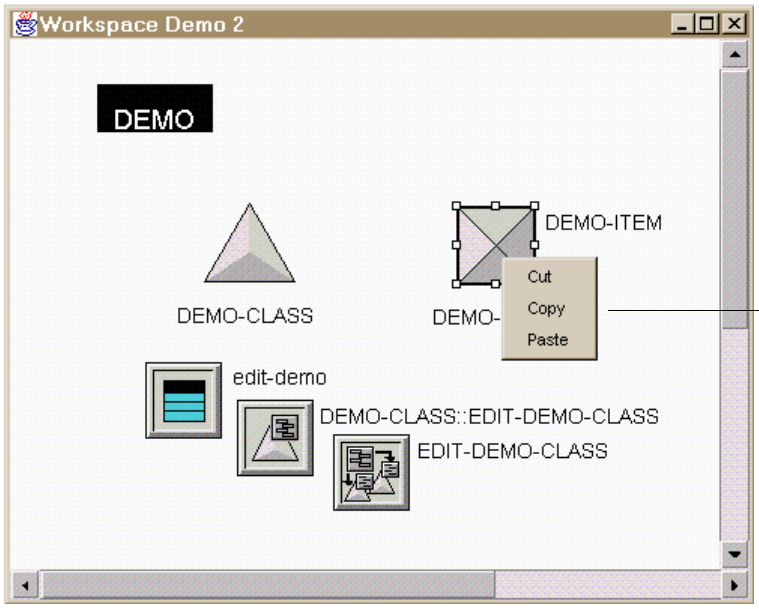
```
//Constructor
public EditCommandsImpl(){
    super(new CommandInformation[]{
        new CommandInformation(CUT_SELECTION, true,
            shortResource, longResource,
            null, null, null, true),
        new CommandInformation(COPY_SELECTION, true,
            shortResource, longResource,
            null, null, null, true),
        new CommandInformation(PASTE_SELECTION, false,
            shortResource, longResource,
            null, null, null, true)});
}

//Action of the command
public void actionPerformed(ActionEvent event) {
    String cmdKey = event.getActionCommand();
    if (cmdKey.equals(CUT_SELECTION)) {
        //cutSelectionToClipboard is an undocumented method.
        workspaceView.cutSelectionToClipboard();
        clipboardDataWasCut = true;
    }
    else if (cmdKey.equals(COPY_SELECTION)) {
        //copySelectionToClipboard is an undocumented method.
        workspaceView.copySelectionToClipboard();
        clipboardDataWasCut = false;
    }
    else if (cmdKey.equals(PASTE_SELECTION)) {
        handlePasteSelectionCommand();
        clipboardDataWasCut = false;
    }
}

private void updateAvailability(){
    if (workspaceView != null) {
        setAvailable(PASTE_SELECTION, true);
            if (currentSelection == null ||
                 currentSelection.length == 0) {
                // There is no selection
                setAvailable(CUT_SELECTION, false);
                setAvailable(COPY_SELECTION, false);
            } else {
                // There is a selection
                setAvailable(CUT_SELECTION, true);
                setAvailable(COPY_SELECTION, true);
            }
    } else {
        setAvailable(CUT_SELECTION, false);
        setAvailable(COPY_SELECTION, false);
        setAvailable(PASTE_SELECTION, false);
    }
}
```

```
    private void handlePasteSelectionCommand() {
        //Handle paste selection
    }

    //BEGIN: SelectionCommand methods
    public void setWorkspaceView(ScalableWorkspaceView workspaceView) {
        if (this.workspaceView != null) {
            this.workspaceView.removeSelectionListener(this);
        }
        this.workspaceView = workspaceView;
        if (this.workspaceView != null) {
            this.workspaceView.addSelectionListener(this);
            currentSelection = workspaceView.getSelection();
        }
        updateAvailability();
    }

    // END: SelectionCommand methods

    // BEGIN: SelectionListener methods
    public void selectionChanged(SelectionEvent event){
        currentSelection = workspaceView.getSelection();
        updateAvailability();
    }
    // END: SelectionListener methods

}
```

To cause the factory to generate popup menus for selected items that contain only the Cut, Copy, and Paste menu choices, you would implement the `createCommands` method of the `SelectionCommandsFactory` interface as follows:

```
public SelectionCommand[] createCommands (ScalableWorkspaceView view) {
    return new SelectionCommand[] {
        new EditCommandsImpl()};
}
```

The resulting popup menu for selected items, as well as the `ScalableWorkspaceView` itself contains only Cut, Copy, and Paste, as this workspace in the client shows:



Registering Popup Menu Choices for Individual Workspaces

Instead of generating menu choices for item popups in all workspaces of an application, you can register commands with the `SelectionCommandGenerator` to generate item popups for specific workspaces.

For example, you might define a single document interface (SDI) application that provides a `MultipleWorkspacePanel` and commands to switch between multiple workspaces, where each workspace provides a different set of popup menu choices for selected items.

You can also use this approach in a multiple document interface (MDI) application that provides multiple document windows in which to view workspaces.

To register popup menu choices for individual workspaces, you must first de-register existing commands that you do not want to appear in the popup menu. Then, you can register individual commands with individual workspaces.

The commands that you register must be implementations of the `SelectionCommand` interface, as described in "Displaying Custom Commands in a Popup Menu" on page 188.

To register popup menu choices for individual workspaces:

1. In the method in which you create a `ScalableWorkspaceView`, call `getSelectionCommandGenerator` on the `ScalableWorkspaceView` to obtain the `SelectionCommandGenerator`.

Tip If you are working with a `MultipleWorkspaceView`, you can call `getCurrentView` to get the `ScalableWorkspaceView` from which you can get the `SelectionCommandGenerator`.

---

2. In the same method, call `getRegisteredCommands` on the `SelectionCommandGenerator` to return a `Vector` of the registered commands.
3. Loop through the `Vector` and call `deregisterCommand` on each command that you do not wish the popup menu to include.
4. For each workspace for which you wish to register commands, call `registerCommand` on the `SelectionCommandGenerator`, passing an instance of an implementation of the `SelectionCommand` interface as the argument.

   For information on commands that implement this interface and implementing your own `SelectionCommand`, see "Displaying Custom Commands in a Popup Menu" on page 188.
5. Repeat step 4. for each command you wish to register for that workspace.

Example

To see this example online, run `com.gensym.demos.docs.custompopups.SimpleWorkspaceApplication`, described in "Relevant Demos" on page 184.

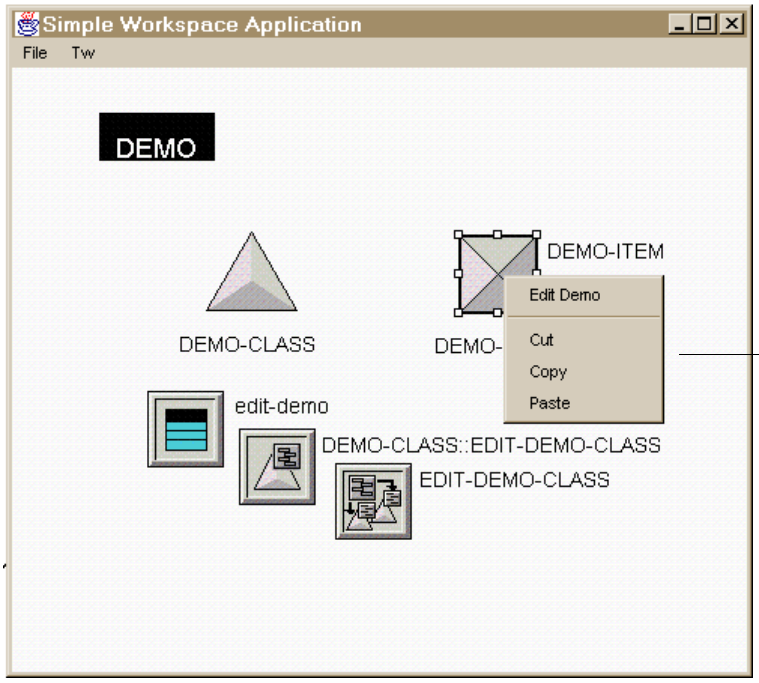
For example, suppose your application has two workspaces, and you want the popup menu for items on one workspace to include user menu choices and a custom menu choice, and you want the popup menu for items on the other workspace to include system menu choices only.

To do this, you might write this code in the method that is responsible for downloading workspaces in the application class:

```
//Deregister default commands
ScalableWorkspaceView wkspView = multiWkspView.getCurrentView();
SelectionCommandGenerator generator =
   wkspView.getSelectionCommandGenerator();
Vector registeredCommands = generator.getRegisteredCommands();
for (int i=0; i<registeredCommands.size(); i++) {
SelectionCommand sc = (SelectionCommand)registeredCommands.
   elementAt (i);
generator.deregisterCommand(sc);
}
```

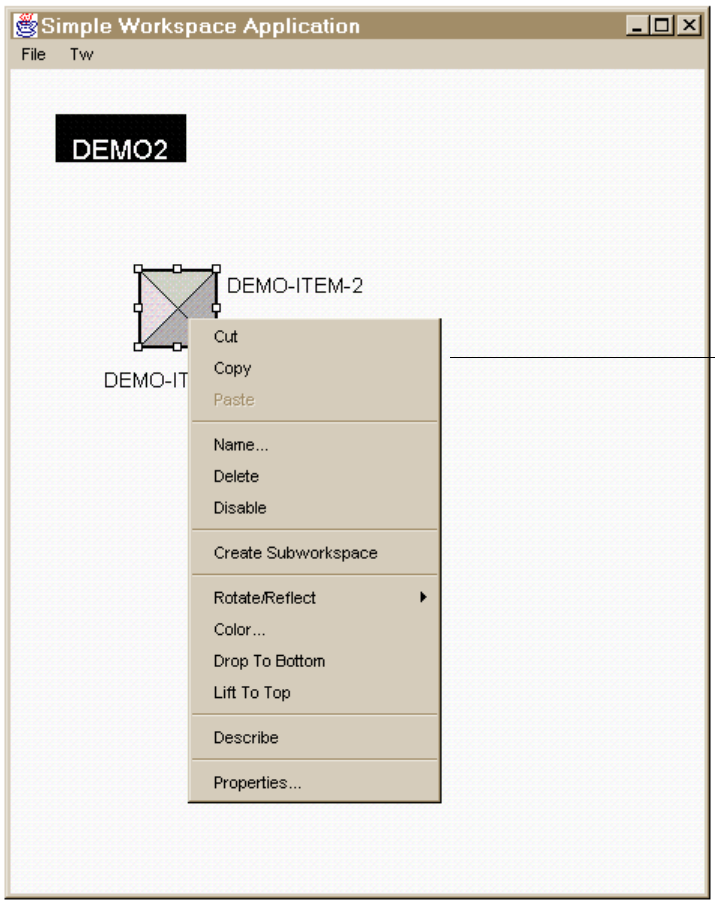
```
//Register individual workspaces with particular commands for
//generating popup menus for selected items.
private static final Symbol DEMO_ = Symbol.intern("DEMO");
private static final Symbol DEMO2_ = Symbol.intern("DEMO2");
if (DEMO_.equals(wkspName){
    generator.registerCommand(new UserMenuChoiceCommands());
    generator.registerCommand(new EditCommandsImpl());
}
if (DEMO2_.equals(wkspName)
    generator.registerCommand(new SystemMenuChoiceCommands());
```

The resulting popup menu for selected items on the Demo workspace contains user menu choices and Cut, Copy, and Paste, as this workspace in the client shows:



Custom popup menu for Demo workspace shows user menu choices and custom menu choices only in the client.

The popup menu for selected items on the Demo2 workspace contains system menu choices only:



Invoking System-Defined User Menu Choices Locally in the Client

You might want to invoke user menu choices only in a particular client. To do this, you implement a `MenuChoiceHandler` that describes the behavior of a particular user menu choice when it is invoked in the client. You then set the `MenuChoiceHandler` by calling a static method on `UserMenuChoiceCommands`.

To invoke user menu choices locally, you must implement a `SelectionCommandsFactory` that creates a `UserMenuChoiceCommands` and set the factory.

The static method that sets the `MenuChoiceHandler` takes a `UserMenuChoice` as one of its arguments. This means that each time your application connects to G2, you must get the `UserMenuChoice` from the connection.

If your application allows connections to a single G2, you only need to get the user menu choice once. However, if your application allows connections to multiple G2s, you need to get the user menu choice each time a connection gets created. This can be cumbersome, especially if you wish to invoke numerous user menu choices locally. Therefore, we recommend that you use this feature only when you wish to invoke a small number of user menu choices. To invoke a large number of user menu choices locally in a multiple connection application, we recommend that you implement your own local invoker.

To invoke a user menu choice locally in the client:

1. Create a class that implements the `com.gensym.wksp.MenuChoiceHandler` interface.
2. Provide an implementation of the `executeMenuChoice` method, which specifies how the user menu choice is to be invoked in the client.

   The handler calls this method when the user invokes the user menu choice in the client.
3. Follow the steps under "Displaying a Popup Menu with User Menu Choices Only" on page 185 to create a factory that generates a `UserMenuChoiceCommands`.
4. In the method in which you create a `ScalableWorkspaceView`, call `getUserMenuChoice` on a `TwGateway` to get the `com.gensym.classes.UserMenuChoice` that you wish to invoke locally.

   For more information, see "Invoking a User Menu Choice" on page 93.
5. In the same method, create an instance of your implementation of the `MenuChoiceHandler` interface.
6. In the same method, call `setLocalMenuChoiceHandler` statically on `UserMenuChoiceCommands`, passing the user menu choice and menu choice handler as arguments.

When the user invokes the specified user menu choice in the client, the local menu choice handler invokes its `executeMenuChoice` method, which invokes the user menu choice only in the client.

Example

To see this example online, run `com.gensym.demos.docs.custompopups.WorkspaceFrame3`, described in "Relevant Demos" on page 184.

The following method shows how to display a workspace named demo and invoke a user menu choice named edit-demo for the demo-class class. The local invocation simply displays a message dialog indicating that it is being invoked locally. This method appears in the application class that displays workspaces.

```
private static final Symbol DEMO_ = Symbol.intern ("DEMO");
private static final Symbol EDIT_DEMO_ = Symbol.intern("EDIT-DEMO");
private static final Symbol DEMO_CLASS_ = Symbol.intern("DEMO-CLASS");

private void displayWorkspace (String host, String port,
    Symbol wkspName_) {
    WorkspaceViewScrollbar vscroll, hscroll;
    try {
    // Make a connection and login
    TwAccess cxn = TwGateway.openConnection (host, port);
    cxn.login ();

    // Get the KB-WORKSPACE to display
    KbWorkspace wksp = (KbWorkspace)cxn.getUniqueNamedItem
        (com.gensym.util.symbol.G2ClassSymbols.KB_WORKSPACE_, wkspName_);

    // Listen for deletion of workspace
    wksp.addItemListener (new ViewDisposer ());

    // Create the ScalableWorkspaceView
    ScalableWorkspaceView wkspView = new ScalableWorkspaceView (wksp);

    //Get UserMenuChoice
    UserMenuChoice userMenuChoice = cxn.getUserMenuChoice
        (EDIT_DEMO_, DEMO_CLASS_);

    //Create a local menu choice handler for user menu choices
    LocalMenuChoiceHandler handler = new LocalMenuChoiceHandler();

    //Set local menu choice handler
    UserMenuChoiceCommands.setLocalMenuChoiceHandler
        (userMenuChoice, handler);

    // Create the scroll-bars

    // Associate the scroll-bars with the view

    // Add all UI components to the Frame

    } catch (Exception e) {
        // Display an error dialog if something went wrong
        new com.gensym.dlg.MessageDialog (this,
            "Error!", false, e.getMessage (),
            (com.gensym.dlg.StandardDialogClient)null).setVisible (true);
    }
}
```

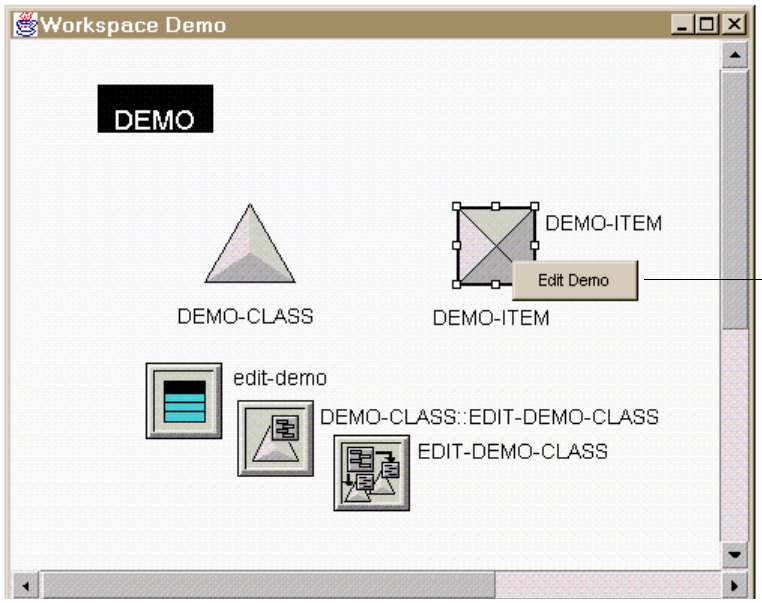
Here is the class definition for the `LocalMenuChoiceHandler`, which implements the `executeMenuChoice` method:

```
public class LocalMenuChoiceHandler implements MenuChoiceHandler {

    public void executeMenuChoice(Symbol menuLabel,
        Item itm, TwConnection cxn) throws G2AccessException {
        new MessageDialog (null, "Message", true,
            "Handling user menu choice locally", null).setVisible (true);
    }
}
```

To cause the factory to generate popup menus for selected items that contain user menu choices, you would implement the `createCommands` method of the `SelectionCommandInterface` as follows:

```
public SelectionCommand[] createCommands (ScalableWorkspaceView view) {
    return new SelectionCommand[] {
        new UserMenuChoiceCommands ()};
}
```

The resulting item popup menu for demo-item, an instance of demo-class, displays the Edit Demo menu choice:



Choosing Edit Demo in the client displays the following message dialog:

The menu choice does not get invoked in G2.

Part IV

Using Dialogs

14

Introduction to Telewindows2 Toolkit Dialogs

Presents the use of dialogs in Telewindows2 Toolkit, how to create them in the Component Editor, and discusses how to manage and launch them.

gensym

Introduction

Telewindows2 (TW2) Toolkit uses dialogs for displaying data and messages of any kind. Some dialogs are created automatically, using default controls, others are created automatically, using custom configurations, while others exist as dialog resources or classes that your application launches directly.

Various ways exist for you to create and manage dialogs. Using TW2 Toolkit dialog components you can:

- Create dialogs in a JavaBeans-compliant visual programming environment.
- Write Java code that uses dialog components within an application container.
- Customize the automatically generated dialogs for similar items.

The TW2 Toolkit dialogs described in this guide fall into two major categories:

- **Item properties dialogs —** Dialogs that you use to edit G2 item attributes.
- **General dialogs —** Informational dialogs for displaying text and input dialogs for obtaining data from a user.

While an item properties dialog is also an input dialog, it is treated separately in this document because of its special status in being used only for G2 item attributes.

This chapter describes both categories of dialogs, and how to create and manage them. Remaining chapters in this part present other specific topics:

This chapter...	Describes...
Chapter 15, "Using Dialog Components"	The Telewindows2 Toolkit components you use to create dialog resources.
Chapter 16, "Launching Custom Item Properties Dialogs"	How to save custom item property dialogs as part of your KB, and how to register these dialogs in G2 so they launch automatically.
Chapter 17, "Customizing Automatically Generated Dialogs"	How to customize the way in which a workspace view automatically generates dialogs for similar items.
Chapter 18, "Launching General Dialogs"	The programmatic requirements for launching general dialogs after you have created them.

Terminology

This part uses these terms:

- **Dialog —** A generic term for a set of one or more components that you display in a Java container.
- **Dialog resource —** A dialog that you create from components, which is stored as a serialized file.
- **Dialog class —** A dialog that you create from components in any Java programming environment.

This part uses the term dialog to refer to both dialog resources and dialog classes, unless a finer distinction is required.

Standard Dialogs

TW2 Toolkit also supplies a set of **standard dialogs**, which are Java classes that you use within Java applications or subclass to tailor to your specific needs.

Standard dialogs also include informational and input dialogs, but they are limited to use as TW2 Toolkit Java classes, as opposed to resources that you create using TW2 Toolkit components.

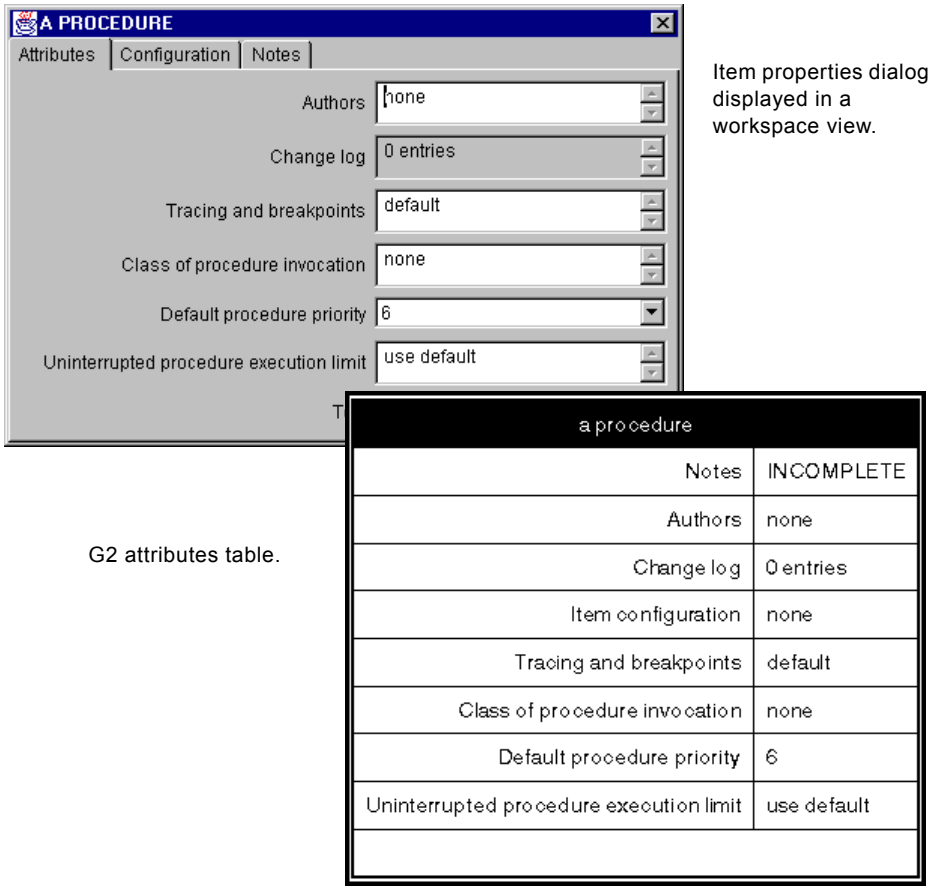
For information about using these dialog classes, see Chapter 4 "Using Standard Dialogs" in the *Telewindows2 Toolkit Java Developer's Guide: Application Classes*.

Item Properties Dialogs

As G2 uses attribute tables to display item attributes, Telewindows2 Toolkit uses item properties dialogs.

By default, when a user chooses Properties from an item's popup menu in a workspace view, TW2 Toolkit creates and displays a properties dialog for editing the item. The dialog includes all of the attributes visible in the current user mode of that TW2 Toolkit session.

This diagram illustrates the difference between an automatically generated item properties dialog that appears in a workspace view and its counterpart attribute table for the same item in G2:



Item properties dialog displayed in a workspace view.

G2 attributes table.

For information on how to use item properties dialogs in a workspace view, see "Using Workspace View Item Properties Dialogs" on page 141.

Automatically Generated Item Properties Dialogs

By default, workspace views automatically generate item properties dialogs when the user chooses Properties on an item. The workspace view gathers the properties and types necessary to create an automatically generated dialog through class introspection, then uses the `com.gensym.dlgruntime` package classes to manage and launch the dialog. The attributes that appear in the dialog depend on the attributes visible in the user's current user mode.

Each time a user displays a properties dialog, the workspace view calls G2 to obtain the current attribute values for the item being edited.

The automatically generated dialog maps attribute types to specific components based on the G2 attribute type, as follows:

G2 Type	Component
truth-value	com.gensym.jcontrols.G2Checkbox
symbol	com.gensym.jcontrols.G2TextField
text	com.gensym.jcontrols.G2TextField
integer	com.gensym.jcontrols.G2TextField
float	com.gensym.jcontrols.G2TextField
quantity	com.gensym.jcontrols.G2TextField
value	com.gensym.jcontrols.G2TextField
structure	com.gensym.gcg.G2TextArea
sequence	com.gensym.gcg.G2TextArea
item	com.gensym.gcg.G2TextArea

The item-configuration and notes attributes of an item appear in their own tab page. All other attributes appear on one tab page. The notes tab page is only visible when the item has notes. If the `autoUpload` property of the `ItemProxy` is `false`, then OK, Apply, and Cancel buttons appear at the bottom of the dialog; otherwise, no buttons appear and changes in the dialog are immediately sent to G2.

Note The `autoDownload` property of the `ItemProxy` determines whether the dialog is updated when the attribute's value changes in G2.

Once an automatically generated dialog exists for an item, the workspace view caches the dialog on the client for use during the current TW2 Toolkit session.

Like their attribute table counterparts in G2, generated item properties dialogs are updated automatically when any displayed value changes in the server. Changes to values displayed in an item properties dialog are similarly updated in the server.

Customizing Item Properties Dialogs

While generated dialogs exist automatically for your user interface, you might want to customize them for your application's requirements. You can create custom item property dialogs in one of two ways, depending on the requirements of your application and the development environment you wish to use:

To customize...	Do this...	Described in...
A small number of dialogs for specific classes or items, but not subclasses	Use the dialog components to create a dialog resource, or a dialog class, using any Java development environment, then register the dialog for a particular class or item.	Chapter 16, "Launching Custom Item Properties Dialogs" on page 305.
Numerous dialogs whose contents depends on the contents of the class or subclass	Customize the way in which a workspace view automatically generates dialogs	Chapter 17, "Customizing Automatically Generated Dialogs" on page 321.

Creating and Registering Custom Dialog Resources and Classes

To create custom item property dialogs as dialog resources or dialog classes, you build each dialog manually, in a JavaBeans-compliant visual programming environment or in pure Java.

Once the custom item properties dialog resource or class exists, you must register it with the `DialogManager` for launching whenever a user chooses the Properties item menu choice. You can register the dialog resource or class for a particular class or item. Registration occurs through an RPC call from G2 to this method on a `com.gensym.dlgruntime.DialogManager`:

```
setDialogResourceEntry
```

This method supports registration of dialog classes created in a Java programming environment.

Chapter 16, "Launching Custom Item Properties Dialogs" presents an example of registering custom item properties dialogs.

Customizing Automatically Generated Dialogs

If you wish to override dialogs for more than several classes or items, you should customize the way in which a workspace view automatically generates dialogs for items. You can customize any aspect of the automatically generated dialog you wish, such as:

- The "editor" or control that the dialog uses for editing a single attribute or a set of attributes.
- The label associated with each attribute editor.
- The order in which the attribute editors and labels appear in the dialog.
- The groups of attributes that the dialog displays, which you can generate based on any number of criteria, including its name, type, defining class, and whether it is system defined.

In addition, you can create your own automatically generated dialogs from scratch by assembling the above components and adding editors that launch these subdialogs:

- A native text editor for editing attributes with a grammar.
- A color dialog for editing color attributes.

Finally, you can generate the attribute panel and/or command button panel of the default automatically generated dialog, and place them inside your own type of container. If you define your own container for automatically generated dialogs, you must explicitly generated the item properties dialog.

General Dialogs

General dialogs consist of all dialogs other than those used to edit and display item properties and include:

- Informational dialogs
- Input dialogs

Numerous reasons exist for creating general dialogs in your application. Informational dialogs typically consist only of text. Warnings, messages, and prompts for users can all be classified as informational dialogs. Input dialogs are what your application displays to obtain various kinds of information from the user. For example, you could create a connection dialog for specifying a G2 process to which to connect, or a login dialog to log on to a secure G2.

General dialogs exist in two forms as a set of components:

- Saved in a serialized resource file and launched within a container, using the `DialogReader` and `DialogLauncher` classes of the `com.gensym.dlgruntime` package.
- Instantiated in a Java program at run time and shown in a Java container.

Using General Dialogs for Event Notification

One common reason for launching or displaying a general dialog is to advise a user that a particular event has occurred in the application. Such events cover numerous situations, including alarm signalling, status changes, or operator input requests. Regardless of the event purpose, however, two main methodologies exist for detecting and reacting to them:

- Event listening from a TW2 Toolkit client to predefined server events.
- Event propagation through RPC calls from the server to one or more clients, using G2 "whenever" rules.

Both techniques require KB design and intervention. For TW2 Toolkit client applications, we recommend using the first method in which your KB is designed to follow the Java publish and subscribe event model. TW2 Toolkit and G2 JavaLink together supply tools and sample KBs outlining how to accomplish this in your KB. For information about setting up your KB to handle events by using this methodology, see the *G2 JavaLink User's Guide*.

Once your KB is set up to publish events, the client registers itself as a listener for one or more KB events to receive necessary updates.

While the initial setup for events and event handling occurs on the G2 server side, listening is handled at the client side, and thus off-loads some KB processing.

Dialog Resources

You can create dialog resources in any Java-beans compliant visual programming environment, using the components that TW2 Toolkit supplies or using other components of your choice. You can create item properties dialogs and general dialogs as dialog resources.

To launch an item properties dialog, you simply need to register the dialog resource with the dialog manager, as described in "Creating and Registering Custom Dialog Resources and Classes" on page 208.

To launch a general dialog, you must launch the dialog yourself by using classes in the `com.gensym.dlgruntime` package.

For information about...	See...
G2 connectivity components	Chapter 3, “Using ItemRetriever” and Chapter 4, “Using TwConnector.”
TW2 Toolkit user interface components available for creating dialog resources	Chapter 15, “Using Dialog Components.”
Launching a general dialog after you have created it	Chapter 18, “Launching General Dialogs.”

15

Using Dialog Components

Presents the data-aware components that you can use to create dialogs.

Introduction

The Telewindows2 (TW2) Toolkit `sequoia.jar` file contains graphical user interface and connectivity components for creating dialogs for your client applications. Using these components, you can construct dialogs and dialog resources to perform such tasks as:

- Connecting to G2 and creating a login session.
- Displaying information about G2 items.
- Getting information from a user to update G2 items.
- Collecting data about multiple items to present to a user.

The two connectivity components are documented in:

- Chapter 3, "Using ItemRetriever."
- Chapter 4, "Using TwConnector."

The workspace view components are documented in Part III, "Viewing Workspaces" on page 125.

Note The examples shown in this chapter were created using the Telewindows2 Toolkit Component Editor, which is no longer supported. However, using the dialog components as Java beans in an IDE such as Symantec Visual Café would be similar.

Packages Covered

All of the components described in this chapter are defined in one of two packages:

- `com.gensym.controls` package, which contains `java.awt` versions of the controls for use in a JavaBeans-compliant visual programming environment, such as Symantec Visual Café or Borland J Builder.
- `com.gensym.jcontrols` package, which contains `javax.swing` versions of the controls.

com.gensym.controls

Interfaces

AttributeEditor

Classes

AttributeHolder
DialogCommand
FieldType
FieldTypeEditor
G2Button
G2Checkbox
G2DropDownChoice
G2Label
G2Listbox
G2Radiobox
G2TextField
ItemProxy
LimitMode
LimitModeEditor
StructureMUX
SymbolVector
SymbolVectorEditor

com.gensym.jcontrols

Classes

G2Button
G2Checkbox
G2ComboBox
G2Label
G2Listbox
G2Radiobox
G2TextField

BeanInfo Classes

BeanInfo classes extend java.beans.SimpleBeanInfo and are not documented here. In general, BeanInfo classes are used for limiting the properties, events, and methods that are visible in a JavaBeans-compliant visual programming environment.

com.gensym.dlgruntime

These classes apply to the `DialogCommand` control only.

Interfaces

`DialogCommandListener`

Classes

`DialogCommandEvent`

com.gensym.dlgevent

These classes are part of the G2 JavaLink.

Interfaces

`ObjectChangeListener`
`ObjectUpdateListener`

Classes

`ObjectChangeEvent`
`ObjectUpdateEvent`

Class Hierarchy of the Dialog Components

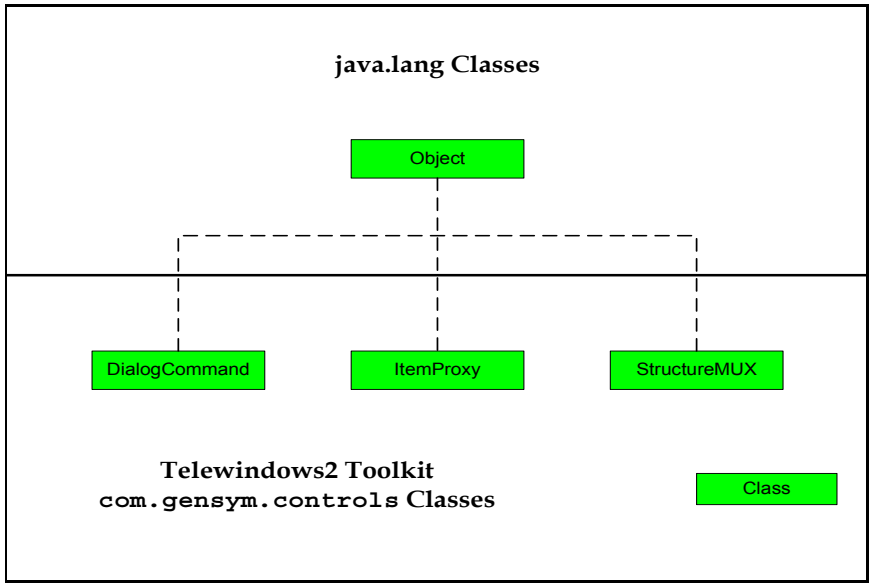
The dialog components in the `com.gensym.controls` package extend the `java.lang.Object` class or classes in the `java.awt` package. The dialog components in the `com.gensym.jcontrols` package extend classes in the `javax.swing` package.

Knowing the class hierarchy of these components provides relevant background information about available Java properties, events, and methods.

Helper Components

A number of the dialog components in the `com.gensym.controls` package are invisible beans that allow you to represent G2 data structures and control dialog events.

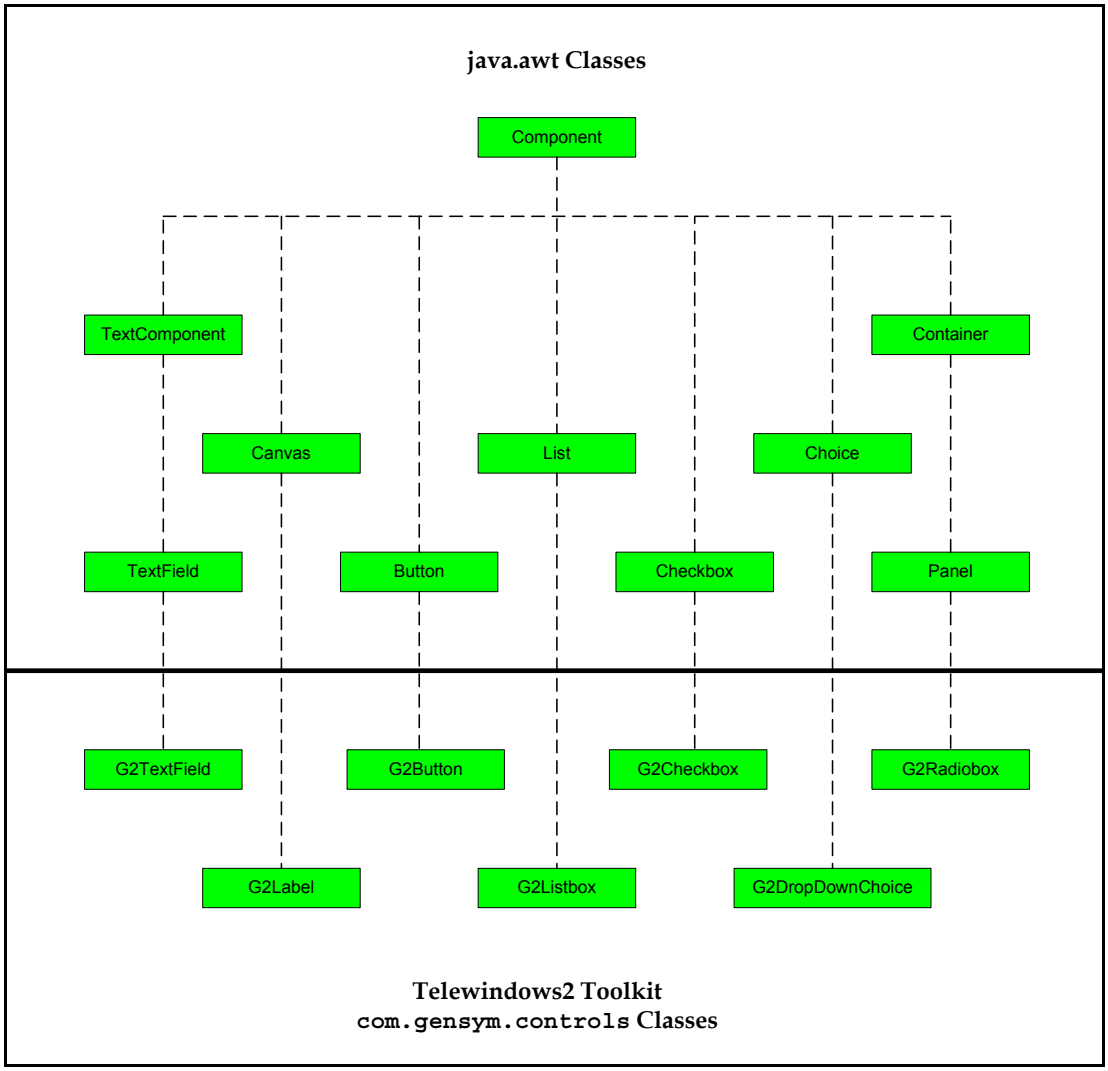
These helper components extend `java.lang.Object`:



Dialog Controls Based on AWT

The bulk of the dialog components in the `com.gensym.controls` package are visible beans that provide UI controls for editing the attributes if G2 items.

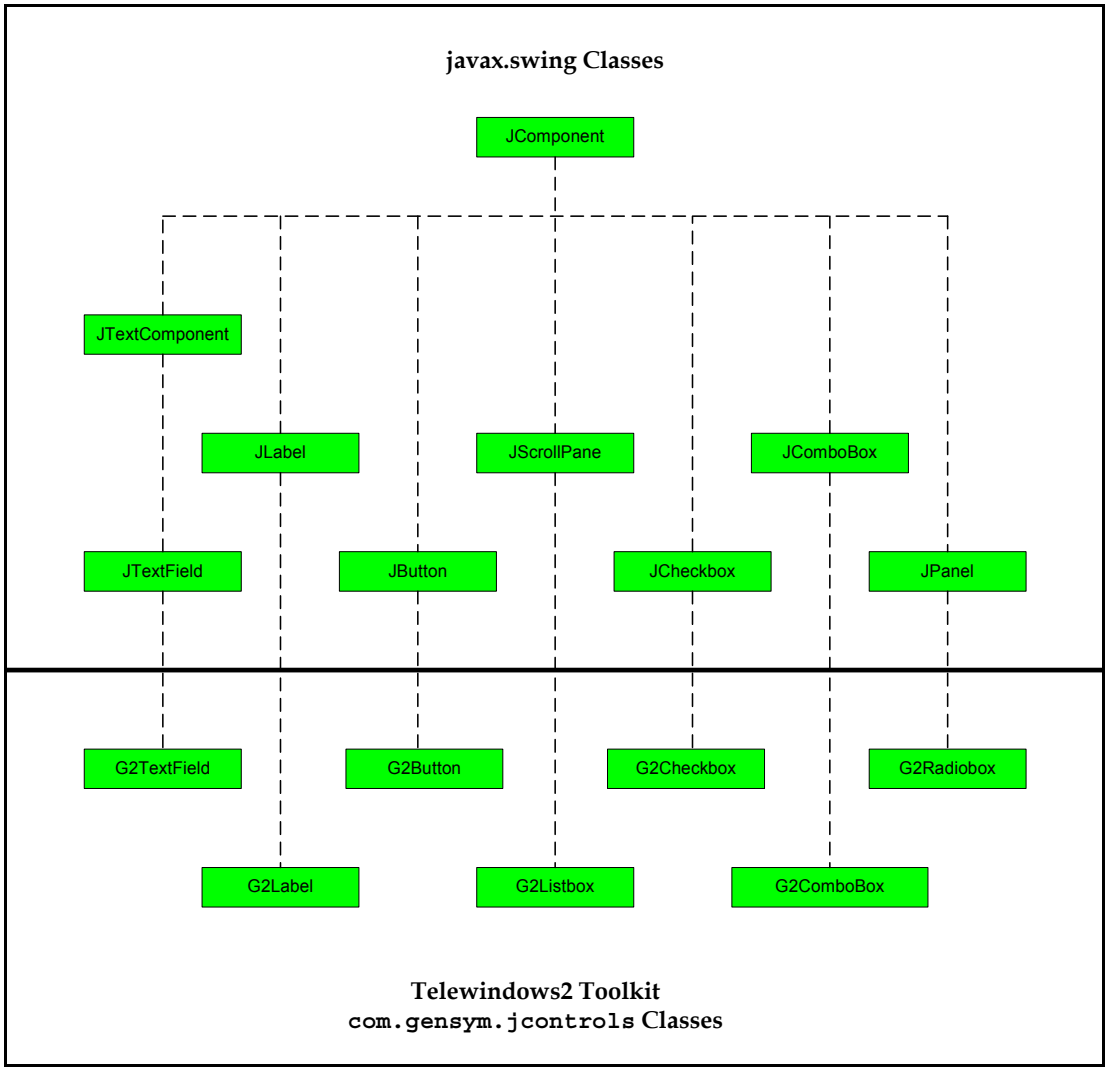
These dialog components extend classes in the `java.awt` package:



Dialog Controls Based on Swing

All the dialog components in the `com.gensym.jcontrols` package are visible beans that provide UI controls for editing the attributes if G2 items.

These dialog components extend classes in the `javax.swing` package:



Component Support Classes

The com.gensym.controls package provides a number of support classes for the visual components, which you see when you load the sequoia.jar file but which are not classes that you use explicitly.

AttributeEditor Interface

This class provides an interface that all TW2 Toolkit data-aware components implement, which causes each component to:

- Generate objectChanged events and notifies registered implementations of the com.gensym.beansruntime.ObjectChangeListener interface.
- Handle event notification for objectUpdated events.
- Get and set the G2 attribute whose value your component represents.

The following data-aware controls, in both the com.gensym.controls and com.gensym.jcontrols packages, implement the AttributeEditor interface:

- G2Checkbox
- G2DropDownChoice
- G2TextField
- G2Listbox

The other data-aware controls implement ObjectChangeListener and/or ObjectUpdateListener directly.

AttributeHolder Class

This class implements the common features of the ItemProxy and StructureMUX components.

FieldType and FieldTypeEditor Classes

All of the data-aware controls define the fieldType property, which is an instance of the FieldType class. This class translates string values that the user enters in a dialog into data types that G2 requires before the value gets sent to G2. It also converts G2 data types to strings before updating the component in a dialog.

The FieldType class supports these G2 data types:

- text
- symbol
- integer

- float
- quantity
- sequence
- structure
- truth-value

To allow editing of the `fieldType` property, the `com.gensym.controls` package provides the `FieldTypeEditor` class, which provides a drop down choice with options for each available data type.

LimitMode and LimitModeEditor Classes

The `G2TextField` component, in both the `com.gensym.controls` and `com.gensym.jcontrols` packages, provides properties called `upperLimit` and `lowerLimit`, which allow you to restrict the values a user enters when `fieldType` is one of the numeric data types. To determine whether the value the user can enter is inclusive or exclusive of the upper and lower limits, the component provides properties called `upperLimitMode` and `lowerLimitMode`, whose value is an instance of the `LimitMode` class.

To provide a way of editing these properties, the `com.gensym.controls` package provides the `LimitModeEditor` class, which provides a dropdown choice with options `Exclusive` and `Inclusive`.

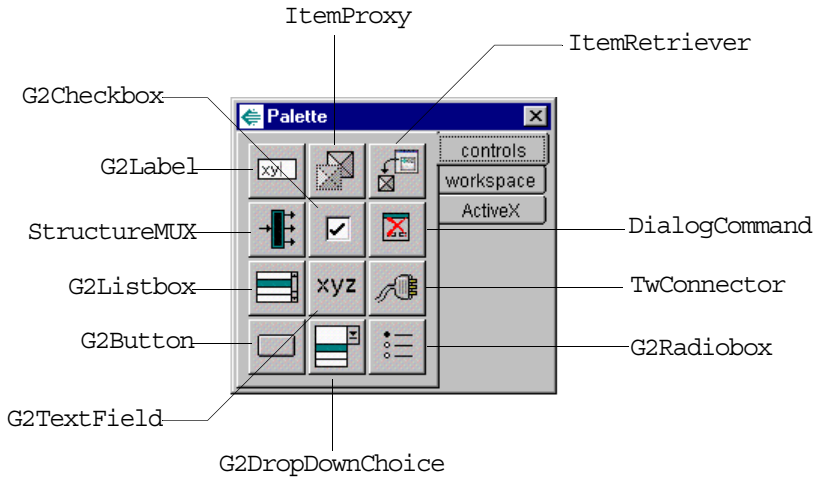
SymbolVector and SymbolVectorEditor Classes

Several data-aware controls provide a list of options from which the user can choose. You specify this list as a vector of symbols of type `SymbolVector`. For example, `com.gensym.controls.G2DropDownChoice`, `com.gensym.jcontrols.G2ComboBox`, and `G2Listbox` in both packages define the `choices` property, and the `G2Radiobox` in both packages defines the `members` property, both of which take a `SymbolVector`.

To allow editing of these properties, the `com.gensym.controls` package provides the `SymbolVectorEditor` class for entering the symbols.

Using Dialog Components

When you load the `sequoia.jar` file into a JavaBeans-compliant visual programming environment, a palette similar to the following appears:



Note This palette shows the `java.awt` version of the visual controls. If you load the `sequoia.jar` into JavaBeans-compliant visual programming environment, such as Symantec Visual Café or Borland J Builder, you would see the `javax.swing` versions of the visual controls, and `G2ComboBox` would replace `G2DropDownChoice` in the palette. Otherwise, the palettes are equivalent.

The dialog components are data aware, as described in "Data-Aware Components" on page 9.

They handle updates to and from G2 as described in "Change and Update Events" on page 10.

They all support the standard Java properties, events, and methods of their superior classes, as described in "Class Hierarchy of the Dialog Components" on page 216.

The following sections provide details on:

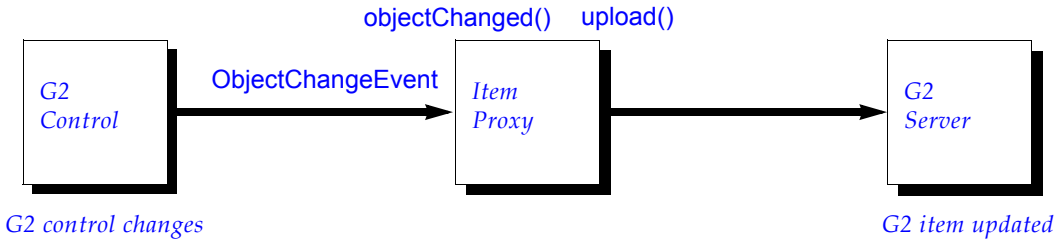
- How the dialog components handle updates.
- Using standard Java properties, events, and methods.
- Localizing dialog text.

The rest of this chapter provides a reference for each dialog component, including examples.

How G2 Gets Data Changes from a Control

This figure shows how a G2 control and an `ItemProxy` interact, when the `autoUpload` property of the `ItemProxy` is `true`, to upload a value from the control to G2, based on an `ObjectChangeEvent`:

ItemProxy Listening for ObjectChangeEvents



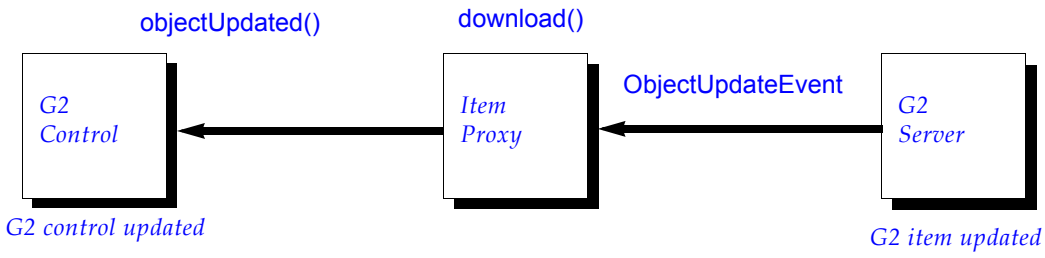
As this figure illustrates:

- When a user edits the value of a control, the control generates an `ObjectChangeEvent` and invokes the `objectChanged()` method in the `ItemProxy`.
- The `ItemProxy` automatically calls its `upload()` method, which sets the attribute value of the item in the G2 server to the new value.

How a Control Gets G2 Data Updates

This diagram shows how a G2 control and an `ItemProxy` interact, when the `autoDownload` property of the `ItemProxy` is `true`, to get a value from G2 and set the value in the control, based on an `ObjectUpdateEvent`:

G2 Control Listening for ObjectUpdateEvents



As this figure illustrates:

- When an update to an attribute value of a G2 item named by the `ItemProxy` occurs in G2, the `download()` method of the `ItemProxy` is automatically invoked, which gets the attribute value from the G2 server.
- When the `ItemProxy` calls its `download()` method, it generates an `ObjectUpdateEvent` and invokes the `objectUpdated()` method in the control.
- The `objectUpdated()` method sets the current value of the control to the new value from the `ItemProxy`.

Using Standard Java Properties

Because the TW2 Toolkit components inherit from the `java.lang.Object` class, `java.awt` classes, or `javax.swing` classes, certain properties are standard Java properties, which this guide does not document. The following table lists these properties and the Java class that defines them. Depending on the inheritance structure of the component, some or all of these properties are visible for the TW2 Toolkit components. For a picture of the inheritance structure of these controls, see "Dialog Controls Based on AWT" on page 218.

Property	Description	Class
`background`	The background color of the component. By default, the background is grey.	`Component` `JComponent`
`enabled`	Whether or not the component can respond to user input and generate events.	`Component` `JComponent`
`font`	The display font, style, and point size.	`Component` `JComponent`
`foreground`	The foreground color of the component, for example, the color of text in a `G2TextField`.	`Component` `JComponent`

`label`	The descriptive text of the component, if applicable.	`Button`
		`Checkbox`
		`JButton`
		`JCheckbox`

`name`	The name of the component.	`Component` `JComponent`

Localizing Dialog Component Text

All TW2 Toolkit components that have text provide standard Java properties and methods to support localization. These are the properties and corresponding accessor methods that support localization:

Property Get Method Set Method	Type	Description

`labelKey`
`getLabelKey`
`setLabelKey`
	`String`	The text key to add to a resource file for localizing dialog text.

`resourceName`
`getResourceName`
`setResourceName`
	`String`	The name of the resource containing the keys and accompanying text to use for localizing component text.

For more information about localization, see the API documentation for the JDK `java.util` package and the *Telewindows2 Toolkit Java Developer's Guide: Application Classes*.

Using Standard Java Events and Methods

Many of the methods visible for the data-aware components are inherited directly from the parent classes and, as such, are not documented here.

Because the `com.gensym.controls` components inherit from classes in the `java.awt` package, most of the events they generate are standard Java events. Similarly, most of the target methods for the components are standard Java methods.

This table lists the Java events and their associated classes that the data-aware controls support. For a picture of the inheritance structure of these controls, see "Dialog Controls Based on AWT" on page 218.

Event Type	Event	Class
`action`	`actionPerformed`	`Button` `JButton`

`component`	`componentResized`
	`componentMoved`
	`componentShown`
	`componentHidden`
		`Component`
		`JComponent`

Event Type	Event	Class
container	componentAdded componentRemoved	Container JContainer
focus	focusGained focusLost	Component JComponent
item	itemStateChanged	Component JComponent
key	keyTyped keyPressed keyReleased	Component JComponent
mouse	mouseClicked mousePressed mouseReleased mouseEntered mouseExited	Component JComponent
mouseMotion	mouseDragged mouseMoved	Component JComponent
text	textValueChanged	TextComponent JTextComponent

For information on these events, as well as the Java target methods for TW2 Toolkit components, refer to the Java API documentation.

Using G2 Item Components in Dialogs

You can create dialogs that include G2 items as Java Beans, which you create by using the G2 Bean Builder. You can create beans from system-defined or user-defined classes. You use these beans in a dialog in place of an `ItemProxy` and a `TwConnector`, or in place of an `ItemRetriever` to interact with a G2 item.

You use the G2 Bean Builder to generate a bean when you want to:

- Call user-defined methods of G2 items.
- Get and set attributes of G2 items, using third-party controls, which do not support `objectChanged` and `objectUpdated` events.

When you generate the bean, the G2 Bean Builder creates a JAR file for the bean, which you can load into any JavaBeans-compliant visual programming

environment.. The JAR file is located in the `jars` directory of your G2 JavaLink product directory.

If you are working in a pure Java programming environment, you can import into your application the Java classes that the G2 Bean Builder creates.

For information on creating beans from G2 classes, see the *G2 Bean Builder User's Guide*.

Identifying the Item

To identify the item with which the bean is associated, you specify the `sourceURL` property of the bean as follows, where `NAME` is the name of the G2 item, which must appear in upper case:

```
g2://<host>:<port>/<NAME>
```

For example:

```
g2://myhost:1234/MY-ITEM
```

Fetching the Item

To get and set attributes of the item, you must first fetch the G2 item. To do this, you have two options:

To...	Do this...
Call the bean's accessor methods to get and set user-defined attributes of the item, or to call user-defined methods of the item and pass in arguments of the appropriate type	Use an event hookup to call the `fetchG2Item` target method on the bean.
Set the user-defined attributes through the properties table for testing purposes	Set the `G2ItemFetched` property to `true` in the properties table for the bean.

When you fetch the G2 item of a user-defined bean, all the properties, events, and methods of the bean are visible.

Handling Events

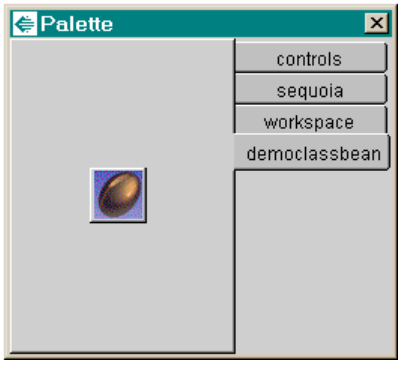
You handle event notification for a G2 bean in the same way you do with an `ItemProxy`:

- Use `objectChanged` events to notify the bean that the value of a data-aware control has changed.
- Use `objectUpdated` events to notify data-aware controls that the G2 item has been updated.

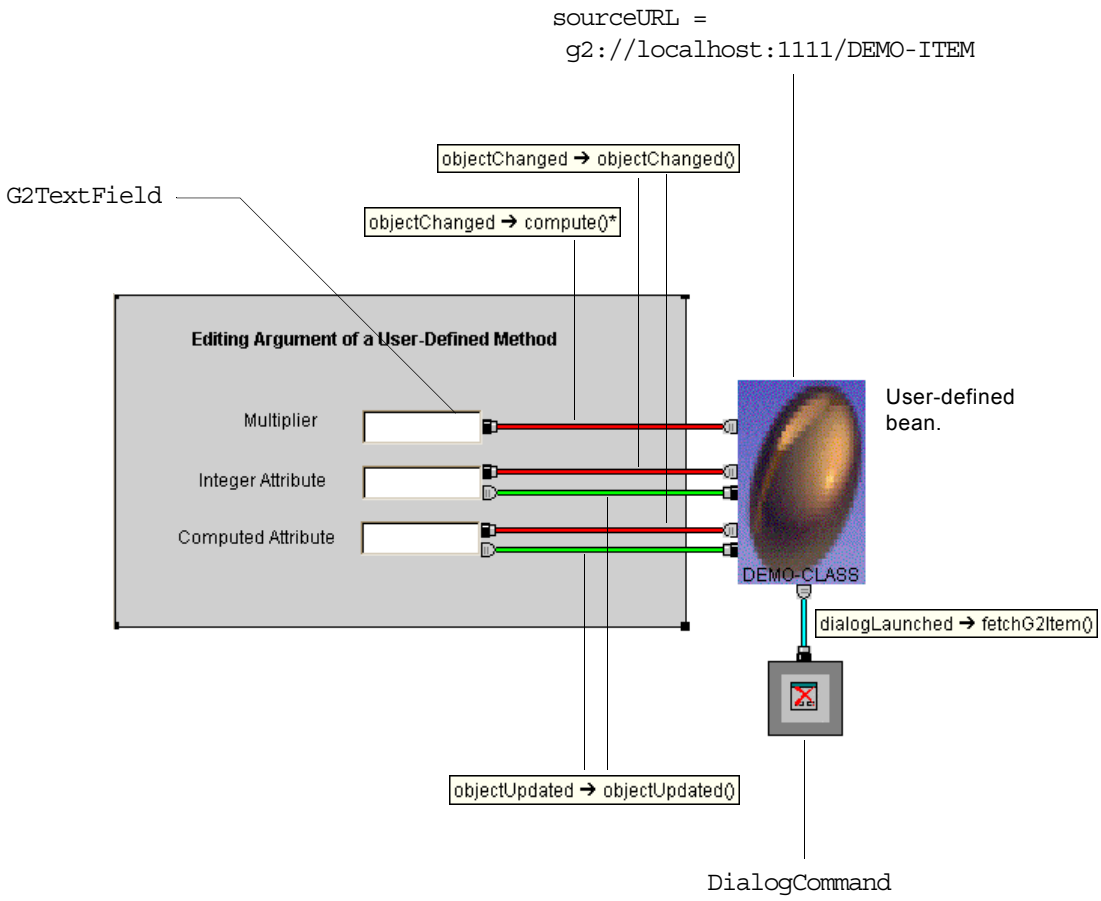
Example

The following example shows how to create a dialog that passes an argument to a user-defined method of a G2 item. The item has been generated as a bean, using the G2 Bean Builder, and the JAR has been loaded into a JavaBeans-compliant visual programming environment.

Here is the palette with the user-defined bean that appears when you load the user-defined JAR file:



This figure shows the layout, event hookups, and `sourceURL` property of the user-defined bean:



The dialog passes the value of the Multiplier field as the argument to the compute method of the demo-item. The method multiplies this argument by the value of the Integer Attribute field and concludes the product as the value of the Computed Attribute field.

The event hookups are as follows:

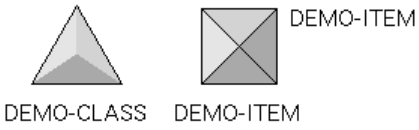
- The Multiplier `G2TextField` notifies the bean of `objectChanged` events and calls the user-defined `compute` method of the bean, using a custom hookup.
- The Integer Attribute `G2TextField` notifies the bean of changes and gets notified of updates in G2.
- The Computed Attribute `G2TextField` is read-only and just gets notified of updates in G2.
- The `DialogCommand` component calls the `fetchG2Item` method of the user-defined bean when the dialog gets launched.

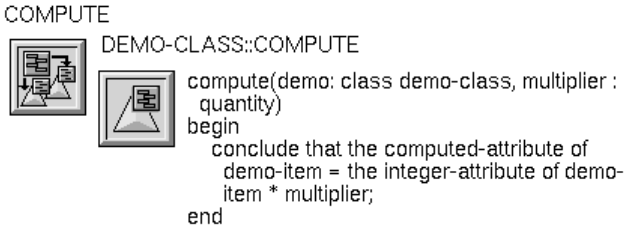
Tip To cause the bean's icon to scale to fit the specified rectangle, set the `ScaleImageToFit` property of the bean to `true` and set the `G2ItemFetched` property to `true`. Note that you will not see the G2 icon of the bean unless you make the bean visible on the dialog.

Here is the method in the custom hookup that you must edit, which initializes the parameter that gets passed to the user-defined `compute` method, where the circled code has been edited:

```
protected void deliverEvent(Object untypedArg0)  {
    com.gensym.dlgevent.ObjectChangeEvent arg0 = (com.gensym.dlgevent.ObjectChangeEvent) untypedArg0;
    /*
     * Cracking Arg #0 com.gensym.dlgevent.ObjectChangeEvent
     */
    java.lang.Object source= arg0.getSource();
    java.lang.Class clazz= arg0.getClass();
    java.lang.Object changedObject= arg0.getChangedObject();
    java.lang.String attributeName= arg0.getAttributeName();

    /*
     * Declarations for target method parameters -- Need initialization
     */
    java.lang.Number param0 = new Integer(((Integer)changedObject).intValue());
      try {
    ((com.gensym.classes.modules.demo.DemoClassBean)target).compute(param0);
      } catch (Exception e) {System.err.println (e);}
}
```

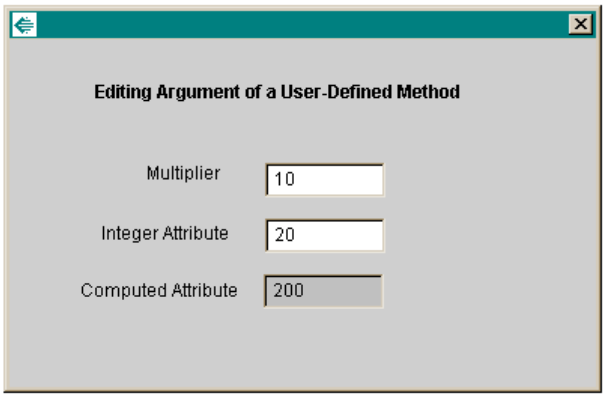
Here is the G2 class definition for the user-defined bean, the item, the method, and the method declaration:





When you launch the dialog and edit the Multiplier and Integer Attribute fields, the bean passes the argument to the compute method in G2, which updates the

computed-attribute of the item in G2, which in turn updates the Computed Attribute field in the dialog, as this figure shows:



Here are the corresponding attributes in the G2 table:

Integer attribute	20
Computed attribute	200

DialogCommand

Use the DialogCommand component to inform dialog listeners that a user has invoked the ok, apply, and/or close methods, typically by using a G2Button.

You use this component to batch uploads through an ItemProxy. By setting the autoUpload property of the ItemProxy to false, uploads occur when the ok or apply method of the DialogCommand is invoked.

This component also dispatches notification of the launching, shutdown, and flushing of changes of the dialog.

The DialogCommand is not a visual component, though it uses this representation:

com.gensym.controls.DialogCommand

Properties and Accessor Methods

These are the properties and associated accessor methods of a DialogCommand component:

Property Get Property Set Property	Type	Description
handleProxiesAutomatically getHandleProxiesAutomatically setHandleProxiesAutomatically	boolean	Determines whether the component flushes and shuts down all ItemProxy components on the dialog automatically. The default value is true, which: • Invokes the upload method of all ItemProxy components in the dialog when the DialogCommand component's ok or apply method is called. • Clears the stubs from all ItemProxy components when the DialogCommand component's ok or close method is called. If set to false, the component does not flush or shut down ItemProxy components.

For information on the additional properties that this component includes, see "Using Standard Java Properties" on page 224.

Events

These are the events that a `DialogCommand` component generates:

Event	Description
`dialogChangesFlushed`	Dispatches a `FLUSH DialogCommandEvent` to all registered listeners to commit their data when the `apply` method is called.
`dialogLaunched`	Dispatches a `LAUNCH DialogCommandEvent` to all registered listeners to indicate that the launch is complete when the `open` method is called.
`dialogShutdown`	Dispatches a `SHUTDOWN DialogCommandEvent` to all registered listeners to indicate that the dialog is no longer needed and no more events will occur when the `close` method is called.

For details, see these classes in the `com.gensym.dlgruntime` package:

- `DialogCommandListener`
- `DialogCommandEvent`

For information on the standard events that this component generates, see "Using Standard Java Events and Methods" on page 225.

Methods

These are the target methods of a `DialogCommand` component:

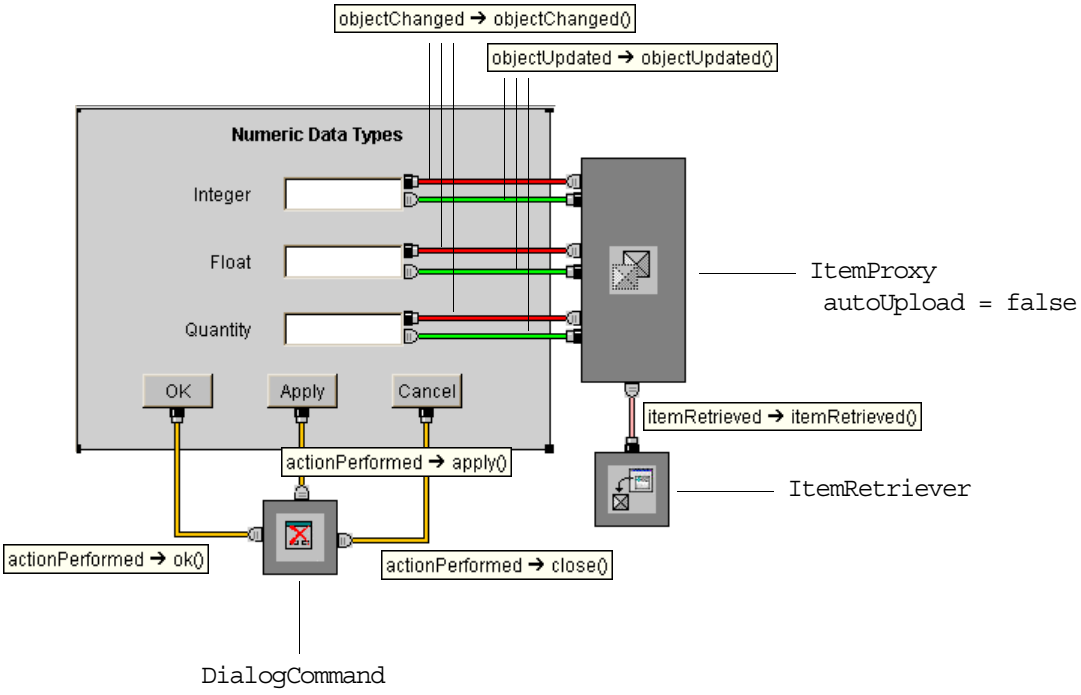
Method	Argument	Description
`apply`	N/A	Applies the changes as indicated on the dialog in all known `ItemProxy` components and leaves the dialog open. The `DialogCommand` generates a `FLUSH DialogStateEvent` to all listeners and handles `ItemProxy` uploads, if necessary.

Method	Argument	Description
close	N/A	Closes the dialog, generates a SHUTDOWN DialogStateEvent to notify all listeners, and sets the items in all known ItemProxy components to null.
ok	N/A	Applies the changes as indicated on the dialog in all known ItemProxy components and closes the dialog. The DialogCommand generates a FLUSH DialogStateEvent to all listeners to commit their data and a subsequent SHUTDOWN DialogStateEvent to indicate that the launch is complete.

For information on the standard methods that this component defines, see "Using Standard Java Events and Methods" on page 225.

Example

This example shows how to implement OK, Apply, and Cancel buttons on a dialog to batch uploads from an ItemProxy to G2. Notice that the autoUpload property of the ItemProxy is set to false, which causes the ItemProxy to hold value changes from the dialog components for uploading to G2 all at once, using the buttons.



G2Button

Use a G2Button component to capture a user clicking the button to perform some action. In addition to the basic capabilities provided by a java.awt.Button or javax.swing.Button, a G2Button adds support for localization of the button label.

The G2Button is a visual component with this icon:

com.gensym.controls.G2Button

Properties and Accessor Methods

These are the properties and associated accessor methods of a G2Button component:

Property Get Property Set Property	Type	Description
actionCommand getActionCommand setActionCommand	String	The name of the action command that this button fires, which is the button label, by default.

For information on the standard properties that this component includes, see:

- "Using Standard Java Properties" on page 224.
- "Localizing Dialog Component Text" on page 225.

Events

The G2Button generates the applicable events described in "Using Standard Java Events and Methods" on page 225.

Methods

Note These methods are available in Expert mode only.

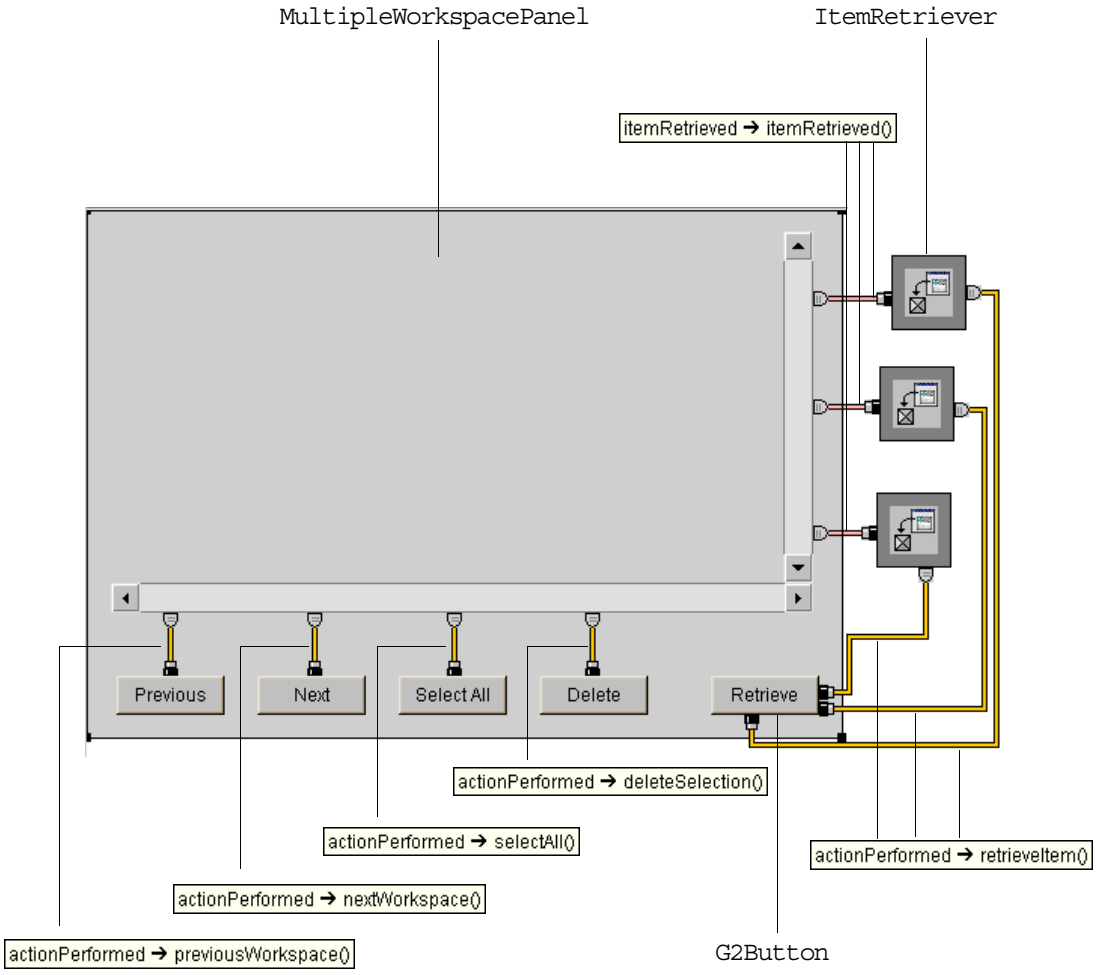
These are the target methods of a G2Button component:

Method	Argument	Description
get	Object	Gets the value of the Object argument to any of the put methods, which is a key. You invoke this method after invoking one of the put methods to retrieve state information from the button. You might do this, for example, if you want to use state information stored in the button as arguments to method calls that the button eventually triggers through its actionPerformed method. If you invoke the put method, the get method returns the value of the first Object argument.
getInt	Object, int	Gets the value of the int argument to the putInt or put method, which is the value associated with the key argument. You invoke this method after invoking the putInt method to retrieve state information from the button in the form of a key and an integer value.
getString	Object	Gets the value of the String argument to the putString and put methods, which is the value associated with the key argument. You invoke this method after invoking the putString method to retrieve state information from the button in the form of a key and a string value.
put	Object, Object	Allows you to store state information, in the form of a key and a value of any type, in the G2Button.
putInt	Object, int	Allows you to store state information, in the form of any key and an integer value, in the G2Button.
putString	Object, String	Allows you to store state information, in the form of any key and a string value, in the G2Button.

For information on the standard methods that this component defines, see "Using Standard Java Events and Methods" on page 225.

Example

This example shows how to create buttons to cycle between the previous and next KB workspaces. The dialog uses a `com.gensym.wksp.MultipleWorkspacePanel` that displays one of three KB workspaces from the `mill.kb` in the `kbs` directory of your TW2 Toolkit product directory for Java. The dialog also provides buttons to select all items in the workspace view, delete all selected items, and retrieve the named workspaces from G2.



For an example of how to use `G2Button` components to implement OK, Apply, and Cancel buttons on a dialog to batch uploads from an `ItemProxy` to G2, see "Example" on page 234 for `DialogCommand`.

G2Checkbox

Use a G2Checkbox component to turn an option on or off, or to set a G2 attribute whose data type is truth-value to true (checked) or false (not checked).

In addition to the basic capabilities provided by a java.awt.Checkbox or a javax.swing.Checkbox, a G2Checkbox adds support for localization of its label.

The G2Checkbox is a visual component with this icon:

com.gensym.controls.G2Checkbox

Properties and Accessor Methods

These are the properties and associated accessor methods of a G2Checkbox component:

Property Get Property Set Property	Type	Description
attribute getAttribute setAttribute	Symbol	The name of the G2 attribute associated with this component, as a com.gensym.util.Symbol. When initializing this property in code, call Symbol.intern(String string), where string is the attribute name as an upper-case string.
defaultContents getDefaultContents setDefaultContents	Boolean	The initial value for this component as a java.lang.Boolean, such as Boolean.TRUE.
state N/A setState	boolean	Whether the component is selected. The default is true, which displays a check mark in the checkbox. **Note**: The getState accessor method is defined in the parent class. **Note:** Overrides setState in java.awt.Checkbox.

For information on the additional properties that this component includes, see:

- "Using Standard Java Properties" on page 224.
- "Localizing Dialog Component Text" on page 225.

Events

These are the events that a `G2Checkbox` component generates:

Event	Description
`objectChanged`	Dispatched when the user checks or unchecks the `G2Checkbox`.

For details, see these classes in the G2 JavaLink `com.gensym.dlgevent` package:

- `ObjectChangeListener`
- `ObjectChangeEvent`

For information on the standard events that this component generates, see "Using Standard Java Events and Methods" on page 225.

Methods

These are the target methods of a `G2Checkbox` component:

Method	Argument	Description
`objectUpdated`	`ObjectUpdateEvent`	Responds to `ObjectUpdateEvents` by setting the state of this component to the current value of the attribute that this component represents.

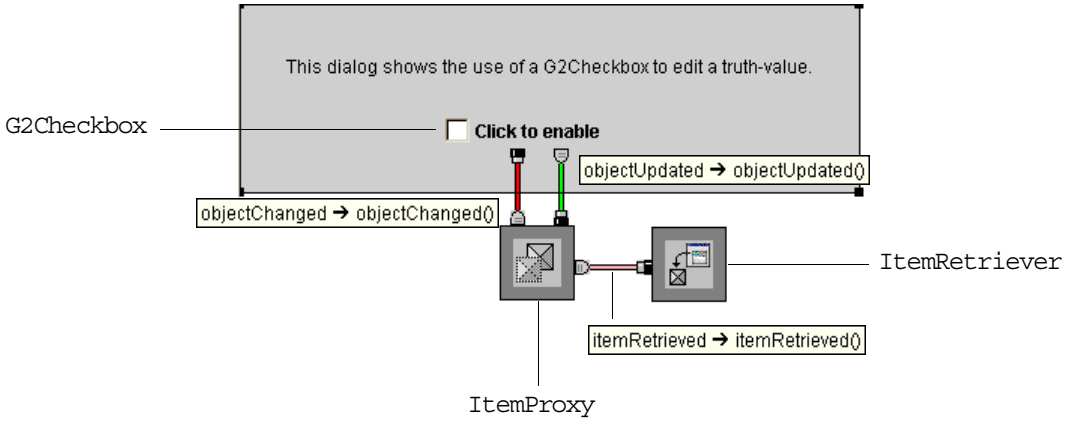
For details, see these classes in the G2 JavaLink `com.gensym.dlgevent` package:

- `ObjectUpdateListener`
- `ObjectUpdateEvent`

For information on the standard methods that this component defines, see "Using Standard Java Events and Methods" on page 225.

Example

This dialog allows the user to toggle a truth-value attribute in G2:



G2ComboBox

Use a G2ComboBox component to display a fixed-length list of options of the same type from which a user can choose one. It extends javax.swing.JComboBox and is located in the com.gensym.jcontrols package.

This component behaves the same as a com.gensym.controls.G2DropDownChoice. For details, see "G2DropDownChoice" on page 242.

G2DropDownChoice

Use a `G2DropDownChoice` component to display a fixed-length list of options of the same type from which a user can choose one. It extends `java.awt.Choice` and is only available in the `com.gensym.controls` package. For the Swing equivalent, use a `com.gensym.jcontrols.G2ComboBox`.

You can initialize the choices by:

- Specifying the `choices` property as a vector of strings that provide the default choices.
- Specifying the `initializationAttribute` property as an attribute of type sequence, if the `G2DropDownChoice` is editing an atomic data type.
- Specifying the `attribute` property as an attribute of type sequence.

The `G2DropDownChoice` is a visual component with this icon:

`com.gensym.controls.G2DropDownChoice`

Properties and Accessor Methods

These are the properties and associated accessor methods of a `G2DropDownChoice` component:

Property Get Property Set Property	Type	Description

`attribute`
`getAttribute`
`setAttribute`
	`Symbol`	The name of the G2 attribute associated with this component, as a `com.gensym.util.Symbol`.
		When initializing this property in code, call `Symbol.intern(String string)`, where `string` is the attribute name as an upper-case string.

`choices`
`getChoices`
`setChoices`
	`StringVector`	The choices that this component includes, as a `com.gensym.beansruntime.StringVector`.
		The value of the `fieldType` property must be able to parse the strings in the vector according to the specified G2 type.
		If the G2 attribute associated with this component defines an enumerated list of values, using the has values syntax, the choices must match some or all of the values defined in the attributes of the class.

`defaultContents`
`getDefaultContents`
`setDefaultContents`
	`String`	The default value of the component, as a `java.lang.String`, when it receives an `objectUpdated` event and the G2 attribute named by the `attribute` property does not provide an initial value.

Property Get Property Set Property	Type	Description

`fieldType`
`getFieldType`
`setFieldType`
	`FieldType`	The G2 type converter for the values that this component represents. See "FieldType and FieldTypeEditor Classes" on page 220.

`initializationAttribute`
`getInitializationAttribute`
`setInitializationAttribute`
	`Symbol`	The name of a G2 attribute of type sequence that provides the contents of the `choices` property, as a `com.gensym.util.Symbol`. This property is used by the `initializeChoices` method.

For information on the additional properties that this component includes, see "Using Standard Java Properties" on page 224.

Events

These are the events that a `G2DropDownChoice` component generates:

Event	Description
`objectChanged`	Dispatched when the user selects a new value from the `G2DropDownChoice`.

For details, see these classes in the G2 JavaLink `com.gensym.dlgevent` package:

- `ObjectChangeListener`
- `ObjectChangeEvent`

For information on the standard events that this component generates, see "Using Standard Java Events and Methods" on page 225.

Methods

These are the target methods of a `G2DropDownChoice` component:

Method	Argument	Description
`add`	String	Adds a `String` to the end of the dropdown choice. **Note:** Overrides `add` in `java.awt.Choice`.
`addItem`	`String`	Adds a `String` to the end of the dropdown choice. **Note:** Overrides `addItem` in `java.awt.Choice`.
`insert`	`String, int`	Inserts a `String` at a specified index, given as an `int`, into the dropdown choice. **Note:** Overrides `insert` in `java.awt.Choice`.
`initializeChoices`	`ObjectUpdateEvent`	Initializes the choices of this component by using the value of the G2 attribute named by the `initializationAttribute` property. You call this method by hooking up an `objectUpdated` event to the `G2DropDownChoice`.
`objectUpdated`	ObjectUpdateEvent	Responds to `ObjectUpdateEvents` by setting the value of this component to the current value of the attribute that this component represents.

Method	Argument	Description
`remove`	`String` `String, int`	Removes the first occurrence of `String` from the dropdown choice, or removes the item at the specified index, given as an `int`. **Note:** Overrides `remove` in `java.awt.Choice`.
`removeAll`	N/A	Removes all items from the dropdown choice. **Note:** Overrides `removeAll` in `java.awt.Choice`.

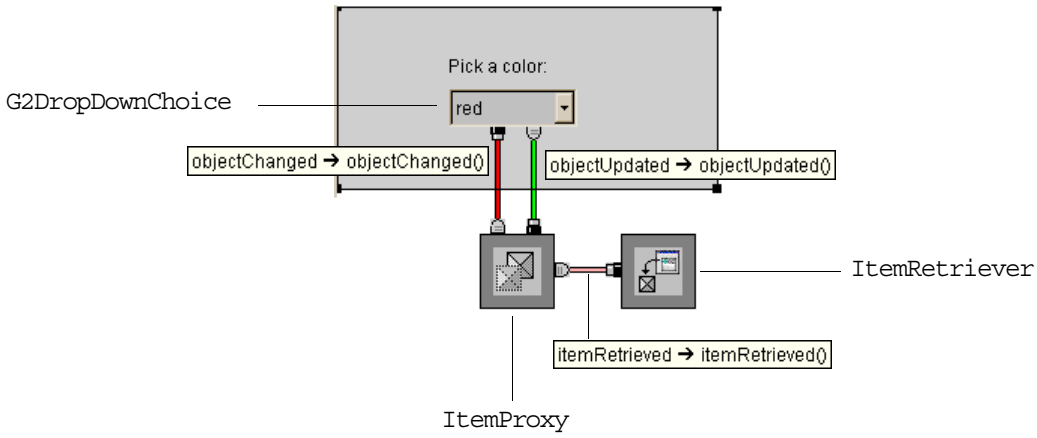
For details on the `objectUpdated` method, see these classes in the G2 JavaLink `com.gensym.dlgevent` package:

- `ObjectUpdateListener`
- `ObjectUpdateEvent`

For information on the standard methods that this component defines, see "Using Standard Java Events and Methods" on page 225.

Example

This example shows how to create a simple dialog that lets the user choose from one of several symbolic values to set the attribute of a G2 item:

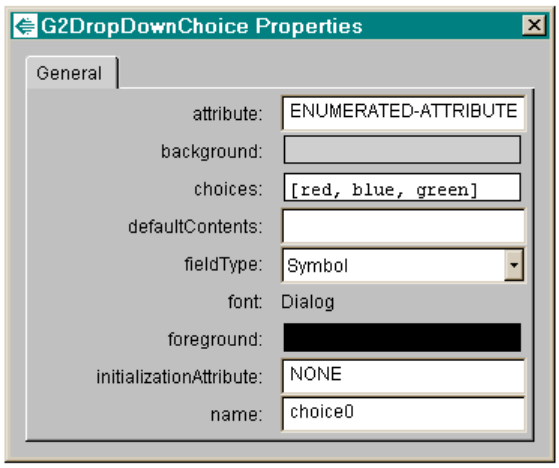


Here is the properties dialog for the `G2DropDownChoice`, which specifies:

- `attribute` as a G2 attribute that defines an enumerated list of symbols, for example:

  enumerated-attribute is a symbol, has values red, blue, or green, initially is red

- `choices` as a vector of strings that match the `fieldType`.
- `fieldType` as `Symbol`.



Here is the dialog when it is launched, with the choices visible:

G2Label

Use a G2Label component as a dialog title or in any location on your dialog where static text should appear.

In addition to the basic capabilities provided by a java.awt.Canvas or a javax.swing.JLabel, a G2Label adds support for localization.

The G2Label is a visual component with this icon:

com.gensym.controls.G2Label

Properties and Accessor Methods

These are the properties and associated accessor methods of a G2Label component:

Property Get Property Set Property	Type	Description
alignment getAlignment setAlignment	Enum	Determines whether the text is right, left, or center justified. The default value is right justified.
attribute getAttribute setAttribute	Symbol	The name of the G2 attribute associated with this component, as a com.gensym.util.Symbol. When initializing this property in code, call Symbol.intern(String string), where string is the attribute name as an upper-case string.
fieldType getFieldType setFieldType	FieldType	The G2 type converter for the values that this component represents. See "FieldType and FieldTypeEditor Classes" on page 220.

Property Get Property Set Property	Type	Description
minimumSize getMinimumSize N/A	Dimension	The minimum size of this component as a java.awt.Dimension. **Note:** Overrides minimumSize in java.awt.Component.
preferredSize getPreferredSize N/A	Dimension	The preferred size of this component as a java.awt.Dimension. **Note:** Overrides preferredSize in java.awt.Component.
showQuotesForTextType	boolean	Determines whether the label shows quotes when the fieldType property is Text. The default value is true. To hide quotes, set this property to false.

For information on the additional properties that this component includes, see:

- "Using Standard Java Properties" on page 224.
- "Localizing Dialog Component Text" on page 225.

Events

The G2Label generates the applicable events described in "Using Standard Java Events and Methods" on page 225.

Methods

These are the target methods of a G2Label component:

Method	Argument	Description
objectUpdated	ObjectUpdateEvent	Responds to ObjectUpdateEvents by setting the value of this component to the current value of the attribute that this component represents.
paint	N/A	Paints the Graphics context for this component. **Note:** Overrides paint in java.awt.Canvas.

For details on the objectUpdated method, see these classes in the G2 JavaLink com.gensym.dlgevent package:

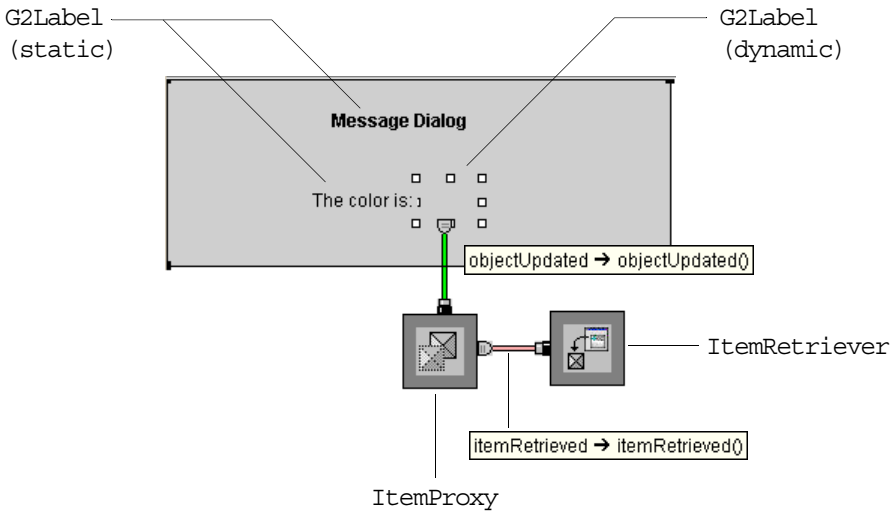
- ObjectUpdateListener
- ObjectUpdateEvent

For information on the standard methods that this component defines, see "Using Standard Java Events and Methods" on page 225.

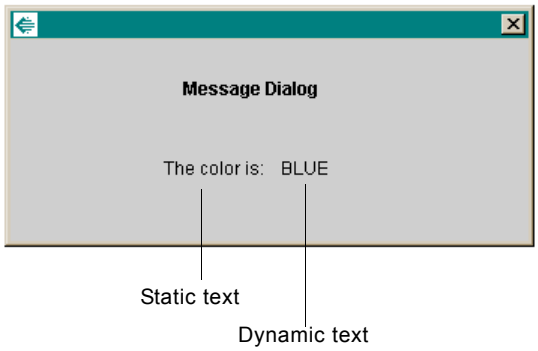
Example

This dialog shows how you create a message dialog that uses G2Label components for both static and dynamic text. The title and "The color is:" are static text, while the color text updates dynamically based on the value of an attribute of a G2 item. To do this, you specify the attribute property of the dynamically updating label and connect an objectUpdated event to the label to receive updates from the attribute in G2. The G2Label has no need to notify G2 of any objectChanged events.

Because the dynamically updating text is initially empty, the figure shows the handles on the `G2Label`.



Here is the message dialog that appears when you launch the dialog:



G2Listbox

Use the `G2Listbox` component for displaying multiple members in a single control. This component inherits from `java.awt.List` or `javax.swing.JScrollPane`.

You can use this component in one of two modes:

- Selection — A scalar control for editing atomic data structures such as numbers, symbols, strings, and truth values.
- Collection — An aggregate control for editing G2 sequence data structures.

You can initialize the choices by:

- Specifying the `choices` property as a vector of strings that provide the default choices.
- Specifying the `initializationAttribute` property as an attribute of type sequence, if the `G2Listbox` is editing an atomic data type, that is, if it is a selection list.
- Specifying the `attribute` property as an attribute of type sequence, if the `G2Listbox` is editing a sequence, that is, if it is a collection list.

The `G2Listbox` is a visual component with this icon:

`com.gensym.controls.G2Listbox`

Properties and Accessor Methods

These are the properties and associated accessor methods of a `G2Listbox` component:

Property Get Property Set Property	Type	Description

`attribute`
`getAttribute`
`setAttribute`
	`Symbol`	The name of the G2 attribute associated with this component, as a `com.gensym.util.Symbol`.
		When initializing this property in code, call `Symbol.intern(String string)`, where `string` is the attribute name as an upper-case string.

`choices`
`getChoices`
`setChoices`
	`StringVector`	The choices that this component includes, as a `com.gensym.beansruntime.StringVector`.
		The value of the `fieldType` property must be able to parse the strings in the vector according to the specified G2 type.
		If the G2 attribute associated with this component defines an enumerated list of values, using the has values syntax, the choices must match some or all of the values defined in the attributes of the class.

Property Get Property Set Property	Type	Description
defaultContents getDefaultContents setDefaultContents	String	The default value of the component, as a java.lang.String, when it receives an objectUpdated event and the G2 attribute named by the attribute property does not provide an initial value.
fieldType getFieldType setFieldType	FieldType	The G2 type converter for the values that this component represents. See com.gensym.controls.FieldType.
formatter	Formatter	The com.gensym.controls.FormatterEditor for this component, which formats the label. You do not need to specify this property.
initializationAttribute getInitializationAttribute setInitializationAttribute	String	The name of a G2 attribute of type sequence that provides the contents of the choices property, as a com.gensym.util.Symbol. This property is used by the initializeChoices method.

Property Get Property Set Property	Type	Description
listType getListType setListType	Enum	Determines the type of attribute the control represents and the behavior of the list. The options are: Selection — Represents scalar attribute values in G2 and allows the user to select a single element from the list. Collection — Represents a G2 sequence data type and allows the user to manipulate the size and elements of the list through a scalar control.
multipleMode	boolean	Determines whether the user can select more than one element in the list. The default value is false. This property is inherited from java.awt.List.
preferredSize getPreferredSize N/A	Dimension	The preferred size of this component as a java.awt.Dimension. **Note:** Overrides preferredSize in java.awt.List.

For information on the additional properties that this component includes, see "Using Standard Java Properties" on page 224.

Events

These are the events that a G2Listbox component generates:

Event	Description
objectChanged	In collection or selection mode, dispatched when the user chooses an item from the G2Listbox. You connect an objectChanged event from the G2Listbox to an ItemProxy.
objectUpdated	In collection mode, dispatched when the value of the attribute that this component represents gets updated in the G2 server. You connect an objectUpdated event from the G2Listbox to a scalar control, such as a G2TextField, to edit the collection list.

For details, see these classes in the G2 JavaLink com.gensym.dlgevent package:

- ObjectChangeListener and ObjectChangeEvent
- ObjectUpdateListener and ObjectUpdateEvent

For information on the standard events that this component generates, see "Using Standard Java Events and Methods" on page 225.

Methods

These are the target methods of a G2Listbox component:

Method	Argument	Description
add	String String int	Adds a String to the end of the list box, or adds a String at a specified index, given as an int. **Note:** Overrides add in java.awt.List.
addItem	String String int	Adds a String to the end of the list box, or adds a String at a specified index, given as an int. **Note:** Overrides addItem in java.awt.List.
extend	N/A	In collection mode, adds an element to the end of the collection list, which is the string G2, by default. To edit the element, connect a scalar control to the collection list via an objectChanged event.
getCurrentSelection	N/A	In collection mode, returns the currently selected list element. In selection mode, returns null.
initializeChoices	ObjectUpdatedEvent	Initializes the choices of this component by using the value of the G2 attribute named by the initializationAttribute property. You call this method by hooking up an objectUpdated event to the G2Listbox.

Method	Argument	Description
objectChanged	ObjectChangeEvent	In collection mode, responds to ObjectChangeEvents by setting the value of the currently selected element to the current value of a scalar control, which is the source of the objectChanged event.
objectUpdated	ObjectUpdatedEvent	In collection or selection mode, responds to ObjectUpdateEvents by setting the value of this component to the current value of the attribute that this component represents.
remove	String int	Removes the first occurrence of String from the list box, or removes the item at the specified index, given as an int. **Note:** Overrides remove in java.awt.List.
removeAll	N/A	Removes all items from the list box. **Note:** Overrides removeAll in java.awt.List.
replaceItem	String, int	Replaces the item at the specified index, given as an int, with a new String. **Note:** Overrides replaceItem in java.awt.List.

For details on the objectChanged and objectUpdated method, see these classes in the G2 JavaLink com.gensym.dlgevent package:

- ObjectUpdateListener and ObjectUpdateEvent
- ObjectChangeListener and ObjectChangeEvent

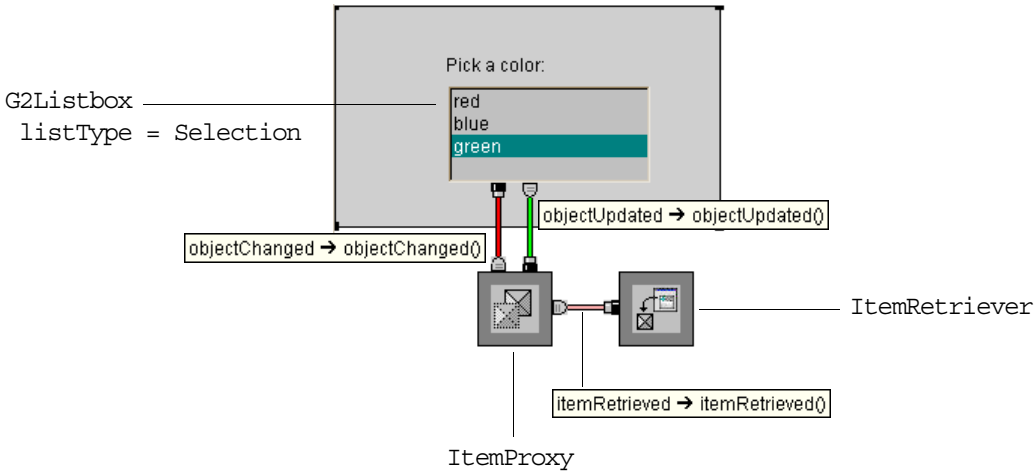
For information on the standard methods that this component defines, see "Using Standard Java Events and Methods" on page 225.

Examples

Using a G2Listbox in Selection Mode

A `G2Listbox` in selection mode is a scalar control for editing individual typed attributes of G2 items.

This example shows how to create a simple dialog that lets the user choose from one of several symbolic values to set an attribute of a G2 item:



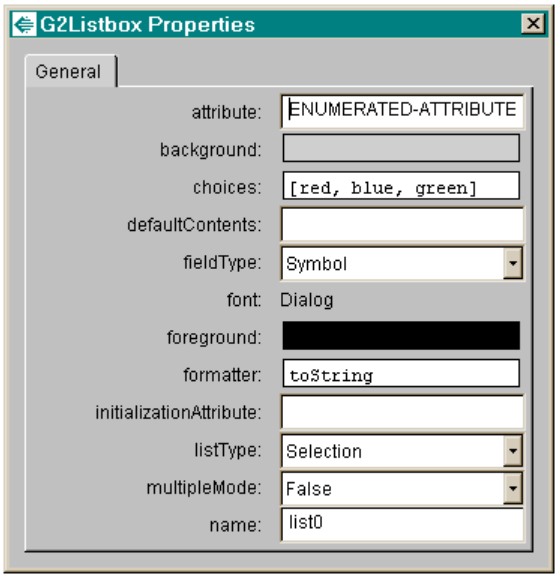
The properties dialog for the `G2Listbox` specifies these properties:

- `attribute` as a G2 attribute that specifies an enumerated list of symbols, for example:

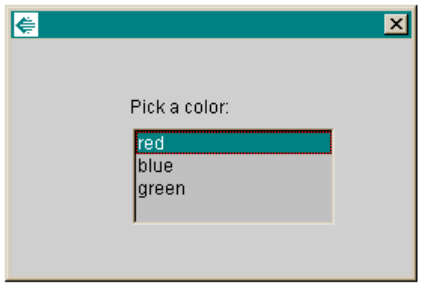
  enumerated-attribute is a symbol, has values red, blue, or green, initially is red

- `choices` as a vector of strings that match the `fieldType`.
- `fieldType` as `Symbol`.

Here is the properties dialog:



Here is the dialog when it is launched:



Using a G2Listbox in Collection Mode

A `G2Listbox` in collection mode is an aggregate control for editing an attribute of a G2 item that is defined as a G2 sequence. In collection mode, you use the `G2Listbox` in conjunction with a scalar control to edit individual elements of the collection list or the overall list itself.

Note You cannot currently use this control to edit the elements of a G2 list or array.

This figure shows how you use a `G2Listbox` with a `G2TextField` to edit a G2 attribute, whose value is a structure. This figure shows the class definition for the demo-class-1:

DEMO-CLASS-1

Class specific attributes flavors is a sequence, initially is sequence ();
animals is an instance of a text-array, initially
is an instance of a my-text-array

This figure shows the demo-item-1 and its table, which defines a sequence for the flavors attribute:

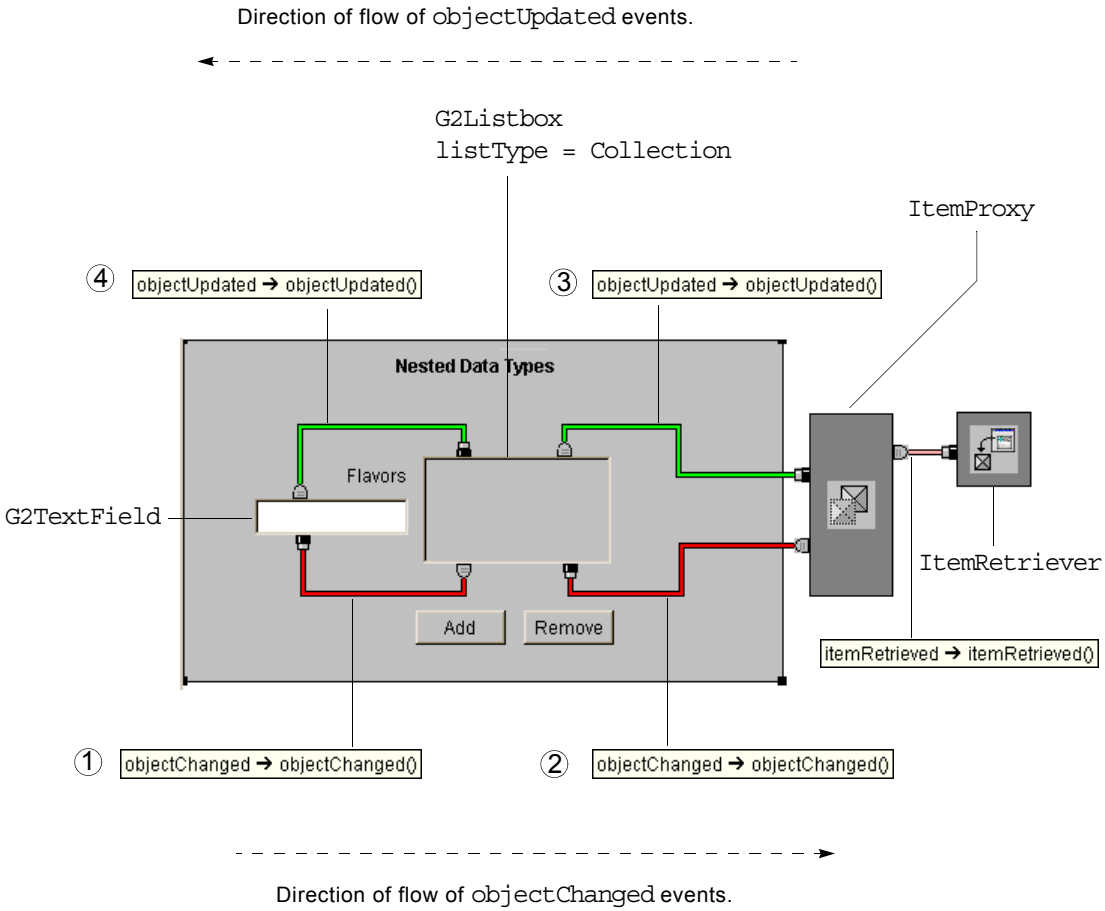
DEMO-ITEM-1

DEMO-ITEM-1, a demo-class-1	
Notes	OK
Item configuration	none
Names	DEMO-ITEM-1
Flavors	sequence (the symbol vanilla, the symbol chocolate, the symbol cherry)
Animals	a my-text-array

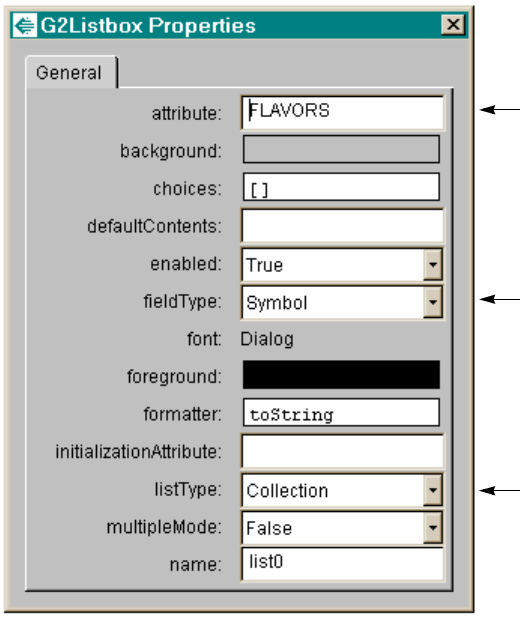
The following figure shows the dialog for editing the sequence with the `objectChanged` and `objectUpdated` event hookups showing. The `objectUpdated` events are located above the controls and flow from right to left, while the `objectChanged` events are located below the controls and flow from left to right.

The event hookups are labeled as follows:

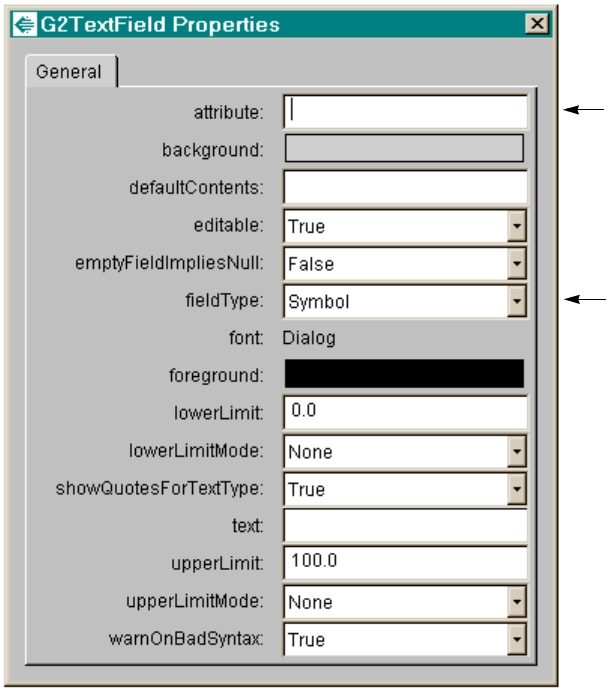
1. The `G2TextField` notifies the `G2Listbox` of `objectChanged` events.
2. The `G2Listbox` notifies the `ItemProxy` of `objectChanged` events.
3. The `ItemProxy` notifies the `G2Listbox` of `objectUpdated` events.
4. The `G2Listbox` notifies the `G2TextField` of `objectUpdated` events.


Direction of flow of objectUpdated events.
G2Listbox
listType = Collection
ItemProxy
④ objectUpdated → objectUpdated()
③ objectUpdated → objectUpdated()
Nested Data Types
Flavors
G2TextField
ItemRetriever
Add
Remove
itemRetrieved → itemRetrieved()
① objectChanged → objectChanged()
② objectChanged → objectChanged()
Direction of flow of objectChanged events.

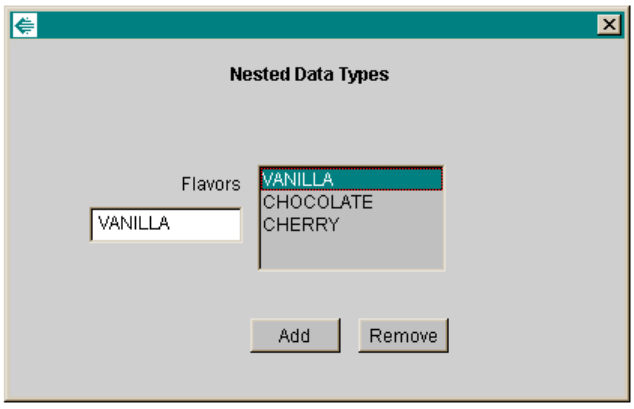
This figure shows the properties tables for the `G2Listbox` associated with the Flavors field. Notice that the `listType` of the `G2Listbox` is `Collection`:



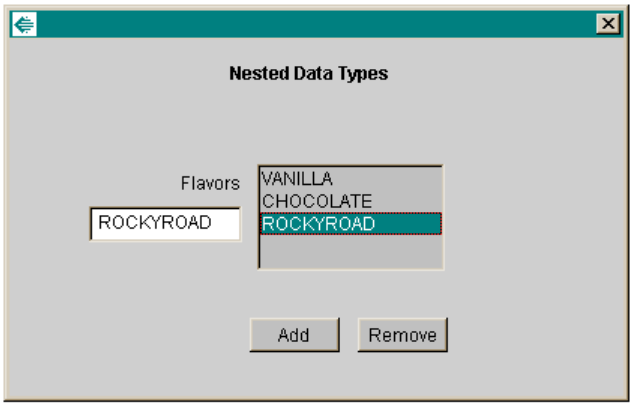
This figure shows the properties table for the `G2TextField` associated with the Flavors field. Notice that the `G2TextField` does not specify any value for the `attribute` property:



This figure shows the result of launching the dialog. The `G2Listbox` initializes with the contents of the sequence:



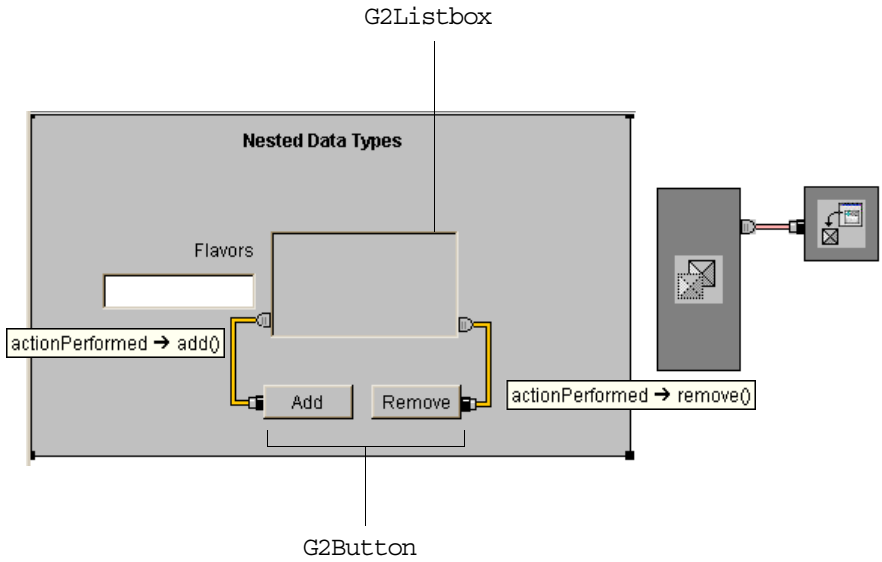
You can now edit any values of the lists by selected the list element to edit and entering a new value in the text field, for example:



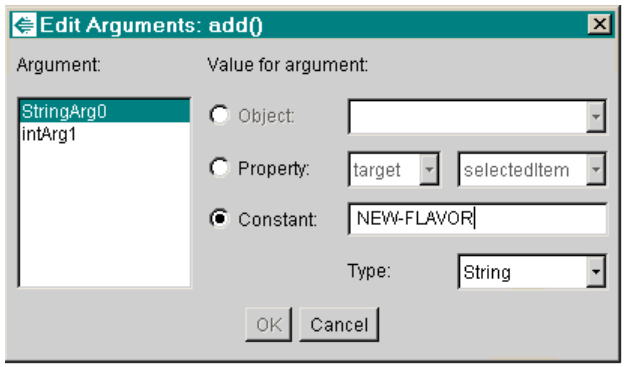
You use the Add and Remove buttons to add elements to the Flavors list. The event hookups for the Add and Remove `G2Button` controls use an `actionPerformed` event to invoke these methods in the `G2Listbox`:

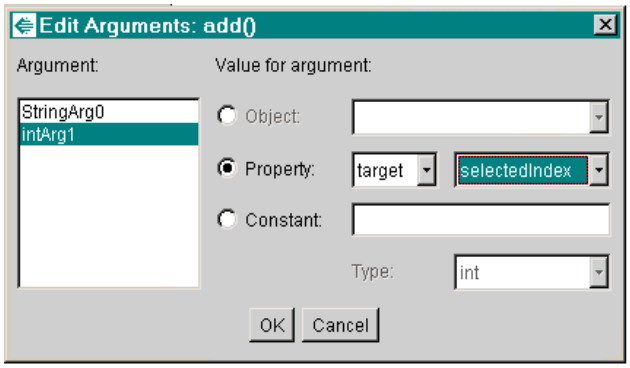
- `add(String int)` — Adds the specified `String` element to the collection at a particular index, which you specify as the `selectedIndex` property of the target object, which is the `G2Listbox`.
- `remove(int)` — Removes the list element at the `selectedIndex` of the target.

This figure shows the dialog with the `actionPerformed` events showing:

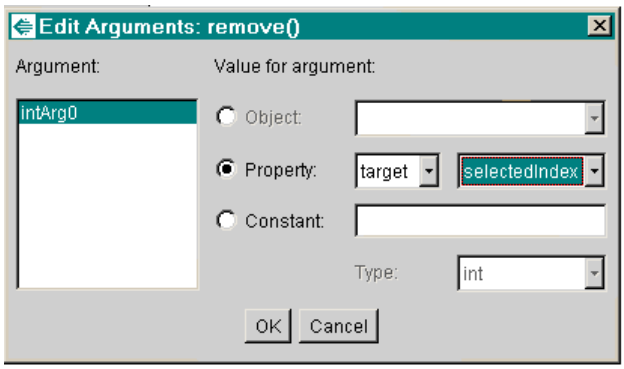


Here is the Edit Arguments dialog for each argument to the `add` target method:

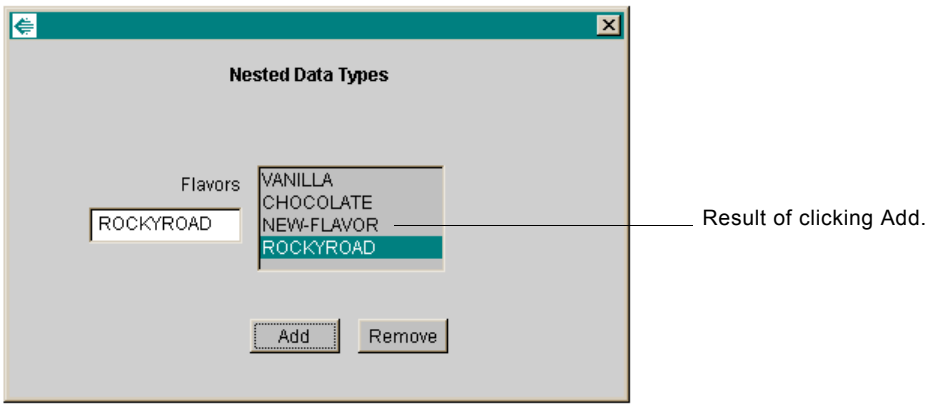




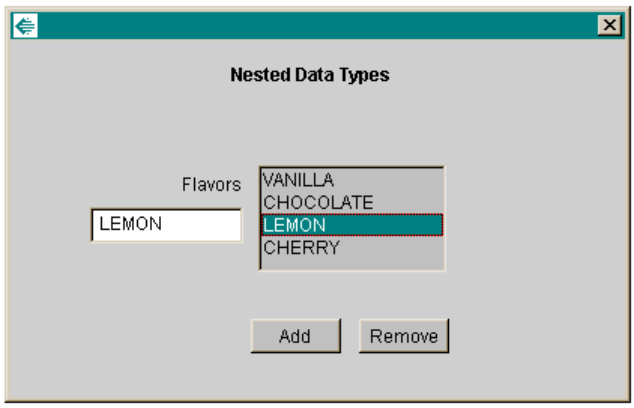
Here is the Edit Arguments dialog for the argument to the `remove` target method:



This figure shows the result of adding a new flavor to the list just above the last list element:



After editing the new flavor to be `LEMON`, the dialog and G2 sequence looks like this:



DEMO-ITEM-1, a demo-class-1	
Notes	OK
Item configuration	none
Names	DEMO-ITEM-1
Flavors	sequence (the symbol vanilla, the symbol chocolate, the symbol lemon, the symbol rockyroad)

Using a G2Listbox to Edit the Elements of a G2 List or Array

In addition to using a `G2Listbox` to edit the elements of a G2 sequence, you can use this component to edit the elements of a G2 list or array. To do this, you edit the following class attributes of the list or array that holds the elements, which are both G2 sequences:

G2 List	G2 Array
g2-list-sequence	g2-array-sequence

For additional class attributes that you can edit, see the *G2 Class Reference Manual*.

To edit these attributes of a list or array item directly, you would use the same event hookups that you use to edit the elements of a sequence, as described in the previous example.

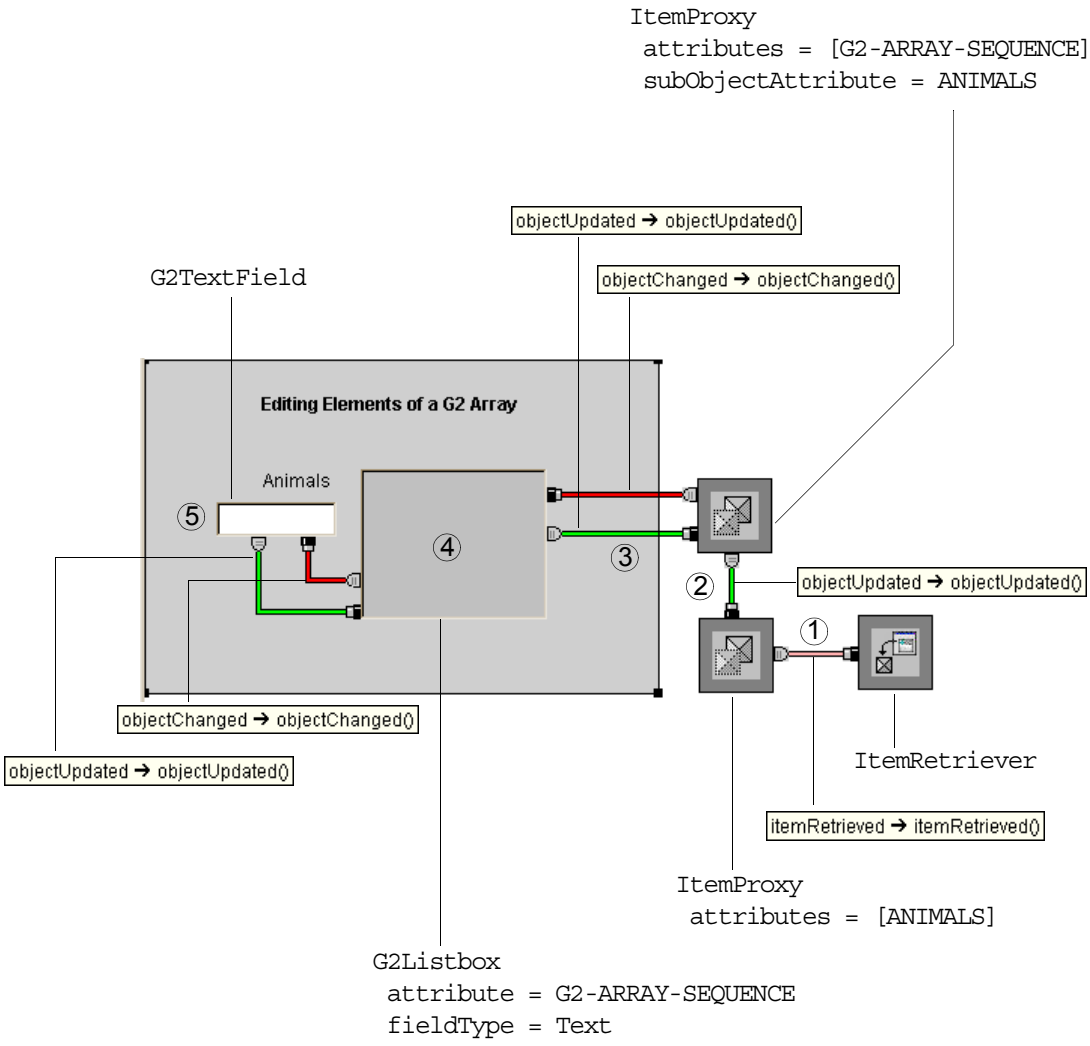
However, if your G2 list or array is a subobject of an attribute of a G2 item, as is typically the case, the dialog requires an additional `ItemProxy` and corresponding

event hookups to edit an attribute of a subobject. For details on the properties and event hookups required for editing subobjects of an `ItemProxy`, see "Editing Attributes of a Subobject" on page 295.

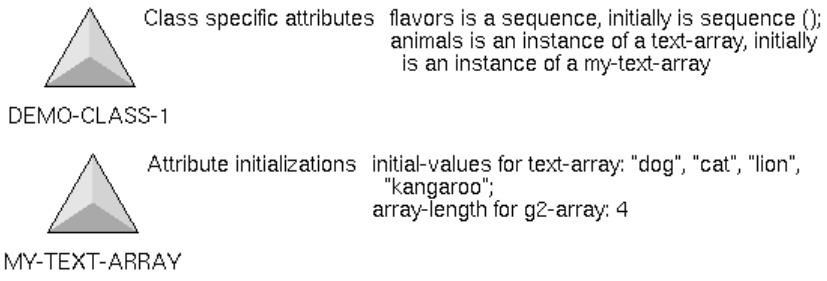
The following dialog performs these tasks, which are labeled:

1. The `ItemRetriever` retrieves the item whose attribute corresponds with `ANIMALS`, as the first `ItemProxy` specifies in its `attributes` property.
2. The first `ItemProxy` notifies the second `ItemProxy` when the value of the animals attribute changes in G2, through an `objectUpdated` event.
3. The second `ItemProxy` represents the animals subobject, and, thus, handles event notification to and from the `G2Listbox` when the value of the class attribute g2-array-sequence changes in the dialog or gets updated in G2.
4. The `G2Listbox` edits the g2-array-sequence attribute, which is a G2 sequence whose values are all text.
5. The `G2TextField` allows the user to edit the current element of the array by handling the appropriate events.

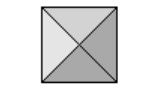
Here is the dialog for editing the elements of the animals subobject, which is a G2 text array:



Here are the class definitions for the demo-class and its subobject, which is a my-text-array:



Here is the demo-item-1 item, an instance of demo-class-1, with its table and the result of describing the initial value of the attributes attribute:



DEMO-ITEM-1

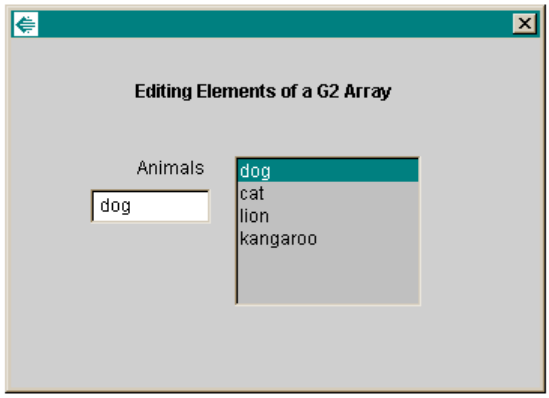
DEMO-ITEM-1, a demo-class-1	
Notes	OK
Item configuration	none
Names	DEMO-ITEM-1
Flavors	sequence ()
Animals	a my-text-array

Description of the animals of demo-item-1.

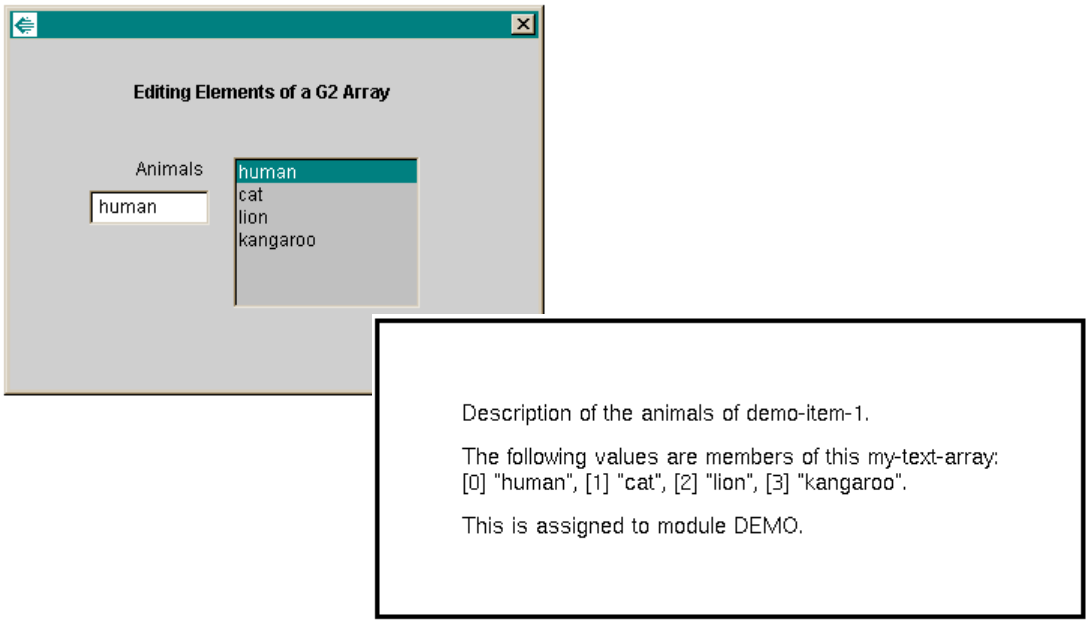
The following values are members of this my-text-array:
[0] "dog", [1] "cat", [2] "lion", [3] "kangaroo".

This is assigned to module DEMO.

Here is the dialog that appears when you launch:



This figure shows the effect of editing the elements of the array, both in the dialog and in G2:



G2Radiobox

Use a G2Radiobox component to make a single choice from a list of multiple options. You can also use this component to represent a boolean value. This component extends java.awt.Panel or javax.swing.JPanel.

The G2Radiobox is the visual component with this icon:

com.gensym.controls.Radiobox

Properties and Accessor Methods

These are the properties and associated accessor methods of a G2Radiobox component:

Property Get Property Set Property	Type	Description
attribute getAttribute setAttribute	Symbol	The name of the G2 attribute associated with this component, as a com.gensym.util.Symbol. When initializing this property in code, call Symbol.intern(String string), where string is the attribute name as an upper-case string.
columns getColumns setColumns	int	The number of columns that the G2Radiobox uses to align its members. The default value is 1.
defaultContents getDefaultContents setDefaultContents	String	The default value of the component, as a java.lang.String, when it receives an objectUpdated event and the G2 attribute named by the attribute property does not provide an initial value.
fieldType getFieldType setFieldType	FieldType	The G2 type converter for the values that this component represents. See "FieldType and FieldTypeEditor Classes" on page 220.

Property Get Property Set Property	Type	Description
insets getInsets setInsets	Insets	The size of the container's border, as a java.awt.Insets. **Note:** Overrides insets in java.awt.Container.
labels getLabels setLabels	StringVector	The label for each radio box, as a com.gensym.beansruntime.StringVector. If the label property is not specified, the value of the members property is uses for the labels.
members getMembers setMembers	StringVector	The members that this component includes, as a com.gensym.beansruntime.StringVector. The value of the fieldType property must be able to parse the strings in the vector according to the specified G2 type. If the G2 attribute associated with this component defines an enumerated list of values, using the has values syntax, the choices must match some or all of the values defined in the attributes of the class.
rows getRows setRows	int	The number of rows that the G2Radiobox uses to align its members. The default value is 3.

For information on the additional properties that this component includes, see "Using Standard Java Properties" on page 224.

Events

These are the events that a G2Radiobox component generates:

Event	Description
itemStateChanged	Dispatched when the state of this item changes.
objectChanged	Dispatched when the user clicks a G2Radiobox button.

For details on the objectChanged event, see these classes in the G2 JavaLink com.gensym.dlgevent package:

- ObjectChangeListener
- ObjectChangeEvent

For details on the itemStateChanged event, see these classes in the java.awt package:

- ItemListener
- ItemEvent

For information on the standard events that this component generates, see "Using Standard Java Events and Methods" on page 225.

Methods

These are the target methods of a G2Radiobox component:

Method	Argument	Description
objectUpdated	ObjectUpdateEvent	Responds to ObjectUpdateEvents by setting the value of this component to the current value of the attribute that this component represents.
setEnabled	boolean	Specifies whether this component can respond to user input and generate events. **Note:** Overrides setEnabled in java.awt.Component.
setState	String	Sets the selected radio button based on a String, which must match one of the strings in members.

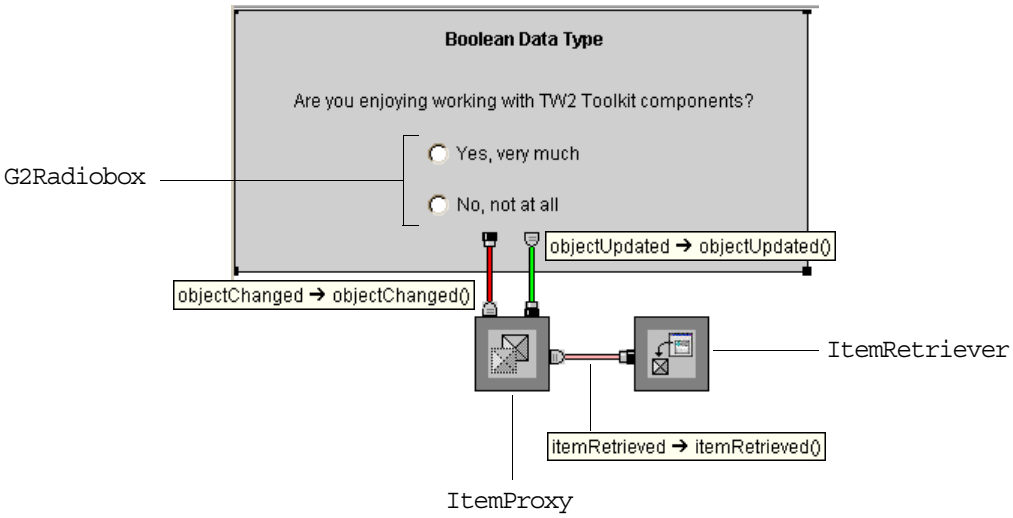
For details on the `objectUpdated` method, see these classes in the G2 JavaLink `com.gensym.dlgevent` package:

- `ObjectUpdateListener`
- `ObjectUpdateEvent`

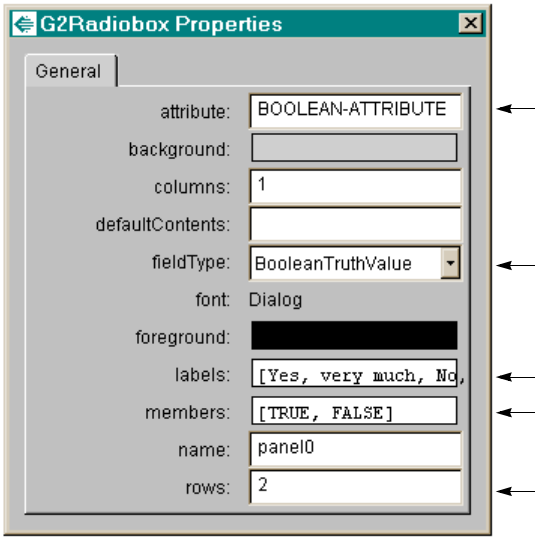
For information on the standard methods that this component defines, see "Using Standard Java Events and Methods" on page 225.

Example

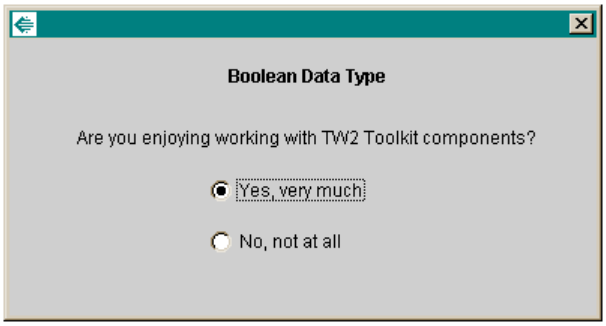
This example shows how to create a simple dialog that lets the user choose from one of two values to set an attribute of a G2 item whose value is a G2 truth-value:



Here is the properties dialog for the `G2Radiobox`, which specifies the `attribute`, `fieldType`, `labels`, `members`, and `rows` properties:



Here is the dialog that appears when you launch:



G2TextField

Use a `G2TextField` component to obtain data from a user. The field is type sensitive. If the value does not match the type you specify, the component can display a warning message. This component extends `java.awt.TextField` or `javax.swing.JTextField`.

The `G2TextField` is a visual component with this icon:

`com.gensym.controls.G2TextField`

Properties and Accessor Methods

These are the properties and associated accessor methods of a `G2TextField` component:

Property Get Property Set Property	Type	Description

`attribute`
`getAttribute`
`setAttribute`
	`Symbol`	The name of the G2 attribute associated with this component, as a `com.gensym.util.Symbol`.
		When initializing this property in code, call `Symbol.intern(String string)`, where `string` is the attribute name as an upper-case string.

`caretPosition`
`getCaretPosition`
`setCaretPosition`
	`int`	The current position of the cursor in the `G2TextField`, as an integer.
		Note: This property is inherited from `java.awt.TextComponent`.

`defaultContents`
`getDefaultContents`
`setDefaultContents`
	`String`	The default value of the component, as a `java.lang.String`, when it receives an `objectUpdated` event and the G2 attribute named by the `attribute` property does not provide an initial value.

Property Get Property Set Property	Type	Description
editable getEditable setEditable	boolean	Determines whether the user can edit the value of the G2TextField. The default value is true. Set to false to make the field read-only. **Note:** This property is inherited from java.awt.TextComponent.
emptyFieldImpliesNull getEmptyFieldImpliesNull setEmptyFieldImpliesNull	boolean	Determines how G2 interprets an empty G2TextField. The default value is false, which means the user must always provide a value for the G2TextField. Set this property to true to allow the user to leave the G2TextField blank, which G2 interprets as the symbol none. Thus, if the field is empty, any G2 expression that tests for the existence of the specified attribute returns false. You can only set this property to true for an untyped attribute; setting it to true for a typed attribute has no effect.
fieldType getFieldType setFieldType	FieldType	The G2 type converter for the values that this component represents. See "FieldType and FieldTypeEditor Classes" on page 220.
lowerLimit getLowerLimit setLowerLimit	double	For numeric field types, specifies the lower limit of a range of allowable values, when the lowerLimitMode property is Exclusive or Inclusive.

Property Get Property Set Property	Type	Description
lowerLimitMode getLowerLimitMode setLowerLimitMode	LimitMode	For numeric field types, determines whether the value of the G2TextField can include the lowerLimit property. The value is a com.gensym.controls.LimitMode, which can be Inclusive or Exclusive.
propagateEveryKeyTyped getPropagateEveryKeyTyped setPropagateEveryKeyTyped	boolean	Determines whether the component generates an objectChanged event for each character the user enters, or only for focusLost or actionPerformed events. The default is false.
selectionEnd getSelectionEnd setSelectionEnd	int	The cursor position of the end of the selected text, as an integer. **Note:** This property is inherited from java.awt.TextComponent.
selectionStart getSelectionStart setSelectionStart	int	The cursor position of the beginning of the selected text, as an integer. **Note:** This property is inherited from java.awt.TextComponent.
showQuotesForTextType	boolean	When fieldType is Text, determines whether the component requires the user to enter double quotes around the text. The default is true.
upperLimit	double	For numeric field types, specifies the upper limit of a range of allowable values, when the upperLimitMode property is Exclusive or Inclusive.

Property Get Property Set Property	Type	Description
upperLimitMode	LimitMode	For numeric field types, determines whether the value of the G2TextField can include the upperLimit property. The value is a com.gensym.controls.LimitMode, whose value can be Exclusive or Inclusive.
warnOnBadSyntax	boolean	Determines whether the component displays an error message if the data type of the value entered does not match the fieldType property of the component. The default is true, which means the dialog becomes modal if the user enters a bad value. These are some of the error messages a G2TextField, by default: • Incompatible type warning if the user does not enter a text value in quotation marks when showQuotesForTextType is true. • Incomplete text message if the user does not complete the field. To prevent these errors from displaying, set this property to false.

For information on the additional properties that this component includes, see "Using Standard Java Properties" on page 224.

Events

These are the events that a `G2TextField` component generates:

Event	Description
`objectChanged`	Dispatched when the text of the `G2TextField` changes.

For details on the `objectChanged` event, see these classes in the G2 JavaLink `com.gensym.dlgevent` package:

- `ObjectChangeListener`
- `ObjectChangeEvent`

For information on the standard events that this component generates, see "Using Standard Java Events and Methods" on page 225.

Methods

These are the target methods of a `G2TextField` component:

Method	Argument	Description
`objectUpdated`	`ObjectUpdatedEvent`	Responds to `ObjectUpdateEvents` by setting the value of this component to the current value of the attribute that this component represents.

For details on the `objectUpdated` method, see these classes in the G2 JavaLink `com.gensym.dlgevent` package:

- `ObjectUpdateListener`
- `ObjectUpdateEvent`

For information on the standard methods that this component defines, see "Using Standard Java Events and Methods" on page 225.

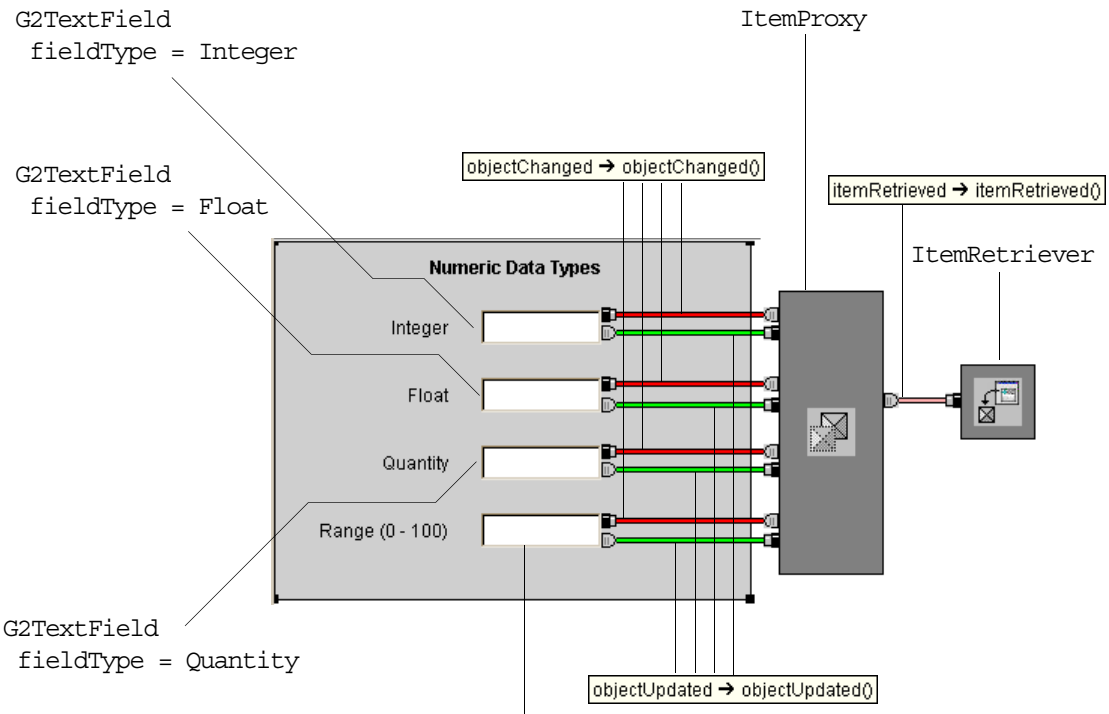
This component defines this `protected` method, which subclasses can call:

- `fireObjectChangeOnContents`

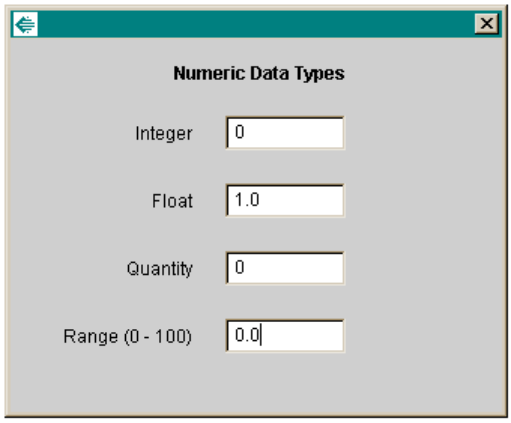
Examples

Editing Numeric Data Types

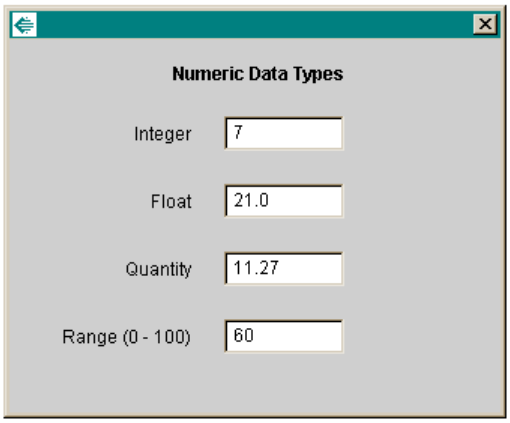
This example shows how to use a `G2TextField` to edit numeric data types, including a G2 integer, float, and quantity. The Range field allows the user to enter a number between 0.0 and 100, excluding 100.



Here is the dialog that appears when you launch, which gets the default values of each `G2TextField` from the corresponding G2 attribute:

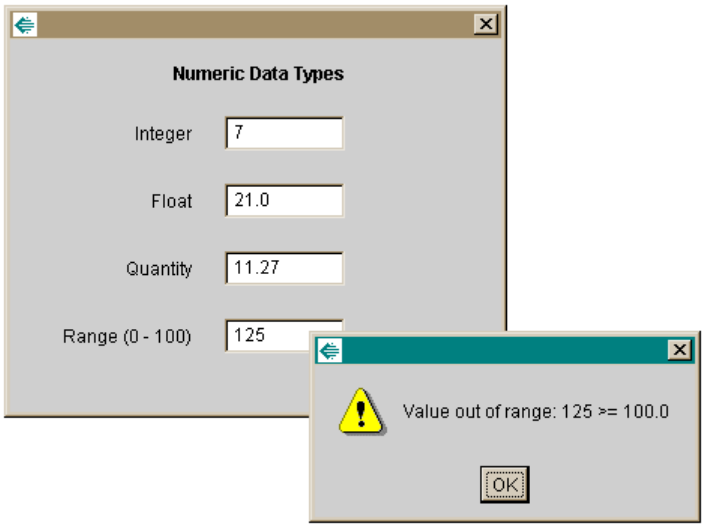


Here is the dialog that appears when you edit values in the text fields and the corresponding values in the G2 attribute table. Note that, by default, the value only gets updated in G2 when the user presses the Return key or moves the cursor to another field.



Integer attribute	7
Float attribute	21.0
Quantity attribute	11.27
Range attribute	60

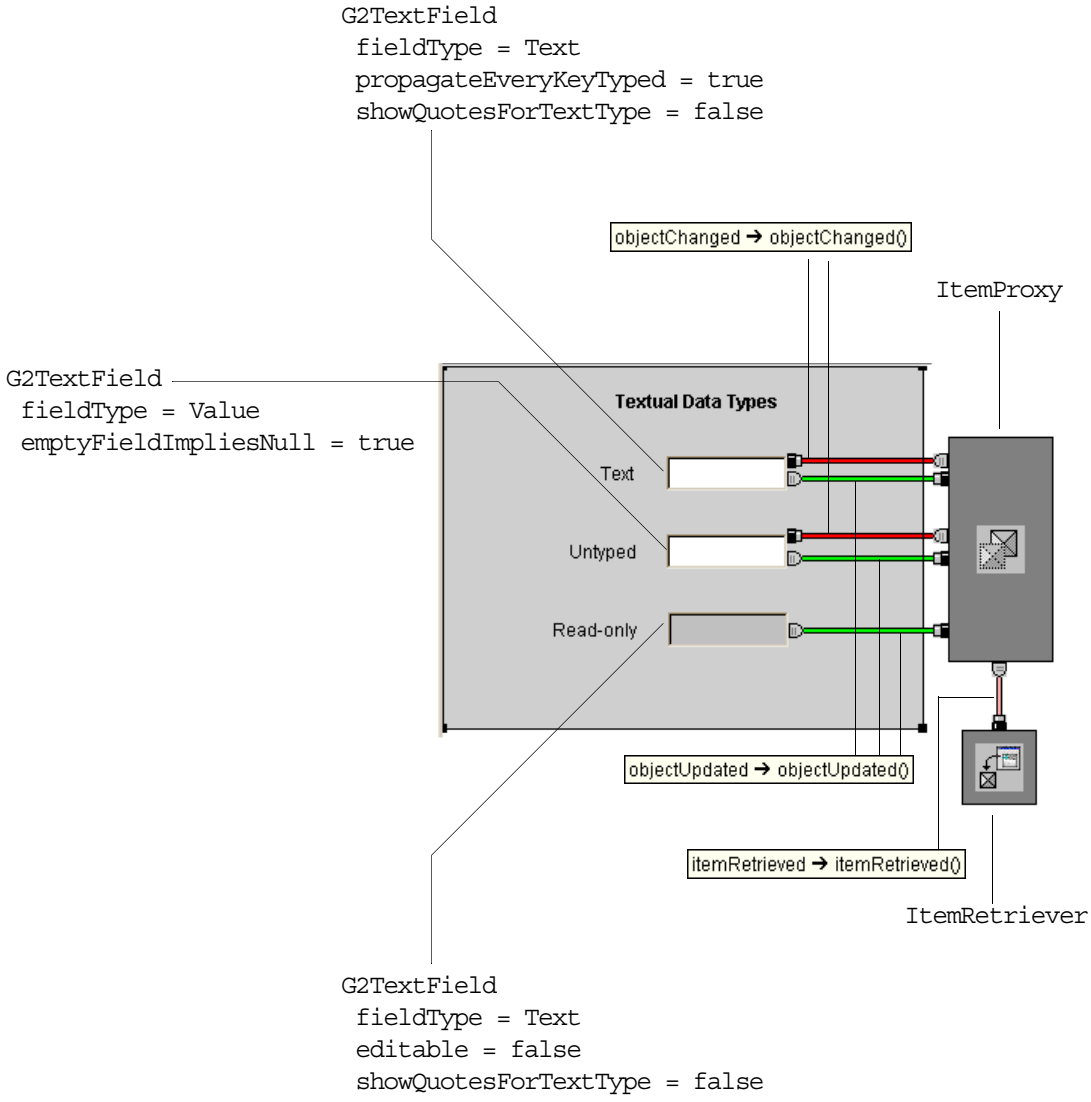
If the user attempts to enter a value for any text field that is not of the correct type or within the specified range, the `G2TextField` displays an error dialog, as this figure shows:



Editing Textual Data Types and Untyped Attributes

This example shows how to use a `G2TextField` to edit textual and untyped attributes. The Text field represents an attribute that takes a G2 text data type, while the Untyped field represents an attribute with no type specification at all.

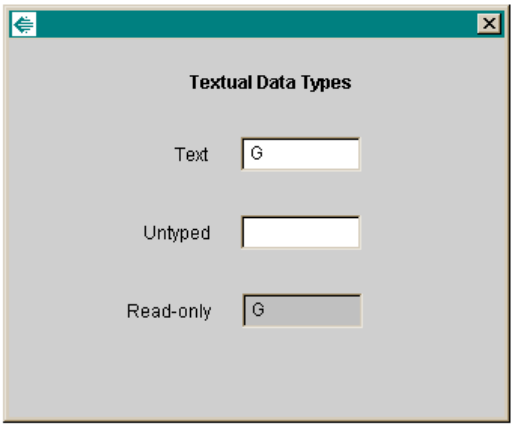
The Text field hide quotes and propagates every key as it is typed. The Untyped field allows the user to leave the field blank to imply a value of none.



This figure shows two stages of editing the Text field in the dialog that is launched and the corresponding attributes in the G2 table. The Text field notifies G2 of changes each time the user enters a character in the field. The value of the Text field is a G2 string that includes double quotes, although the user does not need to enter them in the dialog. The value of the Untyped field is the symbol none, indicating that the blank field implies that the attribute "does not exist"

according to the G2 compiler. The Read-only field gets updated each time the Text field changes.

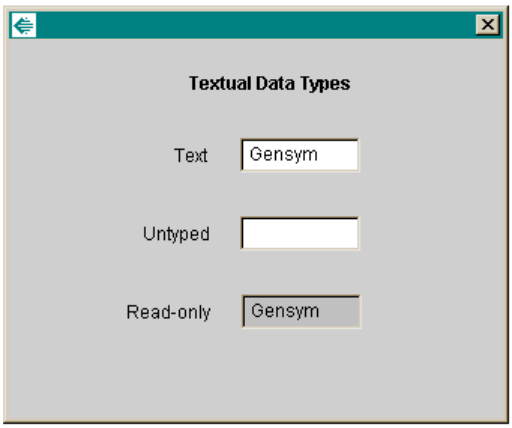
①



Text attribute	"G"

Untyped attribute	none

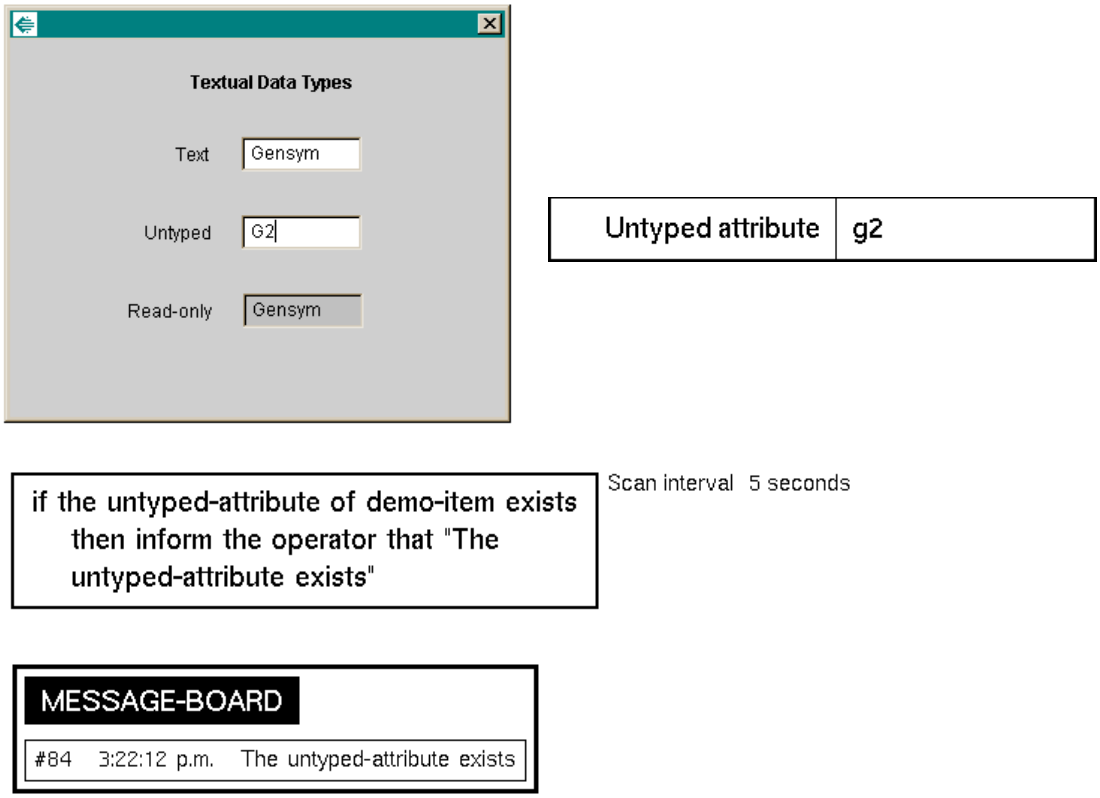
②



Text attribute	"Gensym"

Untyped attribute	none

This figure shows the result of entering a value in the Untyped field and enabling the following G2 rule, which tests for the existence of untyped-attribute:



When the Untyped field is left blank, no message appears.

Passing G2TextField Values as Arguments to Methods

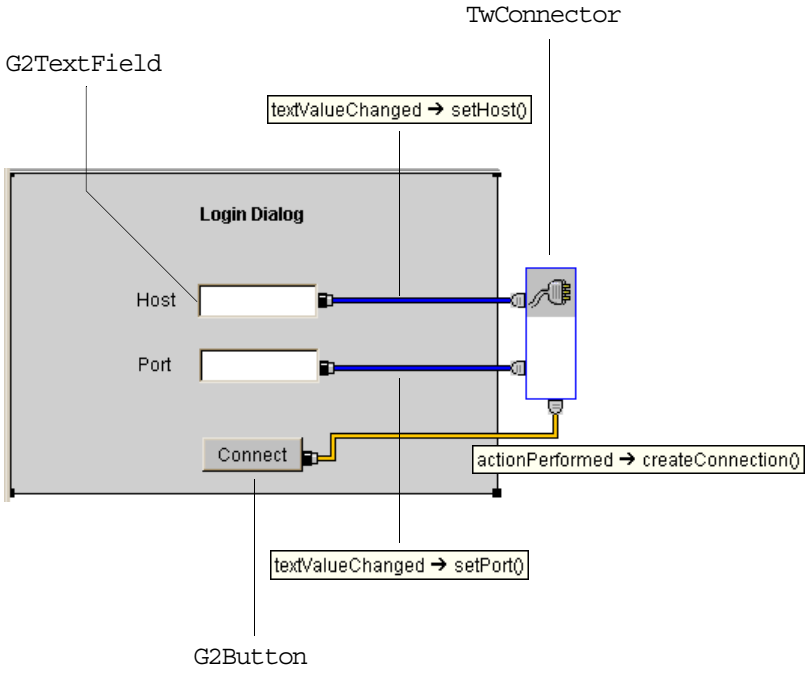
One common use of the `G2TextField` component is to obtain data that a user inputs and pass it as an argument to a G2 method or procedure that requires it. To do this:

- Choose an event that notifies listeners when the user has entered the value.
- Create an event hookup from the `G2TextField` to a component and invoke a method that takes an argument.

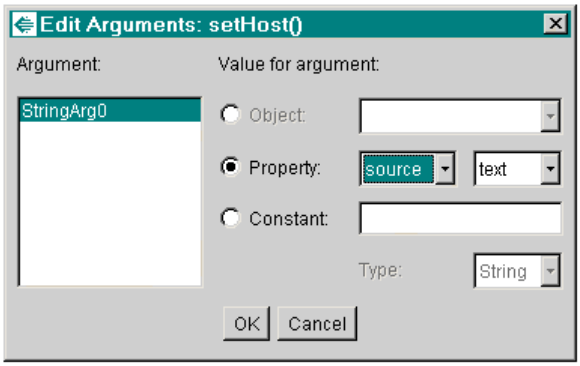
For example, you might use a `focusLost` event, which occurs when the cursor moves to another text field, to pass the `host` and `port` arguments to the `setHost` and `setPort` methods of a `TwConnector`.

You can also use this technique to provide arguments to G2 methods or procedures, which requires that you use the G2 Bean Builder to create a component representation of a G2 class. When the value gets uploaded to G2, your application can perform the action, using input data from the `G2TextField`. For an example, see “Using G2 Item Components in Dialogs” on page 226.

This dialog allows the user to set the host and port of the G2 to which to connect, then connect to the specified server:

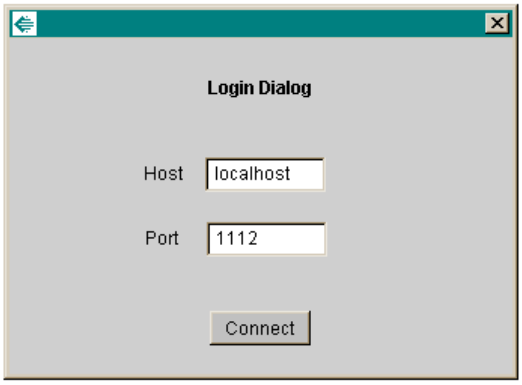


When you connect the `textValueChanged` events from the `G2TextField` components and you choose the specified target methods, the Edit Arguments dialog appears for you to specify the arguments to the methods. In both cases, the argument to the target method is the `text` property of the source, which is the `G2TextField`. This figure shows the configured Edit Arguments dialog for both target methods:



When the user enters values for the Host and Port fields, the `TwConnector` calls its `setHost` method with the text value entered in the Host field as its argument, and it calls its `setPort` method with the text value entered in the Port field as its

argument. When the user clicks the Connect button, the `TwConnector` uses the specified host and port information to create the connection, as this dialog shows:



ItemProxy

An `ItemProxy` component represents an item in G2. When you create a custom item properties dialog, you use an `ItemProxy` to represent the item whose attributes the dialog displays and edits.

If you are building dialogs in a JavaBeans-compliant visual programming environment, you can use an `ItemProxy` with an `ItemRetriever` to connect to G2 and retrieve an item to test the dialog.

You can also use an `ItemProxy` to edit the attributes of a subobject.

The `ItemProxy` handles `objectUpdated` and `objectChanged` events for the attributes your dialog is displaying.

The `ItemProxy` is not a visual component, though it uses this representation:

`com.gensym.controls.ItemProxy`

Properties and Accessor Methods

These are the properties and associated accessor methods of an `ItemProxy` component:

Property Get Property Set Property	Type	Description

`attributes`
`getAttributes`
`setAttributes`
	`SymbolVector`	The names of the G2 attributes connected to this `ItemProxy` as a `com.gensym.controls.SymbolVector`.
		When initializing this property in code, call `Symbol.intern(String string)`, where `string` is the attribute name as an upper-case string.

`autoDownload`
`getAutoDownload`
`setAutoDownload`
	`boolean`	Determines when the `ItemProxy` downloads value changes from G2 to a data-aware component. The default value is `true`, which causes the `ItemProxy` to download value changes from G2 to a component as they occur. Set this property to `false` to hold value changes from G2 for downloading all at once, using an OK or Apply button.

Property Get Property Set Property	Type	Description
autoUpload getAutoUpload setAutoUpload	boolean	Determines when the ItemProxy uploads value changes from a data-aware component to G2. The default value is true, which causes the ItemProxy to upload value changes from a component to G2 as they occur. Set this property to false to hold value changes from the dialog components for uploading to G2 all at once, using an OK or Apply button.
subObjectAttribute getSubObjectAttribute setSubObjectAttribute	Symbol	The name of a G2 attribute associated with this component that contains a subobject, as a com.gensym.util.Symbol. When initializing this property in code, call Symbol.intern(String string), where string is the attribute name as an upper-case string.

For information on the additional properties that this component includes, see "Using Standard Java Properties" on page 224.

Events

These are the events that an ItemProxy component generates:

Event	Description
itemDeleted	Dispatched when the item that this component represents gets deleted in G2.
objectUpdated	Dispatched when the value of the attribute that this component represents gets updated in the G2 server.

For details, see these classes in the following G2 JavaLink packages:

- com.gensym.util.ItemListener
- com.gensym.dlgevent.ObjectUpdateListener and ObjectUpdateEvent

For information on the standard events that this component generates, see "Using Standard Java Events and Methods" on page 225.

Methods

These are the target methods of an `ItemProxy` component:

Method	Argument	Description
`download`	N/A	Causes the `ItemProxy` to download a new attribute value from G2 to a data-aware component.
`editItem`	N/A	Launches a dialog for editing the item that this `ItemProxy` represents.
`getProxy`	N/A	Returns the `com.gensym.classes.Item` that this `ItemProxy` represents in G2 JavaLink.
`itemRetrieved`	`ItemRetrievalEvent`	Responds to `com.gensym.controls.ItemRetrievalEvents` by retrieving the item that this `ItemProxy` represents.
`itemRetrievalFailed`	`ItemRetrievalEvent`	Responds to `com.gensym.controls.ItemRetrivalEvents` by generating a `com.gensym.jgi.G2AccessException` and a `com.gensym.message.MessageEvent`, which indicates that the `ItemProxy` could not retrieve the item. This method also clears the current item.
`objectChanged`	`ObjectChangeEvent`	Responds to `ObjectChangeEvents` by calling the `upload` method of the `ItemProxy`, if `autoUpload` is `true`, which uploads changes in the attributes of the item that the `ItemProxy` holds, from a component to G2.

Method	Argument	Description
objectUpdated	ObjectUpdatedEvent	Responds to ObjectUpdateEvents by calling the download method of the ItemProxy, if autoDownload is true, which downloads changes in the attributes of the item that the ItemProxy holds, from G2 to a component.
setProxy	Item	Sets the com.gensym.classes.Item that the ItemProxy represents. Pass null to clear the current item.
upload	N/A	Causes the ItemProxy to upload changed attribute values from a component to G2.

For details on the objectChanged and objectUpdated methods, see these classes in the G2 JavaLink com.gensym.dlgevent package:

- ObjectChangeListener and ObjectChangeEvent
- ObjectUpdateListener and ObjectUpdateEvent

For information on the standard methods that this component defines, see "Using Standard Java Events and Methods" on page 225.

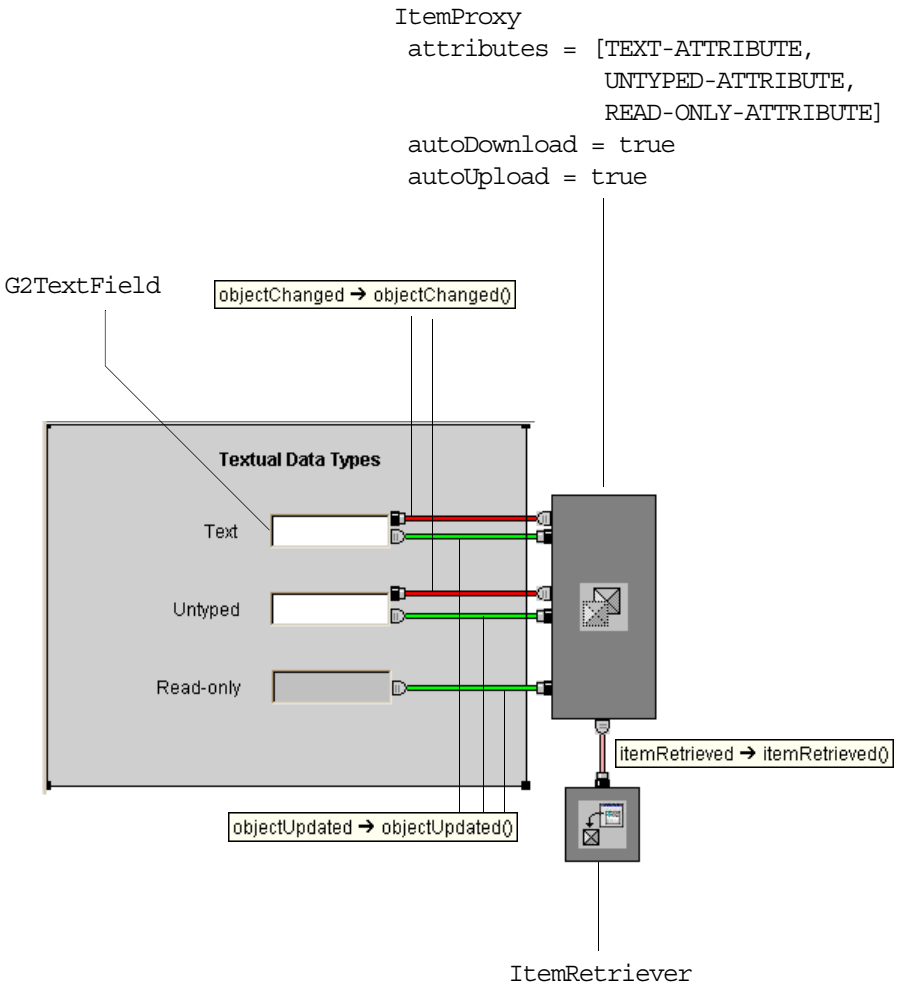
Examples

Automatically Uploading and Downloading Changes

This dialog allows the user to edit several textual attributes of a G2 item named demo-item, which is an instance of demo-class. The ItemProxy generates objectUpdated events when the G2 item changes, and it receives objectChanged events when the dialog component changes. Because the Read-only field is read-only, the ItemProxy simply generates objectUpdated events.

Because the autoUpload property is set to true, the default, the ItemProxy automatically uploads changes based on objectChanged events from the G2TextField components to G2. Similarly, because the autoDownload property is set to true, also the default, the ItemProxy automatically downloads changes based on objectUpdated events from G2 to the G2TextField components.

The ItemProxy also receives an itemRetrieved event from an ItemRetriever, which invokes the proxy's itemRetrieved method to retrieve the item.



For an example of using an ItemRetriever when autoUpload and autoDownload are set to false, see "Example" on page 234 under DialogCommand.

Editing Attributes of a Subobject

You can use an ItemProxy to edit the attributes of an object that is defined as a subobject of another object. For example, you might define the location attribute of an instance of the company class to be an instance of a building class, which defines these attributes:

- street-address
- city

- state
- zip-code

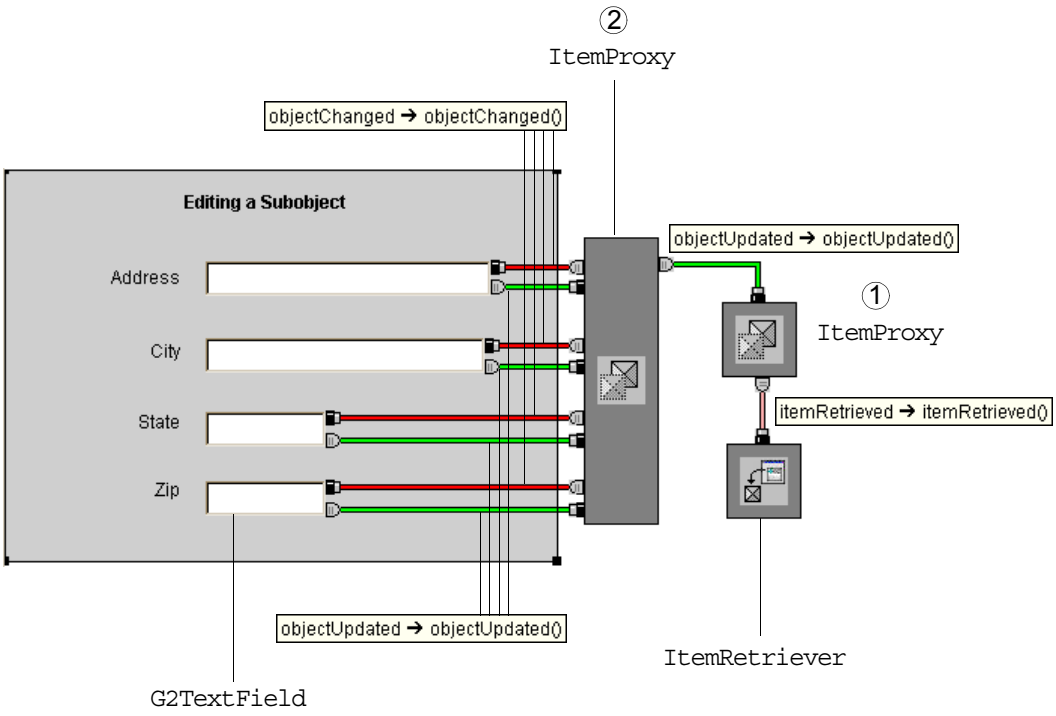
To edit the attributes of a subobject, you use two `ItemProxy` components, as follows:

- One `ItemProxy` names the attribute of the item that defines the subobject, for example, `LOCATION`, which receives events from the connection component.
- The other `ItemProxy` names the attributes of the subobject, for example, `STREET-ADDRESS`, `CITY`, `STATE`, and `ZIP-CODE`, and specifies the subobject attribute as the value of the `subObjectAttribute` property.

You hookup the components as follows:

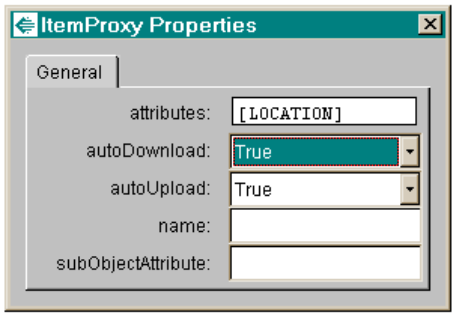
- Connect a connection component, such as a `ItemRetriever`, to the first `ItemProxy` to retrieve the item.
- Connect the first `ItemProxy` to the second `ItemProxy`, using a standard `objectUpdated` event.
- Connect scalar controls, such as a `G2TextField`, to the second `ItemProxy`, using standard `objectUpdated` and `objectChanged` events.

This figure shows such a dialog, used for editing the location of a company:

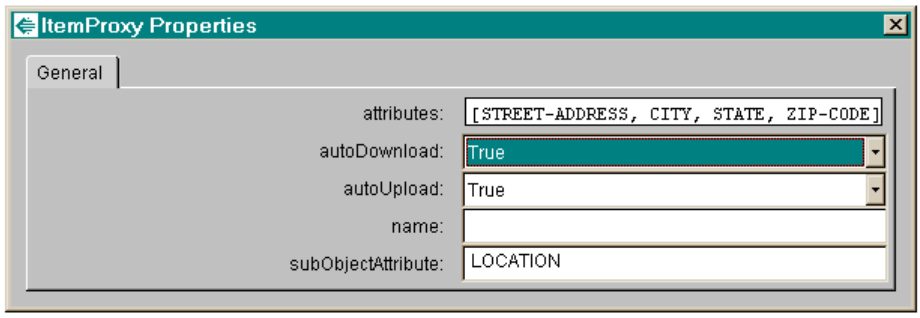


This figure shows the properties dialogs for each `ItemProxy` in the previous diagram:

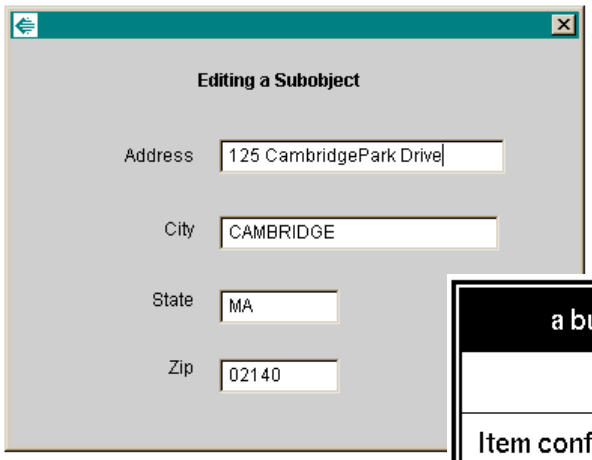
①



②



Here is the dialog that appears when you launch and the corresponding table in G2:



a building, the location of GENSYM	
Notes	OK
Item configuration	none
Names	none
Street address	"125 CambridgePark Drive"
City	cambridge
State	ma
Zip code	"02140"

Finally, here are the class definitions for the company and building, as well as the gensym item and its table:

COMPANY

Class specific attributes headquarters is a structure, initially is
structure ();
location is an instance of a building, initially is
an instance of a building

BUILDING

Class specific attributes street-address is a text, initially is "";
city is a symbol, initially is g2;
state is a symbol, initially is g2;
zip-code is a text, initially is ""

GENSYM

GENSYM, a company	
Notes	OK
Item configuration	none
Names	GENSYM
Headquarters	structure (city: the symbol cambridge, state: the symbol ma)
Location	a building

For an explanation of the headquarters attribute, see “StructureMUX” on page 300.

StructureMUX

Use a `StructureMUX` component to represent G2 structure values, which you cannot view in a single component. You use one of the scalar controls to view and edit the elements of the structure. The scalar controls represent the attributes within the structure, while the `StructureMUX` represents the structure attribute.

A `StructureMUX` is not a visual component, but it uses this representation:

`com.gensym.controls.StructureMUX`

Properties and Accessor Methods

These are the properties and associated accessor methods of a `StructureMUX` component:

Property Get Property Set Property	Type	Description

`attribute`
`getAttribute`
`setAttribute`
	`Symbol`	The name of the G2 attribute associated with this component, as a `com.gensym.util.Symbol`.
		When initializing this property in code, call `Symbol.intern(String string)`, where `string` is the attribute name as an upper-case string.

For information on the additional properties that this component includes, see "Using Standard Java Properties" on page 224.

Events

These are the events that a `StructureMUX` component generates:

Event	Description
`objectChanged`	Dispatched when the value of this component changes. You connect an `objectChanged` event from the `StructureMUX` to an `ItemProxy`.
`objectUpdated`	Dispatched when the value of the attribute that this component represents gets updated in the G2 server. You connect an `objectUpdated` event from the `StructureMUX` to a scalar control, such as a `G2TextField`.

For details, see these classes in the G2 JavaLink `com.gensym.dlgevent` package:

- `ObjectChangeListener` and `ObjectChangeEvent`
- `ObjectUpdateListener` and `ObjectUpdateEvent`

For information on the standard events that this component generates, see "Using Standard Java Events and Methods" on page 225.

Methods

These are the target methods of a `StructureMUX` component:

Method	Argument	Description
`objectUpdated`	`ObjectUpdatedEvent`	Responds to `ObjectUpdateEvents` by setting the value of this component to the current value of the attribute that this component represents.

For details on the `objectUpdated` method, see these classes in the G2 JavaLink `com.gensym.dlgevent` package:

- `ObjectUpdateListener`
- `ObjectUpdateEvent`

For information on the standard methods that this component defines, see "Using Standard Java Events and Methods" on page 225.

Example

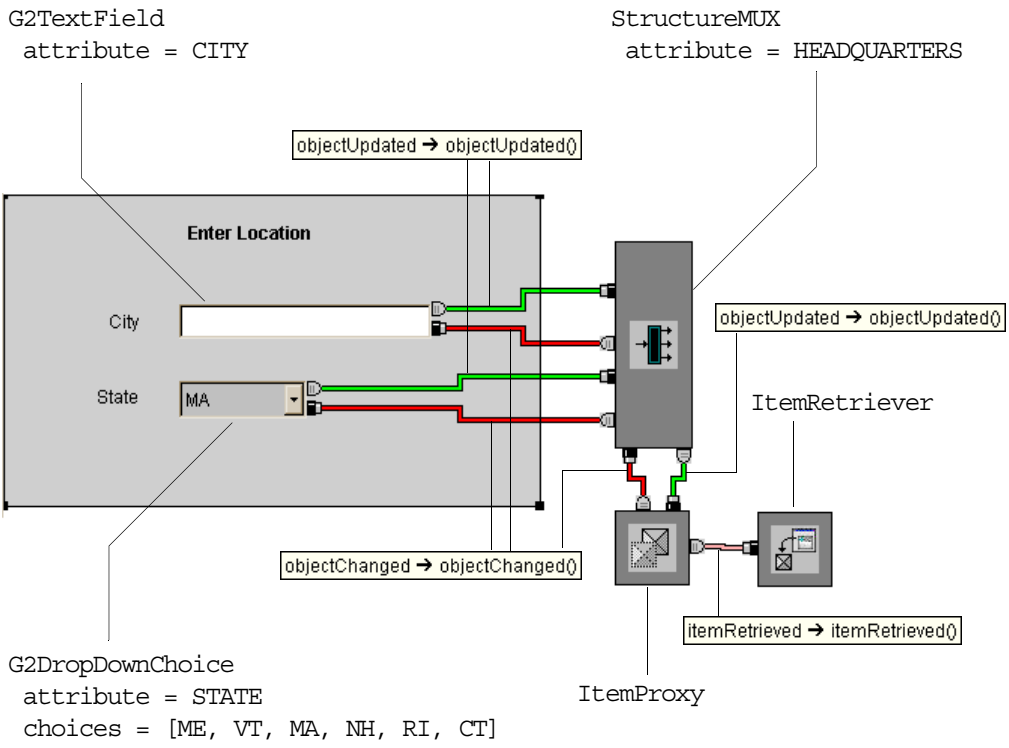
This dialog shows how to use a `StructureMUX` with a `G2TextField` and a `G2DropDownChoice` to edit the attributes of a G2 structure. The structure has two attributes, city and state, each of which is a type of symbol. To set up the dialog to edit the structure in G2 and to receive updates to the structure in the dialog, you use `objectChanged` and `objectUpdated` events, as follows:

- The `objectChanged` events flow from the scalar controls, the `G2TextBox` and `G2DropDownChoice`, to the `StructureMUX`, and from the `StructureMUX` to the `ItemProxy`.
- The `objectUpdated` events flow from the `ItemProxy` to the `StructureMUX`, and from the `StructureMUX` to the scalar controls.

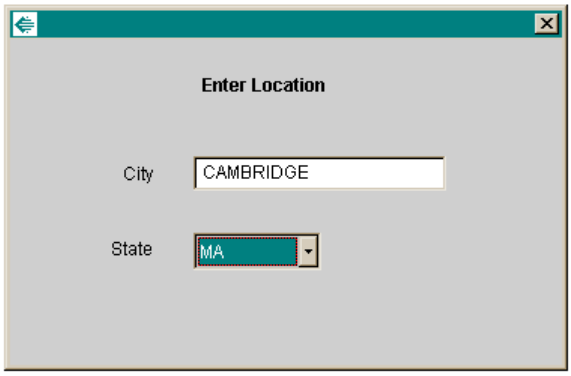
In addition, you must specify the `attribute` property of the controls as follows:

- The `attribute` of the `StructureMUX` is the name of the attribute whose value is a G2 structure.
- The `attribute` of the scalar controls is the name of the attributes of the structure.

The following figure shows the event hookups and `attribute` properties of the sample dialog:

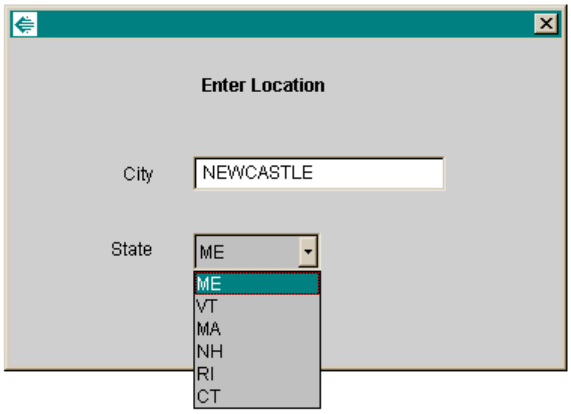


This figure shows the dialog that appears when you first launch the dialog and the corresponding structure in G2, which provides defaults for the `G2TextField` and `G2DropDownChoice`:



Headquarters	structure (city: the symbol cambridge, state: the symbol ma)

This figure shows the result of changing the City and State fields in the dialog and the changed structure in G2:



Headquarters	structure (city: the symbol newcastle, state: the symbol me)

16

Launching Custom Item Properties Dialogs

Describes how to register custom item properties dialogs with a dialog manager to replace automatically generated dialogs for editing the properties of G2 items.

gensym

Introduction

When you create a custom item properties dialog to replace the automatically generated dialog for a specific G2 class or item, you need to configure the Telewindows2 (TW2) Toolkit client to launch the custom dialog.

You have two options for creating and launching individual custom item properties dialogs:

- Use a JavaBeans-compliant visual programming environment to create a **dialog resource**, which is a serialized (`.ser`) file, then register the resource with the `DialogManager` by specifying a G2 media bin, a URL location, or a serialized dialog resource file located on the client.
- Use Java to create a **dialog class**, then register that class with the `DialogManager` by specifying its name.

Because the TW2 Toolkit dialog components are packaged as Java Beans, you can create custom item properties dialogs in a JavaBeans-compliant visual programming environment, such as Symantec Visual Café or Borland J Builder.

Registering dialog resources and dialog classes requires calling a version of this method on a `com.gensym.dlgruntime.DialogManager`:

```
setDialogResourceEntry
```

Typically, when you register custom item properties dialogs, you call the method from your G2 application as a remote procedure call (RPC). To do this, you:

- Declare the RPC in G2.
- Create a procedure that calls the RPC.
- Create a rule that calls your procedure when it detects that a TW2 Toolkit client session has been established.

You can register dialogs with an entire class of items or an individual instance of a class. You can also specify a locale when you register the dialog resource to localize dialog text.

When you create custom item properties dialog classes in a visual Java programming environment, the class you create must implement the `com.gensym.dlgruntime.SingleItemEditor` or `MultipleItemEditor` interface, which allows editing of one or multiple items through the dialog class. You represent the item or items to edit by using a `com.gensym.controls.ItemProxy`.

This chapter describes:

- How to register dialog resources in G2, using an RPC call.
- The steps required to create a dialog class in Java and how to register those dialog classes, using an RPC call.
- How to create your own `DialogManager` for registering dialog resources or for customizing the behavior when the user chooses Properties on an item.

For information on...	See...
Customizing automatically generated dialogs for classes based on their contents	"Customizing Automatically Generated Dialogs" on page 321.
Using the `DialogManager` to manage and launch general dialogs.	Chapter 18, "Launching General Dialogs" on page 337.

Packages Covered

com.gensym.dlgruntime

Interfaces

```
Commandable
DialogManagerFactory
SingleItemEditor
MultipleItemEditor
```

Classes

```
DefaultDialogManagerFactory
DialogManager
```

com.gensym.classes

Interfaces

```
MediaBin
```

Relevant Demos

The dialog resources that this chapter shows how to register are available online in this directory, depending on your platform:

NT:	`%SEQUOIA_HOME%\classes\com\gensym\demos\customdialogs`
UNIX:	`$SEQUOIA_HOME/classes/com/gensym/demos/customdialogs`

To run these dialog resources, you must load this KB file into G2, depending on your platform:

NT:	`%SEQUOIA_HOME%\kbs\dialog-demo.kb`
UNIX:	`$SEQUOIA_HOME/kbs/dialog-demo.kb`

Registering Custom Item Properties Dialog Resources

When a client application creates a G2 connection, the connection creates an associated `com.gensym.dlgruntime.DialogManager`. Once the custom dialog exists as a resource in a G2 media bin, you must register it with the `DialogManager` associated with the connection to use it in place of the automatically generated properties dialog for an item.

The `DialogManager` is responsible for launching and managing both custom and automatically generated item properties dialogs. To do this, it:

- Determines the resource to use.
- Generates the dialog resource, if necessary.
- Caches the dialog resource on the client after generating it.

Caching the generated resource on the client eliminates the need to re-create the dialog resource each time a request is made to view item properties.

If your application registers a custom item properties dialog resource for an item with the `DialogManager` after connecting to G2, whenever a user chooses Properties from the item's popup menu or double-clicks an item, the `DialogManager` launches the custom dialog rather than the automatically generated one.

You register custom dialog resources by completing these tasks in G2:

- Monitor client sessions through a whenever rule. When a TW2 Toolkit client establishes a session, G2 can register custom item properties dialogs with the client's `DialogManager`.
- Create a remote procedure declaration for the `DialogManager` method `setDialogResourceEntry`, called set-dialog-resource-entry.
- Call the set-dialog-resource-entry remote procedure from G2 to register one or more custom item properties dialog resources with a class of items or an individual instance.
- Create a procedure that calls the remote procedure across the interface associated with the TW2 Toolkit client session.

The following sections explain these steps in more detail.

Monitoring Client Sessions

As described in "Determining the Connectivity Class to Use" on page 30, whenever a TW2 Toolkit client establishes a connection, G2 creates an instance of a ui-client-interface to represent the client connection. Similarly, whenever the

client establishes a login session, G2 creates a ui-client-session item to represent the login session.

You can monitor client session activity in G2 by using a whenever rule. For example, this is the text of the rule that the `dialog-demo.kb` uses, which passes the ui-client-session item to the dialog registration procedure:

```
whenever any ui-client-session session is activated
then start demo-dialog-registration-procedure (session)
```

The client session is an instance of a gsi-interface. You call the remote procedure across this interface, as described in "Creating a Procedure that Calls the RPC Across the Interface" on page 315.

Declaring the Remote Procedure in G2

To register one or more custom dialogs from G2, you must write a procedure that calls the `DialogManager` class method `setDialogResourceEntry` remotely, through the use of a G2 remote procedure call (RPC) declaration.

The `setDialogResourceEntry` method is one of several TW2 Toolkit `DialogManager` class methods of that name. The method you call to register custom item dialog resources is specifically for use from G2 and indicates whether a resource exists for a dialog, and if it does, the location of that resource. The registration method also allows you to register dialog classes by specifying the Java class name instead of the resource location.

The `sequoia-support.kb` module includes this remote procedure declaration:

```
declare remote set-dialog-resource-entry
    ({ITEM-OR-CLASS} (item-or-value as
    handle), {USER-MODE} symbol,
    {LOCALE} structure, {RESOURCE-
    DESCRIPTION}(structure as handle)) = ()
```

Your KB must include a remote-procedure-declaration with this signature. For more information about remote procedure declarations, see the *G2 Reference Manual*.

Calling the Remote Procedure

The procedure you write to register one or more custom item dialog resources or classes calls the `setDialogResourceEntry` method of the `com.gensym.dlgruntime.DialogManager` class as a remote procedure call.

This is the signature of the remote procedure:

```
set-dialog-resource-entry
    (item-or-class: (item-or-value as handle), user-mode: symbol,
     locale: structure, resource-description: (structure as handle))
```

Argument	Description
item-or-class	The G2 item or class, as a handle, for which a dialog is to be launched. Specify this as either: • An instance name: my-object • A class name: the symbol *class-name*
user-mode	The G2 user mode for which the dialog resource is applicable, as a symbol.
locale	A G2 structure consisting of language, country code, and variant subattributes.
resource-description	A G2 structure, described after the *locale* structure, as a handle, of the dialog resource location or dialog class name, and whether it is applicable to an entire class.

The following sections explain the *locale* and *resource-description* structure arguments, provide examples of different types of resource descriptions, and show how to register a custom item properties dialog resource for a G2 class and for a G2 instance.

Locale Structure

The *locale* structure consists of these three subattributes for localizing the dialog resource text:

Subattribute	Type	Description
language	text	A two-character language code as a text value.
country	text	A two-character country code as a text value.
variant	text	An optional code that is vendor- and browser-specific.

Both the language and country values are lower-case, two-letter codes defined by the two standards:

- ISO Language Code
- ISO Country Code

For more information, refer to the API documentation for `java.util.Locale`.

For example, you might declare the locale structure argument for your client in a local variable of your procedure. As an example, the `dialog-demo.kb` sample procedure declares the locale argument as follows for a particular operating system:

```
{The locale structure for US. See java.util.Locale. }
locale: structure = structure (LANGUAGE: "en", COUNTRY: "US");
```

Resource Description Structure

The *resource-description* structure consists of these subattributes:

Subattribute	Type	Description
source	item-or-value	The type of location for your dialog resource or class. The options are: • MediaBin • the symbol URL • the symbol FILE • the symbol JAVA-CLASS If you are registering a dialog resource, then you specify one of the first three values and a corresponding value for the location attribute. If you are registering a dialog class, then you specify the last value and a corresponding value for the class-name attribute.

Subattribute	Type	Description
location	text	The specific location of the dialog resource. You specify the location in conjunction with the source subattribute, as follows: • MediaBin: A text string that defines a pathname in the MediaBin, separated by slashes. • URL: The full URL path, including the protocol, such as FTP or HTTP. • FILE: The pathname for the file.
class-name	string	The fully qualified Java class name as a string, for example: "com.test.dlg.EmployeeDialog"
applicable-for-class	truth-value	Whether the dialog should be used for the entire class of items. You specify this attribute only when you do not want to use a custom item properties dialog for an entire class. The default value is true.

If the source subattribute contains the symbol URL, there should be an attribute that holds a string containing the protocol, such as "http", and the location is used as the tail of the URL.

If the source attribute contains the symbol FILE, the location subattribute should contain a file name.

You specify either the location or class-name subattribute, but not both, depending on the value of the source subattribute.

The following sections provide examples of registering dialog resources stored in a media bin, as a URL, and in a file.

For an example of registering a dialog class, see "Registering Custom Item Properties Dialog Classes" on page 316.

Specifying a Media Bin Resource Location

If you stored your dialog resource in a media bin within your KB, you specify:

- The source subattribute as the name of the media bin you created, as a symbol.
- The location subattribute as the serialized dialog resource file name in the media bin, as a text string, preceded with a slash (/) character.

For an explanation of the across intf statement in the following example, see "Creating a Procedure that Calls the RPC Across the Interface" on page 315.

To specify a media bin location:

```
call set-dialog-resource-entry
     (tank-1,
      the symbol ADMINISTRATOR,
      locale,
      structure
           (SOURCE: my-media-bin,
            LOCATION: "/tank1.ser",
            APPLICABLE-FOR-CLASS: false))
     across intf;
```

Specifying a URL Resource Location

If your serialized dialog resource file is accessible through a URL location, you specify:

- The source subattribute as URL.
- The location subattribute as the actual URL, as a text string.

To specify a URL location:

```
call set-dialog-resource-entry
     (tank-1,
      the symbol ADMINISTRATOR,
      locale,
      structure
           (SOURCE: the symbol URL,
            LOCATION: "http://mydir/my-special-dir/tank1.ser",
            APPLICABLE-FOR-CLASS: false))
     across intf;
```

Specifying a File Resource Location

While we do not recommend saving a dialog as a serialized file and managing it as a separate resource, your application might need to do this. If you saved your serialized dialog resource file on the disk, you specify:

- The source subattribute as the symbol FILE.
- The location subattribute as the directory path to the dialog resource file, as a text string.

For example, your application might create a custom dialog resource for the tank1 instance of the tank class, named `tank1.ser`. To do this, create a local text variable for the location of the dialog file, for example:

```
DialogLocation: text;
DialogLocation = "c:\Program Files\Gensym\g2-6.1\tw2\resources\tank1.ser";
```

To specify a file location using a variable:

```
call set-dialog-resource-entry
     (tank-1,
      the symbol ADMINISTRATOR,
      locale,
      structure
           (SOURCE: the symbol FILE,
            LOCATION: DialogLocation,
            APPLICABLE-FOR-CLASS: false))
     across intf;
```

Registering a Custom Dialog for a Class

To register a custom dialog resource for an entire class of items, specify the first argument to the set-dialog-resource-entry remote procedure as follows:

the symbol *class-name*

In addition, the applicable-for-class subattribute of the *resource-description* structure determines whether the dialog resource file you pass to the set-dialog-resource-entry remote procedure is for the entire class or for the instance named by the first argument.

By default, the value of this subattribute is true. Thus, when registering a custom dialog resource for a class, you do not need to provide this subattribute. For example, the `dialog-demo.kb` calls the procedure as follows to register a custom dialog for the entire dial class of items:

```
{ Use a single overriden resource from media-1 for all dials }
call set-dialog-resource-entry
     (the symbol DIAL,
      the symbol ADMINISTRATOR,
      locale,
      structure (SOURCE: media-1, LOCATION: "/dial.ser"))
     across intf;
```

Registering a Custom Dialog for Items

You can register a custom item property dialog for an individual instance of a class by providing the item as the first argument to the remote procedure.

In addition, if you do not want to register the custom item property dialog for an entire class, you must explicitly specify the applicable-for-class subattribute as false.

The `dialog-demo.kb` illustrates how to register custom dialog resources for two instances of the juggler class. The dialog resource for the instance named jughead is stored with the KB in the media-bin item called media-1, while the dialog resource for the instance named johnny is stored in a specific location on the client. Notice that in both cases, the applicable-for-class subattribute is explicitly set to false.

```
{ Use specific resources for JUGHEAD (from media-1) and JOHNNY (from disk) }
call set-dialog-resource-entry
     (jughead,
      the symbol ADMINISTRATOR,
      locale,
      structure (SOURCE: media-1, LOCATION: "/jughead.ser",
      APPLICABLE-FOR-CLASS: false))
     across intf;

call set-dialog-resource-entry
     (johnny,
      the symbol ADMINISTRATOR,
      locale,
      structure (SOURCE: the symbol FILE, LOCATION: dlgLocn,
      APPLICABLE-FOR-CLASS: false))
     across intf;
```

In the `dialog-demo.kb`, the `DialogManager` automatically generates dialogs for all other instances of the juggler class.

Creating a Procedure that Calls the RPC Across the Interface

When creating a procedure to register custom dialog resources, you must pass the ui-client-session item as an argument to the procedure to obtain the interface across which to call the remote procedure.

To create a procedure that calls the RPC across the interface:

1. Create a procedure that passes the ui-client-session as an argument, for example:

   ```
   my-dialog-registration-procedure (session: class ui-client-session)
   ```

2. Call the remote procedure across the interface, by referring to the ui-client-interface attribute of the ui-client-session, for example:

   ```
   call set-dialog-resource-entry (tank-1, the symbol ADMINISTRATOR, locale,
        structure (SOURCE: the symbol FILE, LOCATION: DialogLocation,
        APPLICABLE-FOR-CLASS: false))
        across the ui-client-interface of session;
   ```

Registering Custom Item Properties Dialog Classes

You can create item properties dialogs in Java, using TW2 Toolkit dialog components. Dialogs that you create in Java are classes.

The technique you use for registering dialog classes created in Java is similar to the technique you use for registering dialog resources: you call the `setDialogResourceEntry` method on the `DialogManager` as a remote procedure. However, to register dialog classes, you use a slightly different version of this method, which specifies a Java dialog class rather than a dialog resource. In addition, when creating your Java dialog class, you must use particular TW2 Toolkit classes to keep track of the G2 item the dialog is editing.

Telewindows2 Toolkit has been tested with the following two Java-based IDEs:

- Symantec Visual Café
- Borland J Builder

The following sections describe how to:

- Create a dialog class that you can use in a TW2 Toolkit application to edit G2 items.
- Register a Java dialog class within G2.

Creating Dialog Classes for Editing G2 Items

The following steps show the general techniques for defining a Java class for use as a G2 item properties dialog from within a TW2 Toolkit client application.

Note You must include in your IDE's `CLASSPATH` environment variable these JAR files: `sequoia.jar` file and `coreui.jar`, located in the `classes` directory of the `Java` directory of your TW2 Toolkit product directory, and `javalink.jar`, `core.jar`, and `classtools.jar`, located in the `classes` directory of your G2 JavaLink product directory.

To create a dialog class for use as a G2 item properties dialog in the client:

1. Define a Java class for the dialog that inherits from `java.awt.Component`, or a subclass, for example, `java.awt.Dialog`.
2. Define a constructor, which takes no arguments.

   Currently, the method that registers the dialog class does not take as one of its arguments the window from which to launch the dialog. Therefore, if your dialog class extends `java.awt.Dialog`, you must provide as part of the constructor the window from which the dialog is to be launched.

If your TW2 Toolkit client application extends `com.gensym.core.UiApplication`, you can provide the window by calling the `getCurrentFrame` static method on `UiApplication`. For example:

```
import java.awt.Dialog;
import com.gensym.core.UiApplication;

public class MyDialog extends Dialog {
public MyDialog() {
   super(UiApplication.getCurrentFrame());
   }
}
```

If your Java application does not extend `UiApplication`, you are responsible for getting the frame as part of the constructor.

If your dialog class is not a subclass of `java.awt.Dialog`, you can define an empty constructor with no arguments.

For information on `UiApplication`, see Chapter 9 "Creating Telewindows2 Toolkit Applications" in the *Telewindows2 Toolkit Java Developer's Guide: Application Classes*.

3 Use the following classes to keep track of the item that the dialog is supposed to edit:

- `com.gensym.controls.ItemProxy` — Your dialog class should use an `ItemProxy` to hold the item being edited.
- `com.gensym.dlgruntime.SingleItemEditor` — The dialog class must implement the `SingleItemEditor` interface. This interface has a single method, `getProxy()`, which should return the `ItemProxy`. The superior class for the interface, `com.gensym.dlgruntime.Commandable`, defines the `getDialogCommand` method, which can safely return `null`.

Defining a Procedure that Calls the RPC to Register the Dialog Class

The following example shows how to call the set-dialog-resource-entry remote procedure from G2 to associate the Java dialog class with a G2 class for use as a custom item properties dialog in the client. Once you have registered the dialog class, choosing Properties from the popup menu for any instance of the class launches the custom dialog. You can also use set-dialog-resource-entry to register the dialog class for a particular instance.

The technique for registering the dialog class directly by using the TW2 Toolkit API from within your Java application would be similar, except that you would call the `setDialogResourceEntry` method directly on `com.gensym.dlgruntime.DialogManager`.

To register a Java dialog class as a custom properties dialog for a G2 class:

1. Monitor the client session, as described in "Monitoring Client Sessions" on page 308.
2. Declare the remote procedure, as described in "Declaring the Remote Procedure in G2" on page 309.
3. Create a procedure that calls set-dialog-resource-entry and provide as the final structure these subattributes:
   - The source subattribute is the symbol JAVA-CLASS.
   - The location subattribute is the Java class name that is the dialog, as a text string.

The following procedure shows an example of registering a Java dialog class for use as the properties dialog for a G2 class. The key differences between this method call and the method call shown in the `dialog-demo.kb` is the last argument to the set-dialog-resource-entry RPC, which describes the type and location of the dialog class.

```
demo-dialog-registration-procedure(session: class ui-client-session)
    intf: class gsi-interface = the ui-client-interface of session;
    begin
        call set-dialog-resource-entry
            (the symbol EMPLOYEE,
             the symbol ADMINISTRATOR,
             structure (LANGUAGE: "en", COUNTRY: "US"),
             structure (SOURCE: the symbol JAVA-CLASS,
                        CLASS-NAME: "com.test.dlg.EmployeeDialog"))
                        across intf;
    end
```

Creating Your Own Dialog Manager

By default, each connection has an associated `DialogManager` for launching and reading dialog resources and dialog classes for use as custom item properties dialogs. By default, `com.gensym.ntw.TwGateway` uses a `DefaultDialogManagerFactory` to generate a `DialogManager` for this purpose.

You can create your own `DialogManager` to customize:

- The behavior when the user choose Properties from the popup menu on an item.
- The dialog registration procedure that the dialog manager uses, for example, to register dialog resources that are saved out in a format other than a serialized file or a Java class file.

You use an implementation of `DialogManagerFactory` to generate your `DialogManager`. You should install the factory in the `main` method of your

application, before you make a connection request. That way, when you register custom item properties dialogs in G2 by calling `setDialogResourceEntry` across an interface, the connection uses the `DialogManager` that your `DialogManagerFactory` creates. When the client makes a connection request, the connection asks your factory to create a dialog manager, before it actually connects to G2.

To create your own dialog manager:

1. Subclass `DialogManager` and override its methods, as needed.

   For example, you might override the `editItem` method to customize the behavior when the registered dialog resource is requested, or you might implement your own `setDialogResourceEntry` method to register different types of dialog resources.

2. Implement the `DialogManagerFactory` interface.

3. Install your implementation of `DialogManagerFactory` by calling the `TwGateway.setDialogManagerFactory` static method.

17

Customizing Automatically Generated Dialogs

Describes how to customize automatically generated dialogs that appear when the user chooses Properties on an item on a workspace.

gensym

Introduction

Each connection to G2 has a `DialogManager` that manages requests for item properties dialogs. Each `DialogManager` specifies a `GeneratedDialogFactory` that is responsible for generating a properties dialog for a specific item. The `com.gensym.gcg` package provides the `DefaultGeneratedDialogFactory` class, a default implementation of the `GeneratedDialogFactory` interface, which provides useful methods for each step in the creation and assembly of a dialog.

You can create your own factory to customize automatically generated properties dialogs for items. Typically, you subclass `DefaultGeneratedDialogFactory` and customize the automatically generated dialogs by using one of two approaches:

- Bottom-up — Call `super` in the subclass constructor, then override various methods to customize the default dialog components.
- Top-down — Call various methods on `DefaultGeneratedDialogFactory` to construct the dialog from scratch from its components.

You set the factory in the `DialogManager` associated with a connection.

DeafultGeneratedDialogFactory

To customize an automatically generated dialog, using a bottom-up approach, you subclass `DefaultGeneratedDialogFactory` and override one or more of its methods. For example, to include only user-defined attributes in all automatically generated item properties dialogs, define a subclass of `DefaultGeneratedDialogFactory` and override the `getG2AttributeEditor` method. When taking a top-down approach, you would call these methods to build the dialog from individual components.

The following is a top-down list of some of the public and protected methods in `DefaultGeneratedDialogFactory`.

- `generateDialog`

  Explicitly generates a properties dialog for editing an item. This method implicitly calls the `generateAttributePanel` and `generateCommandButtonPanel` methods and determine the contents of the dialog.

- `generateAttributePanel`

  Creates a `java.awt.Container` for the attribute panel area and tab pages of the automatically generated dialog. When subclassing `DefaultGeneratedDialogFactory`, calling this method on the superior class automatically creates a Notes tab whenever the item has notes and hides it otherwise. When constructing a dialog from scratch, you can call this method on the superior class to create the attribute panel and tab pages and place the results inside your own type of container.

- `getAttributeEditors`

  Returns a list of `AttributeEditor` objects for an item, which allows you to rearrange the attributes and editors within a tab page or across tab pages

- `getAttributeInfos`

  Returns a list of `com.gensym.gcg.AttributeInfo` objects for an item, which provides access to the visible attributes for the current user mode of

the connection. You can use this information to filter the attributes to include in the automatically generated dialog.

- `getG2AttributeEditor`

  Returns the `com.gensym.gcg.G2AttributeEditor` for editing the attribute of an item, which encapsulates the group name, label, editor, item, and attribute. You use this method to override any of these features of a single attribute and implicitly call `hookUpEditor`.

- `getAttributeGroupName`

  Returns the group name associated with an attribute or set of attributes of an item. By default, the notes are assigned to the `NOTES_GROUP`, the item configurations are assigned to the `CONFIGURATION_GROUP`, and the rest of the attributes are assigned to the `ATTRIBUTES_GROUP`. You use this method to control the groups to which attributes and editors are assigned.

- `getAttributeEditor`

  Returns the `AttributeEditor` to use for editing a single attribute. For example, you might want your dialogs to use a dropdown choice rather than a check box for editing boolean attributes.

- `getAttributeLabel`

  Returns the `com.gensym.gcg.AttributeLabel`, which is the text label associated with the attribute name for a given attribute or set of attributes, in a given locale. You override this method to localize dialog labels and override the default formatting of attribute labels.

In addition to these methods, `DefaultGeneratedDialogFactory` defines these methods:

- `hookUpEditor`

  Provides the necessary event hookups between an `AttributeEditor` and an `ItemProxy` for a particular `AttributeEditor`, and lets you set whether the attribute is read-only. This method is generally not designed to be overridden. Call this method if you are building a dialog from the top down.

- `getGroupNames`

  Returns a list of names of each attribute group. By default, the dialog places each attribute group on its own tab page.

- `generateCommandButtonPanel`

  Creates a `java.awt.Container` for the command button panel area of the automatically generated dialog. By default, the panel is empty. When `autoUpload` on an `ItemProxy` is `false`, the panel contains OK, Apply, and Cancel buttons for accepting the edits and closing the dialog. When constructing a dialog from scratch, call this method on the superior class

to create the default behavior of the command button panel and place it inside your own type of container.

Dialog Components

An automatically generated dialog provides access to these dialog elements, which you can customize or use to construct your own dialog:

- `com.gensym.gcg.AttributeInfo`

  An object that encapsulates all the information about a particular attribute, such as its name, type, defining class, whether it is system defined, and so on. This information is useful in determining whether to include an attribute on the dialog or what component to use to edit the attribute.

- `com.gensym.gcg.G2AttributeGroup`

  A logical grouping of attribute editors, which you can use to organize the attributes and editors in the dialog. Each group has a name. For example, the default implementation creates a tab page for each G2 attribute group.

- `com.gensym.gcg.G2AttributeEditor`

  An object that encapsulates the group name, label, editor, item, and attribute name. You can override one or more of these features for item attributes.

- `com.gensym.controls.AttributeEditor`

  An interface that describes the editor to use to edit the attribute of an item. The editor handles events of type `ObjectUpdateEvent`.

- `com.gensym.controls.FieldType`

  A class that handles the conversion of G2 data types to Java, and vice versa. You can obtain the type from any item attribute, then set the field type of any control in the `com.gensym.controls` package.

- `com.gensym.gcg.AttributeLabel`

  The label associated with an item attribute. By default, the label is the attribute name, using spaces in place of hyphens, initial capitalization, followed by a colon.

- `com.gensym.gcg.G2ReadOnlyTextArea`

  A `JScrollPane` used for displaying complex attributes such as sequences and structures or attributes with a grammar or attributes that display items.

- `com.gensym.gcg.SubDialogLauncher`

  Launches a subdialog for editing an attribute of an item that is an object.

- com.gensym.gcg.G2TextArea

  A G2ReadOnlyTextArea used for editing attributes with a grammar and subobjects. G2ReadOnlyTextArea implements com.gensym.gcg.SubDialogLauncher. If the attribute is a text value with a grammar, calling launchSubDialog launches the TW2 Toolkit text editor. If the attribute is an object, clicking the text area launches a properties dialog for editing the attributes of the subobject.

- com.gensym.gcg.G2ColorField

  A com.gensym.controls.G2TextField whose background is a color, used for editing color attributes. Clicking the text field launches a color dialog for editing the color.

Registering the Generated Dialog Factory

To override the default automatically generated dialogs, you must create a factory and set the factory in the DialogManager. You set the factory once for the dialog manager associated with a connection. To set the factory, call setGeneratedDialogFactory, which takes an implementation of the GeneratedDialogFactory interface as its argument.

This line of code registers the generated dialog factory to use the BooleanEditorFactory shown in "Overriding the Editor for Attributes of a Given Type" on page 325.

```
DialogManager().setGeneratedDialogFactory
    (new com.gensym.demos.gcg.BooleanEditorFactory());
```

Overriding the Editor for Attributes of a Given Type

To override the editor that the automatically generated dialog uses for editing attributes of a particular type, subclass DefaultGeneratedDialogFactory and override the getAttributeEditor method to return the AttributeEditor to use when editing attributes of a given type. All controls in the com.gensym.controls package implement the AttributeEditor interface.

The signature of this method is:

```
protected AttributeEditor getAttributeEditor(TwAccess connection,
                                             ItemProxy itemProxy,
                                             DialogCommand dlgCommand,
                                             AttributeInfo info,
                                             Locale locale)
                                    throws com.gensym.jgi.G2AccessException
```

To determine the attribute type, call `getType` on the `AttributeInfo` argument to `getAttributeEditor` and set the return value to a `com.gensym.controls.FieldType`. To set the attribute type for the editor, call `setFieldType` on any control in the `com.gensym.controls` package.

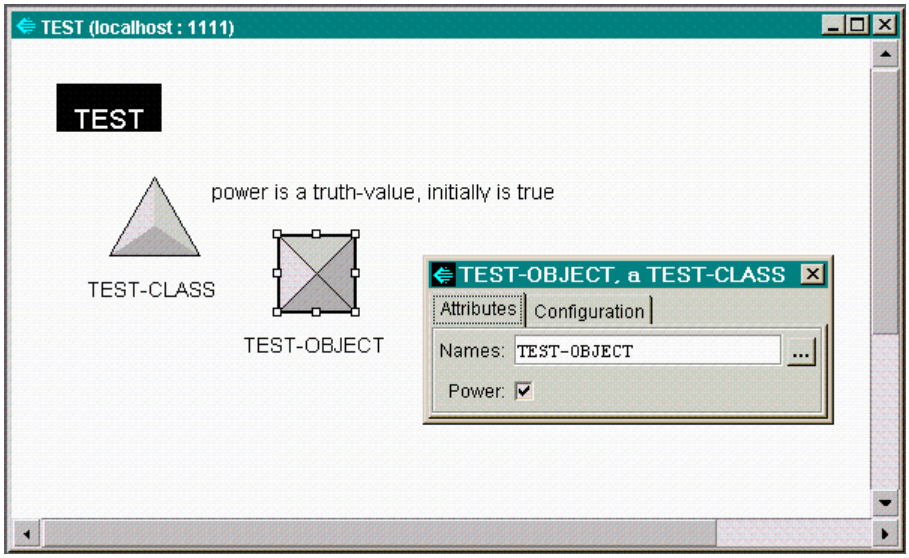
If you create your own `AttributeEditor`, you can explicitly call methods on the `DialogCommand` argument to apply and close the dialog. If you are using one of the editors in the `com.gensym.controls` package, this is not necessary.

When overriding the attribute editor, you must explicitly call `hookUpEditor` to create the necessary event hookups. You can get the attribute name and read-only status by calling methods on the `AttributeInfo` as well.

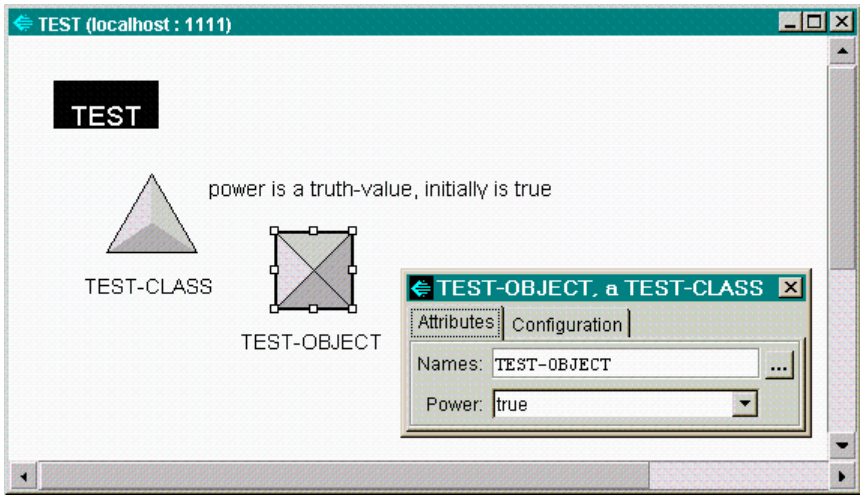
This factory uses a `com.gensym.jcontrols.G2ComboBox` instead of a `G2Checkbox` for boolean attributes; otherwise, the factory uses the default attribute editor:

```
public class BooleanEditorFactory extends
      DefaultGeneratedDialogFactory{

   protected AttributeEditor getAttributeEditor
       (TwAccess connection, ItemProxy itemProxy,
        DialogCommand dlgCommand, AttributeInfo info, Locale locale)
       throws G2AccessException{

       //Test for boolean attribute types
       if (info.getType() instanceof BooleanTruthValueType){

          //Create combo box
          G2ComboBox comboBox = new G2ComboBox();

          //Create and set FieldType
          FieldType fieldType = new FieldType(info.getType());
          comboBox.setFieldType(fieldType);

          //Create and set choices
          StringVector choices = new StringVector();
          choices.addElement("true");
          choices.addElement("false");
          comboBox.setChoices(choices);

          //Handle events
          hookUpEditor((AttributeEditor)comboBox, itemProxy,
                       info.getAttributeName(),
                       info.isValueWritable());
          return comboBox;
       }
       else

          //Return default editor for all other attribute types
          return super.getAttributeEditor(connection, itemProxy,
                                          dlgCommand, info, locale);
   }
}
```

The following example shows the default automatically generated dialog for an object with a truth-value attribute named power, and the dialog that gets generated when you set the `BooleanEditorFactory` with the default application shell, `Shell.java`:



Default



`BooleanDialogFactory`

Localizing Attribute Labels

To localize the attribute labels in an automatically generated dialog, subclass `DefaultGeneratedDialogFactory` and override the `getAttributeLabel` method to return the `AttributeLabel` to use for each attribute editor in the dialog.

The signature for this method is:

```
protected Component getAttributeLabel(TwAccess connection,
                                      ItemProxy itemProxy,
                                      DialogCommand dlgCommand,
                                      AttributeInfo info,
                                      Locale locale)
                               throws com.gensym.jgi.G2AccessException
```

By default, this method returns an `AttributeLabel`, which you must create. Here is the constructor:

```
AttributeLabel(TwAccess connection,
               com.gensym.classes.Item item,
               com.gensym.util.Symbol attribute,
               boolean live)
         throws com.gensym.jgi.G2AccessException
```

You pass the `Item`, which you get by calling `getProxy` on the `ItemProxy` argument to `getAttributeLabel`, and the attribute name, which you get by calling `getAttributeName` on the `AttributeInfo` argument. You also pass in a `boolean` to determine whether the attribute label is automatically updated, which you can obtain by calling `getAutoUpload` on the `ItemProxy` argument.

You call `setText` on the `AttributeLabel` and pass in the label text to display in the dialog.

You define the resource by calling `getBundle` on the `com.gensym.messages.Resource` class, passing in a string that names the properties file.

The following factory uses the attribute name as a key for localizing the label text, using a resource, then uses the translated name as the label. The factory uses the default label for the `NOTES_` and `ITEM_CONFIGURATION_` attributes.

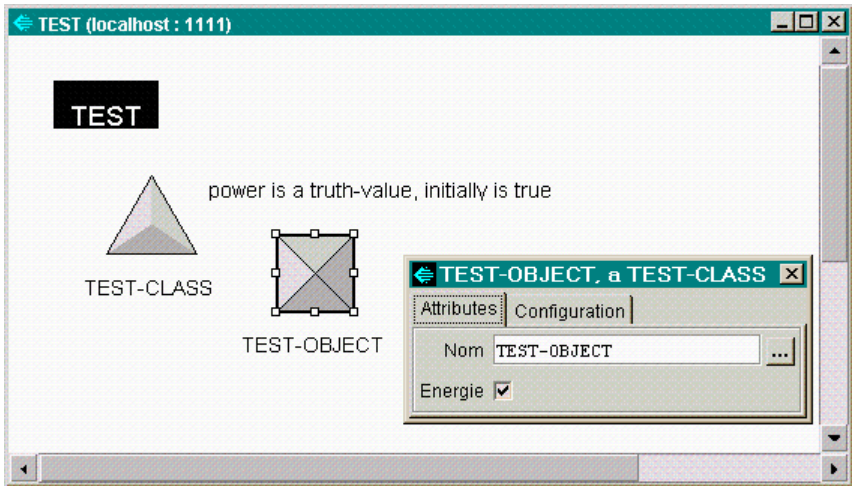
```
public class TranslatedLabelsFactory extends
   DefaultGeneratedDialogFactory{

   //Create resource
   private static final Resource i18nAttributeLabels =
      Resource.getBundle("com.gensym.demos.gcg.AttributeLabels");

   protected Component getAttributeLabel(TwAccess connection,
                                         ItemProxy itemProxy,
                                         DialogCommand dlgCommand,
                                         AttributeInfo info,
                                         Locale locale)
```

```
        throws G2AccessException{
            AttributeLabel label = null;

            //Get attribute name
            Symbol attributeName = info.getAttributeName();

            //Set the label text to the localized text string
            //for all attributes other than notes and item configuration
            if (!attributeName.equals(SystemAttributeSymbols.NOTES_) &&
                !attributeName.equals
                    (SystemAttributeSymbols.ITEM_CONFIGURATION_)){
                    label = new AttributeLabel(connection,
                                               itemProxy.getProxy(),
                                               attributeName,
                                               itemProxy.getAutoUpload());
                    label.setText(i18nAttributeLabels.getString
                        (attributeName.getPrintValue()));
            }
            return label;
        }
    }
```

The properties file named `AttributeLabels.properties` defines translated labels for an object with attributes named Names and Power, as follows:

```
NAMES=Nom
POWER=Energie
```

The following example shows the dialog that gets generated when you set the `TranslatedLabelsFactory` with the default application shell, `Shell.java`:



`TranslatedLabelsFactory`

Creating Tabs for Groups of Attributes

By default, automatically generated dialogs create three tabs: Attributes, Notes, and Item Configurations. If the item contains no notes, the dialog contains no notes tab.

To create different tabs for groups of attributes, you subclass `DefaultGeneratedDialogFactory` and override the `getG2AttributeEditor` method to return the `G2AttributeEditor` to use for each attribute of an item.

The signature for this method is:

```
protected G2AttributeEditor getG2AttributeEditor
    (TwAccess connection,
     ItemProxy itemProxy,
     DialogCommand dlgCommand,
     AttributeInfo info,
     Locale locale)
throws com.gensym.jgi.G2AccessException
```

The `G2AttributeEditor` contains the group, label, editor, item proxy, and attribute for each item attribute, any of which you can override. To override the groups that the dialog uses to create tab pages in the dialog for its attributes, create a new `G2AttributeEditor`, passing in each of these elements. Here is the constructor:

```
public G2AttributeEditor(String groupName,
                         Component label,
                         AttributeEditor editor,
                         ItemProxy itemProxy,
                         com.gensym.util.Symbol attribute)
```

To override the group name for each `G2AttributeEditor`, pass in a string to use for the group name. The following factory uses the name of the defining class as the group name, which it determines by calling `getDefiningClass` on the `AttributeInfo` argument to `getG2AttributeEditor`.

To use the default label and editor, the factory calls `getLabel` and `getEditor` on the default `G2AttributeEditor`. To use the default attribute name, the factory calls `getAttributeName` on the `AttributeInfo` argument.

Once the new `G2AttributeGroup` has been defined, you can override the group names for the dialog by overriding the `getGroupNames` method, which returns a list of strings that define the groups.

The signature for this method is:

```
public String[] getGroupNames(TwAccess connection,
                              ItemProxy itemProxy,
                              Locale locale)
```

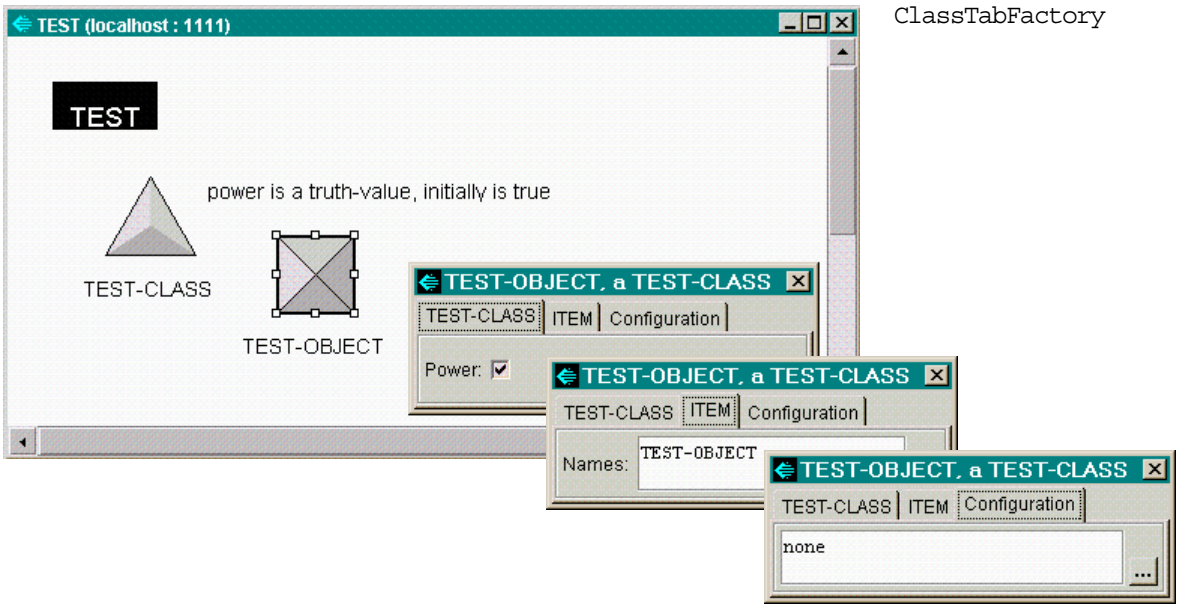
The factory uses the class inheritance path, which it gets from the `ItemProxy` argument to `getGroupNames` to determine the list of groups. It then generates a list

of strings from this list and adds two additional groups for configuration and notes.

```
public class ClassTabFactory extends DefaultGeneratedDialogFactory{

  //Override the group name, label, editor, item, and attribute
  protected G2AttributeEditor getG2AttributeEditor
    (TwAccess connection, ItemProxy itemProxy,
     DialogCommand dlgCommand, AttributeInfo info, Locale locale)
    throws G2AccessException{

      //Get default G2 attribute editor
      G2AttributeEditor g2Editor =
        super.getG2AttributeEditor(connection,itemProxy,dlgCommand,
                                   info,locale);

      //Return default G2AttributeEditor for Notes and
      //Item Configuration attributes
      if
        (info.getAttributeName().
           equals(SystemAttributeSymbols.NOTES_) ||
         info.getAttributeName().
           equals(SystemAttributeSymbols.ITEM_CONFIGURATION_))
         return g2Editor;

      //For all other attributes, override the group name to be
      //the defining class of each attribute; use default label,
      //editor, and attribute
      else
        return new G2AttributeEditor
          (info.getDefiningClass().getPrintValue(),
           g2Editor.getLabel(), g2Editor.getEditor(), itemProxy,
           info.getAttributeName());
}

//Override the group names
public String[] getGroupNames(TwAccess connection,
  ItemProxy itemProxy, Locale locale){
  try{

    //Get inheritance path from item proxy
    Sequence inheritancePath = itemProxy.getProxy().
      getDefinition().getClassInheritancePath();

    //Bind inheritancePath size, plus create two
    //additional groups for Configuration and Notes
    String[] groups = new String[inheritancePath.size()+2];
    for (int i=0; i<inheritancePath.size(); i++)
      groups[i] = inheritancePath.elementAt(i).toString();
    groups[inheritancePath.size()]="Configuration";
    groups[inheritancePath.size()+1]="Notes";
    return groups;
  }
  catch(G2AccessException ex){
    ex.printStackTrace();
```

```
        }
        return super.getGroupNames(connection, itemProxy, locale);
      }
    }
```

The following example shows each tab page of the dialog that gets generated when you set the `ClassTabFactory` with the default application shell, `Shell.java`:



`ClassTabFactory`

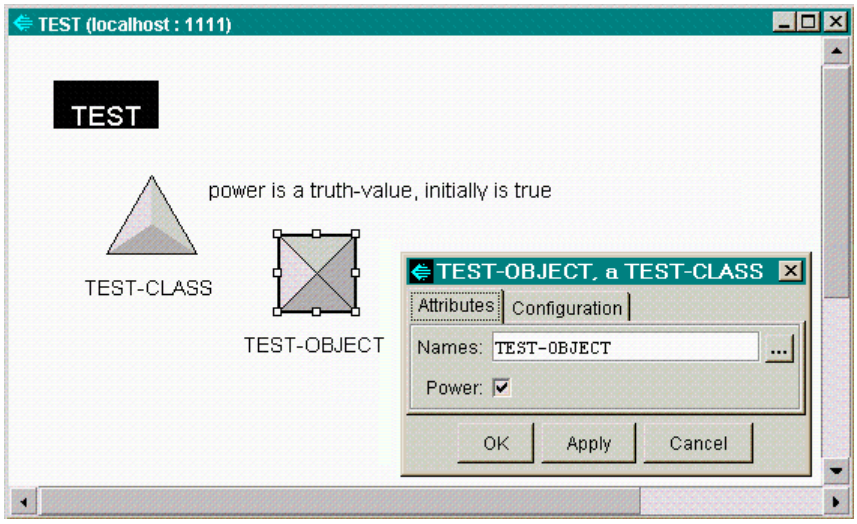
Adding Buttons to Automatically Generated Dialogs

By default, automatically generated dialogs do not include OK, Apply, and Cancel buttons for accepting the edits and closing the dialog. This is because, by default, `autoUpload` is `true` for the `ItemProxy` component, which means that edits to controls linked to an item attribute are automatically uploaded to the G2 item whenever the edits occur.

You might want to set `autoUpload` to `false` and provide OK, Apply, and Cancel buttons on the dialog instead. To do this, set `autoUpload` to `false` and override the `generateDialog` method on the `DefaultGeneratedDialogFactory` subclass, as this factory shows:

```
public class CommandButtonsFactory extends
    DefaultGeneratedDialogFactory{

    public Component generateDialog(Frame frame,
                                    TwAccess connection,
                                    ItemProxy itemProxy,
                                    DialogCommand dlgCommand,
                                    Locale locale)
        throws G2AccessException{

            //Set autoUpload to false
            itemProxy.setAutoUpload(false);

            //Generate the dialog, which automatically creates buttons
            Component component = super.generateDialog(frame,
                                                       connection,
                                                       itemProxy,
                                                       dlgCommand,
                                                       locale);

            return component;
        }
}
```

This example shows each tab page of the dialog that gets generated when you set the `CommandButtonsFactory` with the default application shell, `Shell.java`:



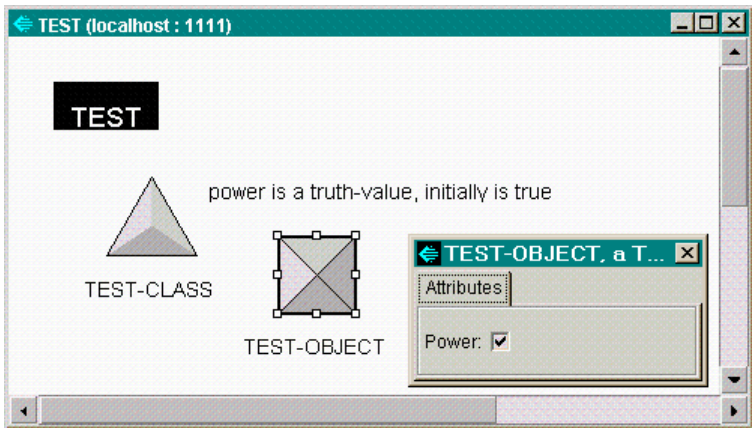
`CommandButtonsFactory`

Creating a Dialog with User-Defined Attributes Only

By default, the automatically generated dialog contains both system-defined and user-defined attributes. To restrict the dialog to display user-defined attributes only, override `getG2AttributeEditor` to return `null` if the `AttributeInfo` is system-defined, which you check by calling `isSystemDefined`. Otherwise, the method returns the default `G2AttributeEditor`, as this factory shows:

```
public class UserDefinedAttributesFactory extends
   DefaultGeneratedDialogFactory{

   protected G2AttributeEditor
      getG2AttributeEditor(TwAccess connection, ItemProxy itemProxy,
                           DialogCommand dlgCommand,
                           AttributeInfo info, Locale locale)
      throws G2AccessException{
         G2AttributeEditor g2Editor =
            super.getG2AttributeEditor(connection, itemProxy,
                                       dlgCommand, info, locale);

         //Return null if AttributeInfo is system-defined
         if (info.isSystemDefined())
            return null;
         //Otherwise, return default G2AttributeEditor
         else
            return new G2AttributeEditor(g2Editor.getGroupName(),
                                         g2Editor.getLabel(),
                                         g2Editor.getEditor(),
                                         itemProxy,
                                         info.getAttributeName());
   }
}
```

This example shows the dialog that gets generated when you set the `UserDefinedAttributesFactory` with the default application shell, `Shell.java`:



`UserDefinedAttributesFactory`

18

Launching General Dialogs

Describes how to implement your own dialog launcher and dialog reader, and how to create your own dialog resource for dialog resources saved in a format other than a serialized file or a Java class file.

Introduction

To create general dialogs, you use Java to create a dialog class.

Because the TW2 Toolkit dialog components are packaged as Java Beans, you can create general dialogs in a JavaBeans-compliant visual programming environment, such as Symantec Visual Café or Borland J Builder.

After you create a general dialog that you want to launch from your client application, you use the classes in the `com.gensym.dlgruntime` package to read and launch the dialog.

This chapter describes how to launch general dialogs and how to create your own types of dialog resources.

Relevant Packages

com.gensym.dlgruntime

Interfaces

DialogCommandListener
DialogLauncher
DialogReader

Classes

DefaultDialogLauncher
DefaultDialogReader
DialogClassReader
DialogCommandEvent
DialogManager
DialogResource

Exceptions

ResourceInstantiationException

Relevant Demos

The Java applications that this chapter uses are available online in this directory, depending on your platform:

NT:	`%SEQUOIA_HOME%\classes\com\gensym\demos\docs\launchdialog\*.java`
UNIX:	`$SEQUOIA_HOME/classes/com/gensym/demos/docs/launchdialog/*.java`

The filenames correspond to the class names in each example in this chapter.

Reviewing the Dialog Runtime Interfaces and Classes

These are the interfaces in the `com.gensym.dlgruntime` package that you can use to launch general dialogs:

Interface	Description
`DialogLauncher`	The launching interface.
`DialogReader`	The resource reading interface.
`DialogCommandListener`	The listener interface to handle event notification when the dialog is launched, when dialog edits are flushed, and when the dialog is closed.

These are the classes in the `dlgruntime` package that you use to launch general dialogs:

Class	Description
`DefaultDialogLauncher`	A TW2 Toolkit implementation of the `DialogLauncher` interface, which launches the output of an implementation of `DialogReader`.
`DefaultDialogReader`	A TW2 Toolkit implementation of the `DialogReader` interface, which reads serialized dialog resource (`.ser`) files.
`DialogClassReader`	An implementation of `DialogReader` for reading dialog classes that are saved as Java classes. For details, see "Registering Custom Item Properties Dialog Classes" on page 316.
`DialogResource`	The dialog resource object and its associated launcher and reader.

Class	Description
DialogManager	Registers dialog resources for use as custom item properties dialogs. Depending on the dialog registration method that your application calls, the DialogManager can use: • The DefaultDialogLauncher and DefaultDialogReader to launch and read serialized dialog resources or dialog classes created in Java. • A DialogResource that you create, which provides its own implementation of a DialogReader and/or DialogLauncher. For details on using this class, see Chapter 16, "Launching Custom Item Properties Dialogs" on page 305.
DialogCommandEvent	The event that a com.gensym.controls.DialogCommand generates when a dialog is launched, flushed, or closed. The DialogLauncher calls the open method on a DialogCommand, which generates this event.

Telewindows2 (TW2) Toolkit supplies the DialogLauncher and DialogReader interfaces, and the DialogResource class for developers who wish to design and create their own dialog launchers and readers, as described in "Creating Your Own Types of Dialog Resources" on page 345.

Launching General Dialogs from Your Application

These are the steps for launching a dialog resource once it exists:

- Create a new dialog reader and launcher.
- Create a resource object.
- Get the ItemProxy components from the dialog resource.
- Ask the reader to create the top-level component from the resource.
- Launch the dialog, passing in as arguments the component, the ItemProxy, and the item for which the dialog is to be launched.

Typically, you would launch a general dialog for a particular item in response to some application event. For example, your application might be a com.gensym.

util.ItemListener so it receives notification when an attribute value of an item has exceeded a certain threshold. In response to that event, the application could launch a general dialog for that item.

Alternatively, you might want to launch a general informational dialog about the currently selected item in a workspace view. To do this, you would get the current selection from the workspace view, check if a single item is selected, get the item from the workspace element, and launch the dialog for that item.

You can also launch general dialogs that display information about multiple G2 items. You do this by naming each ItemProxy component in the dialog and passing in an array of names, along with an array of ItemProxy components, and an array of associated items.

If your dialog resource contains any instances of the com.gensym.controls.DialogCommand component for applying dialog changes and closing the dialog, your application must also get the DialogCommand and pass this component as an argument when you launch the dialog.

This section uses a simple example that launches a dialog created for the material-source item named warehouse in the mill.kb, which is shipped in the kbs subdirectory of your TW2 Toolkit product directory. The dialog resource is stored in a serialized file.

Creating a Default Dialog Reader and Launcher

Assuming you want to use the DefaultDialogReader and the DefaultDialogLauncher to manage your dialog resource, your application must import these classes from the com.gensym.dlgruntime package. If you have implemented your own dialog reader and launcher, you would create instances of those classes instead.

The DefaultDialogLauncher notifies registered listeners of DialogCommandEvents when the dialog is launched, flushed, or closed.

To create a default dialog reader and launcher:

```
// Create dialog reader and launcher
private DialogReader reader = new DefaultDialogReader();
private DialogLauncher launcher = new DefaultDialogLauncher();
```

Creating a Resource from a Dialog Resource File

The DefaultDialogReader class provides the makeResource method, which creates a dialog resource from a specified URL. You read the dialog resource by calling the readResource method on the dialog reader.

Because the argument you pass to the makeResource method is a java.net.URL, you can also provide a URL to a G2 media bin in which the dialog resource is stored.

When making and reading a resource, you must catch these exceptions:

- `makeResource` throws `com.gensym.dlgruntime.ResourceInstantiationException`
- `readResource` throws `java.io.IOExcpetion`

This example creates a resource from a resource file stored on the client.

To create a resource from a dialog resource file:

```
try {
   // Make the dialog resource object
   Object resource = reader.makeResource (reader.readResource
       (new java.net.URL ("file:///c:/gensym/Telewindows2Toolkit/
          Java/resources/LaunchDialogExample.ser")));
}
```

To point to a dialog resource stored in a media bin:

➔ Point to the item with a string such as:

"g2://*host:port*//namedBin/*media-bin-name/dialog.ser*"

Element	Description
host:port	The host and port specification for where the G2 is running, for example: `localhost:1111`
media-bin-name	The name of the media-bin item in which the dialog resource is stored. For example: `DIALOG-MEDIA-BIN`
dialog-ser	The name of the serialized dialog file, for example: `MillDiaolg.ser`

Thus, the string would be:

```
"g2://localhost:1111//namedBin/DIALOG-MEDIA-BIN/MillDialog.ser"
```

Getting the ItemProxy Components from the Resource

Before launching the dialog resource, you need to get the `ItemProxy` components that are stored in the resource file, as an array. You will pass this array to the method that launches the dialog resource later in your code. You get the `ItemProxy` components from the `DefaultDialogReader`.

To get the ItemProxy components from the resource:

```
//Get item proxies from the resource
ItemProxy[] proxies = reader.getItemProxies(resource);
```

Creating the Top-Level Component from the Resource

Once you have created the resource, you must create the top-level component in which to display the dialog.

To do this, you call `getComponent` on the `DefaultDialogReader`, which returns different values, depending on how you created the dialog resource. If you use the `DefaultDialogLauncher` to launch the dialog, the dialog gets launched differently, depending on the return value, as follows:

If you create the dialog resource in...	Then `getComponent` returns...	And the `DefaultDialogLauncher`...
A JavaBeans-compliant visual programming environment	An instance of a `java.awt.Panel`	Places the `Panel` component inside of a `java.awt.Dialog`.
Java, and place it in any kind of a `java.awt.Window` component, such as a `Dialog`, `Frame`, `JDialog`, or `JFrame`	The `Window` component	Calls `setVisible` on the `Window`, which launches the component in the center of the parent frame.

To get the dialog component from the resource:

```
//Get component for the resource
Component dlg = reader.getComponent(resource);
```

Launching the Dialog

To launch the dialog, you can call the `launch` method on the `DefaultDialogLauncher`, providing these arguments:

- The dialog component to launch.
- An `ItemProxy`.
- An `Item` that is the stub for the `ItemProxy` component.
- A `DialogCommand` component, if present in your dialog resource.

If your dialog contains multiple `ItemProxy` components, you can call the alternative version of the `launch` method, which takes an array of `ItemProxy`

components, an array of names for each `ItemProxy`, which you specify using the `name` property, and an array of `Item` stubs. For details, see the API documentation.

The following example simply launches a dialog in a frame; it does not launch it for a particular item.

Typically, you would pass in the item for which the dialog is to be launched, which requires making a connection and getting the unique named item from the connection. Your application would then listen for a particular event and launch the dialog for the item when the event occurs. For information on creating connections, getting items from connections, and listening for events, see Part II, "Connecting to G2" on page 23.

In the following example, the dialog resource contains a single `ItemProxy` and no `DialogCommand` components.

To launch the dialog resource:

```
//Launch it!
launcher.launch (dlg, proxies[0], null, null);
```

Example Code

Following is the complete code for the example described in the previous sections:

```
package com.gensym.demos.docs.launchdialog;

import java.awt.Component;
import java.awt.Frame;
import java.awt.Window;
import java.awt.Dialog;
import java.awt.event.*;
import java.net.URL;
import com.gensym.dlgruntime.*;
import com.gensym.controls.ItemProxy;

public class LaunchDialog {

  // Create default reader and launcher
  private DefaultDialogReader reader = new DefaultDialogReader ();
  private DefaultDialogLauncher launcher = new DefaultDialogLauncher();

  private static final String FILE =
    "file:///c:/gensym/Telewindows2Toolkit/classes/com/gensym/
    demos/docs/launchdialog/LaunchDialogExample.ser";
```

```
private void showDialog(Frame f) {
    try{
      // Make the dialog resource object
      Object resource = reader.makeResource (reader.readResource
          (new URL (FILE)));
      //Get item proxies from the resource
      ItemProxy[] proxies = reader.getItemProxies(resource);
      //Get component for the resource
      Component dlg = reader.getComponent(resource);
      //Register the frame
      launcher.registerFrame(f);
      //Launch it!
      launcher.launch (dlg, proxies[0], null, null);
    } catch (Exception e){
      e.printStackTrace();
    }
  }

  public static void main (String args[]) {
    //Create an adapated frame so we can close it
    LaunchDialog d = new LaunchDialog();
    Frame f = new Frame("Launch Dialog Example");
    WindowAdapter wa =
      new WindowAdapter() {
      public void windowClosing (WindowEvent e) {
   System.exit(0);
      }
    };
    f.addWindowListener(wa);
    f.setSize(600,400);
    f.show();
    d.showDialog(f);
  }
}
```

Creating Your Own Types of Dialog Resources

The `com.gensym.dlgruntime` package provides the `DialogResource` class for creating your own type of dialog resource. A `DialogResource` takes as its arguments:

- The dialog resource as a URL.
- An implementation of the `DialogReader` interface.
- An implementation of the `DialogLauncher` interface.

You use a dialog resource to launch both custom item properties dialogs, as well as general dialogs, as follows:

To launch...	Do this...
A custom item properties dialog	Use the version of the `setDialogResourceEntry` method on the `DialogManager` that takes an instance of a `DialogResource`, rather than a serialized dialog resource file or a Java dialog class, as described in Chapter 16, "Launching Custom Item Properties Dialogs" on page 305. See the API for `DialogManager` for details.
A general dialog	Provide the `DialogResource` directly to the `launch` method on your dialog launcher.

When to Create Your Own Dialog Resource

This table describes when you need to implement your own dialog reader and launcher, and when you need to create your own dialog resource:

You do this...	When you want to...
Implement the `DialogReader` interface	Create dialog resources in serialized (`.ser`) files or any other format, such as XML.
Implement the `DialogLauncher` interface	Launch a dialog resource inside a container other than a `java.awt.Dialog`. For example, you might want to launch a dialog inside a Java applet or frame.
Create your own `DialogResource`	Implement your own `DialogReader` or `DialogLauncher`. If you do not implement both of these interfaces, you can use one of the default implementations as either the launcher or reader for the dialog resource.

If you do not wish to implement your own reader and launcher, but you wish to customize the behavior of the default reader and launcher, you may also subclass the `DefaultDialogReader` and `DefaultDialogLauncher` classes.

Launching a Custom Dialog Resource

The steps for launching a `DialogResource` that you create are similar to those you use to launch a dialog resource:

- Implement the `DialogReader` and/or `DialogLauncher` interfaces, as needed.
- Create a new dialog reader and launcher, using your implementations of `DialogReader` and/or `DialogLauncher`.
- Create a resource object from a URL.
- Get the `ItemProxy` components from your dialog resource.
- Get the top-level component that contains your dialog resource from the resource.
- Create an instance of a `DialogReader`, passing in the dialog resource, and your implementations of `DialogReader` and/or `DialogLauncher`.
- Launch the dialog by calling the `launch` method on your implementation of `DialogLauncher`, passing in an instance of your `DialogResource` as the resource.

The following example shows how to create and launch a custom dialog resource by subclassing `DefaultDialogReader` and `DefaultDialogLauncher`. You provide your customizations in the `CustomDialogReader` and `CustomDialogLauncher` inner classes. You could also implement your own `DialogReader` and `DialogLauncher` interfaces. This example behaves exactly the same as the previous example.

```
package com.gensym.demos.docs.launchdialog;

import java.awt.Component;
import java.awt.Frame;
import java.awt.Window;
import java.awt.Dialog;
import java.awt.event.*;
import java.net.URL;

import com.gensym.dlgruntime.*;
import com.gensym.controls.ItemProxy;

public class LaunchDialogResource {

  // Create custom reader and launcher
  private CustomDialogReader reader = new CustomDialogReader ();
  private CustomDialogLauncher launcher = new CustomDialogLauncher();

  // Point to the dialog resource
  private static final String FILE =
    "file:///c:/gensym/Telewindows2Toolkit/classes/com/gensym/
    demos/docs/launchdialog/LaunchDialogExample.ser";
```

```
  //Inner classes
  class CustomDialogReader extends DefaultDialogReader {
    //Provide customizations here or implement DialogReader
  }

  class CustomDialogLauncher extends DefaultDialogLauncher {
    //Provide customizations here or implement DialogLauncher
  }

  private void showDialog(Frame f) {
    try{
      // Make the dialog resource object
      Object resource = reader.makeResource (reader.readResource
          (new URL(FILE)));
      //Create DialogResource
      DialogResource dialogResource = new DialogResource
          (resource, reader, launcher);
      //Get item proxies from the resource
      ItemProxy[] proxies = reader.getItemProxies(resource);
      //Get component for the resource
      Component dlg = reader.getComponent(resource);
      //Register the frame
      launcher.registerFrame(f);
      //Launch it!
      launcher.launch (dlg, proxies[0], null, null);
    } catch (Exception e){
      e.printStackTrace();
    }
  }

  public static void main (String args[]) {
    //Create an adapated frame so we can close it
    LaunchDialogResource d = new LaunchDialogResource();
    Frame f = new Frame("Launch Dialog Example");
    WindowAdapter wa =
      new WindowAdapter() {
      public void windowClosing (WindowEvent e) {
   System.exit(0);
      }
    };
    f.addWindowListener(wa);
    f.setSize(600,400);
    f.show();
    d.showDialog(f);
  }
}
```

Part V

Appendixes, Glossary, and Index

A

Restricted Remote Procedure Calls

These are the restricted RPC calls that require the existence of a login session:

```
g2-add-trend-chart-component
g2-change-mode-for-window
g2-clear-parsing-context
g2-commit-parse-result
g2-create-parsing-context
g2-delete-parsing-context
g2-delete-trend-chart-component
g2-fire-action-button
g2-fire-user-menu-choice
g2-get-containment-hierarchy
g2-get-current-user-menu-choices
g2-get-user-menu-choice
g2-make-ui-client-session
g2-menu-of-names-for-category
g2-release-ui-client-session
g2-represent-trend-chart
g2-represent-workspace
g2-shift-cursor-position
g2-subscribe-to-modules
g2-unrepresent-workspace
g2-unsubscribe-to-modules
g2-update-parsing-context
```

Compatibility Issues

The following table describes the features of G2 that the Telewindows2 Toolkit components do not support. The details list the item configurations that correspond to the feature.

Note If you edit the item configuration of an item in the G2 server, you must redownload the workspace view to see the changes.

G2 Feature	Details
Mouse tracking	constrain moving
Workspace scaling and z-order positioning	All

G2 Feature	Details
Main menu choices	new-title-block neatly-stack-windows network-info system-tables close-telewindows-connection log-out change-password reinstall-authorized-users connect-to-foreign-image disconnect-from-foreign-image load-attribute-file clear-kb shut-down-g2 enter-package-preparation-mode leave-package-preparation-mode strip-text-now make-workspaces-proprietary-now flush-change-log-for-entire-kb enter-simulate-proprietary-mode leave-simulate-proprietary-mode do-not-highlight-invoked-rules highlight-invoked-rules enable-all-items remove-tracing-and-breakpoints launch-online-help refresh long-menus short-menus run-options inspect view-change-log
Workspaces	get-workspace print
KB	load-merge-save load-kb merge-kb save-kb change-mode
Modules	create-new-module delete-module

G2 Feature	Details
Item menu choices	describe-configuration move edit-icon
Chart items	edit-cell-expression edit-cell-format other-edits
Free form tables	edit-cell-expression edit-cell-format other-edits
Obsolete features	clone-as-trend-chart change-min-size table-of-values change-size main-menu operate-on-area miscellany

Glossary

A

aggregate control: A dialog control for editing data structures, such as G2 lists, sequences, and structures. You must use one of the scalar controls to edit subparts of the data structure. *Contrast with* scalar control. *See also* data-aware component.

C

collection: The list of KB workspaces that a multiple workspace panel maintains. *See also* selection.

component developer: A Telewindows2 (TW2) Toolkit user who uses and extends TW2 Toolkit components and core classes to create applications that access and manipulate KB data and knowledge. *Contrast with* UI developer.

connection information object: An object that defines properties related to creating TW2 Toolkit and G2 JavaLink connections to G2.

connectivity class: A TW2 Toolkit class or component for connecting a Java client to G2.

current workspace: The workspace that is visible in a multiple workspace view or panel. *See* multiple workspace view and multiple workspace panel.

D

data-aware component: A TW2 Toolkit component for viewing and representing attributes of G2 items, which can perform automatic updates to and from G2 and which transparently handle G2 data types. TW2 Toolkit provides two categories of data-aware components: visual controls for representing attributes of G2 items and invisible components for representing G2 items and structures. *See also* aggregate control and scalar control.

dialog class: A dialog that you create from components in any integrated development environment (IDE). You can use dialog classes to create custom item properties dialogs for viewing and editing attributes of G2 items in the client. *Contrast with* dialog resource. *See* item properties dialog and general dialog.

G

G2 application programmer's interface (API): A set of G2 procedures that provide access to G2 data and actions, which permits client applications to subscribe to the state of G2 items and attributes programmatically.

general dialog: An informational dialog for displaying messages to the user or an input dialog for obtaining data from a user. You create general dialogs by using data-aware components in any integrated development environment (IDE). *See also* dialog class and dialog resource.

ghost: An empty outline that appears around an object in a workspace view when the user moves or reshapes it in the client. Only when the user lifts the mouse to complete the operation does the client notify G2 that the object has changed, which minimizes processing and network overhead. *See* workspace view.

I

integrated development environment (IDE): A fully integrated Java development environment, such as Symantec Visual Café or Borland J Builder, which sometimes also includes a JavaBeans-compliant visual programming environment. *See also* JavaBeans-compliant visual programming environment.

item properties dialog: A dialog that you use to edit G2 item attributes. When editing item attributes in the client through a workspace view, by default, you use an automatically generate item properties dialog, which provides native controls for editing typed and untyped item attributes. You can override the automatically generated dialogs by using TW2 Toolkit components in an integrated development environment (IDE) to create custom item properties dialogs, which you must register for G2 items or classes. *See also* dialog class and dialog resource.

J

JavaBeans-compliant visual programming environment: A Java-based programming environment that allows UI developers to create Java applications from Java Beans, which are visual representations of Java components that conform to the Java Beans specification of properties, events, and methods. Developers load JAR files of Java Beans into the visual Java programming environment, edit the properties of those beans through a properties table, and hook up event triggers from one component to target methods in another component. The Telewindows2 Toolkit is an example of a JavaBeans-compliant visual programming environment. *See also* integrated development environment.

K

KB workspace: Instances of the G2 kb-workspace class, a representation of which appears in workspace view. *See* workspace view.

M

middle tier: The two processes that use the Java Remote Method Invocation API, which includes an RMI registry and an RMI server. You can use a middle tier to connect to G2 from a TW2 Toolkit client by specifying a broker URL as the RMI server. By default, TW2 Toolkit client connections use a two-tier configuration. *See* RMI registry and RMI server.

multiple workspace display: A multiple workspace view or a multiple workspace panel. *See* multiple workspace view and multiple workspace panel.

multiple workspace panel: A multiple workspace view that has scrollbars and that listens for programmatic show and hide workspace events in G2. *See* multiple workspace view.

multiple workspace view: A representation of a KB workspace that displays any of several KB workspaces, which can exist in one or more G2 applications. *See* KB workspace and workspace view. *Contrast with* multiple workspace panel.

R

RMI registry: A naming service tool of the Java Remote Method Invocation (RMI) system. RMI servers use the registry to bind remote objects to names. *See also* middle tier and RMI server.

RMI server: A process for handling requests from clients to the server, and from the server to its clients. *See also* middle tier and RMI registry.

S

scalar control: A visual data-aware component that you use to view and edit "atomic" data types, such as integers, floats, text, symbols, and truth values. *Contrast with* aggregate controls. *See* data-aware component.

scroll block: A level or granularity for scrolling workspace views, measured in pixels. *Contrast with* scroll unit.

scroll unit: A level or granularity for scrolling workspace views, measured in pixels. *Contrast with* scroll block.

selection: The list of currently selected items in a single workspace view. Clients can register as listeners for selection events to receive notification when an item is added to removed from the selection. *See* single workspace view and workspace view element. *See also* collection.

single workspace view: A representation of a G2 KB workspace that displays a view of a single KB workspace that exists in a G2 application. A single workspace view that supports scaling of the view along the x and y axes. *Contrast with* multiple workspace view. *See* KB workspace.

standard dialog: A G2 JavaLink class that you can use directly within a Java application or subclass to create an informational dialog or a dialog that accepts user input. *Contrast with* general dialog and item properties dialog.

stub: A G2 JavaLink representation of a G2 item in a client, which remains synchronized without polling G2. The underlying technology that enables the creation of stubs for use in a Java client is the G2 and G2 JavaLink application programmer's interface (API). *See* G2 application programmer's interface (API).

U

UI developer: A TW2 Toolkit user who creates client user interfaces for G2 applications. *Contrast with* component developer.

W

workspace list: The list of KB workspaces that a multiple workspace display maintains. *See* multiple workspace panel and multiple workspace view.

workspace view component: A TW2 Toolkit component that can display a representation of a G2 KB workspace in a Java container, such as a TW2 Toolkit application shell or an applet running in a Web browser. *See also* workspace view, multiple workspace view, and multiple workspace panel.

workspace view element: A representation of a G2 item in a workspace view. When you manipulate workspace view elements in the client, G2 is notified of changes only when the client has completed the edit, which minimizes network traffic.

workspace view: Any display of a G2 KB workspace in a Java container. The term workspace view also refers to any of the three types of workspace view components and an instance of the single workspace view.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W