G2 XL Spreadsheet

User's Guide

Version 2015

G2 XL Spreadsheet User's Guide, Version 2015

December 2015

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

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

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

Part Number: DOC027-1200

Contents Summary

Contents

Preface

Describes this guide and the conventions that it uses.

About this Guide

This guide contains complete information about the G2 Spreadsheet utility (GXL), and shows you how to use the module at any supported level. This guide:

- Introduces GXL and describes the functions, classes and associated capabilities that it provides.
- Provides specific instructions for using GXL.
- Describes the GXL application programmer's interface (API), and shows you how to use it to create and manipulate spreadsheets programmatically.
- Lists all GXL API functions in a reference dictionary.

Audience

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

This guide assumes that you have a general familiarity with spreadsheet systems as seen from the user's viewpoint. It does not assume an understanding of the internal operations of GXL.

Organization

This guide contains fifteen chapters, divided into two parts, and an appendix:

	Title	Description
Part I	**Using GXL**	
1	Introduction to GXL	Describes the features of the G2 XL Spreadsheet utility and the basic concepts of GXL spreadsheets and views.
2	Getting Started	Describes how to install the GXL utility and introduces the features available from the top-level workspace.
3	Using Spreadsheet Views	Describes the parts of a spreadsheet view and user interactions with the view.
4	Using the Formula Bar	Introduces formulas and provides instructions for using formulas to calculate values in GXL spreadsheets.
5	Working with Specifications	Describes how to build and use GXL specifications to create spreadsheets and views.
6	Customizing the Toolbar	Describes how to customize the GXL toolbar display and how to include custom buttons on the toolbar.
7	Programming GXL	Describes the programmatic interfaces to GXL.

	Title	Description
Part II	**API Reference**	
8	Creation and Deletion Operations	Describes the API procedures for creating and deleting GXL spreadsheets and views.
9	Loading and Unloading Data	Describes the API procedures for loading and unloading spreadsheet data.
10	Accessing Spreadsheet and View Properties	Describes the API procedures for accessing GXL spreadsheet and view properties.
11	Setting Spreadsheet and View Properties	Describes the API procedures for setting GXL spreadsheet and view properties.
12	Additional View Procedures	Describes miscellaneous API procedures related to GXL views.
13	Row and Column Operations	Describes the API methods for adding and deleting rows and columns, and API procedures for sorting.
14	Toolbar Procedures	Describes the API procedures for managing the appearance of toolbars on GXL spreadsheet views.
15	Tabular Edit Operations	Describes the API procedures for launching and managing GXL edit sessions.
Part III	**Appendix and Glossary**	
A	GXL Memory Requirements	Describes the memory requirements for GXL.

A Note About the API

The G2 XL Spreadsheet API, as described in this guide, is not expected to change significantly, but there may be exceptions. A detailed description of any changes will accompany the GXL release that includes them.

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

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

Conventions

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

Typographic

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

Convention Examples	Description
File > Save Properties	GMS and native menu choices
workspace	Glossary terms
c:\Program Files\Gensym	Windows pathnames
/usr/gensym/g2/kbs	UNIX pathnames
spreadsh.kb	File names
g2 -kb top.kb	Operating system commands
public void main() *gsi_start*	Java, C and all other external code

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

Procedure Signatures

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

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

Related Documentation

G2 Core Technology

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

- *G2 System Procedures Reference Card*
- *G2 Class Reference Manual*
- *Telewindows User's Guide*
- *G2 Gateway Bridge Developer's Guide*

G2 Utilities

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

G2 Developers' Utilities

- *Business Process Management System Users' Guide*
- *Business Rules Management System User's Guide*
- *G2 Reporting Engine User's Guide*
- *G2 Web User's Guide*
- *G2 Event and Data Processing User's Guide*
- *G2 Run-Time Library User's Guide*
- *G2 Event Manager User's Guide*
- *G2 Dialog Utility User's Guide*
- *G2 Data Source Manager User's Guide*
- *G2 Data Point Manager User's Guide*
- *G2 Engineering Unit Conversion User's Guide*
- *G2 Error Handling Foundation User's Guide*
- *G2 Relation Browser User's Guide*

Bridges and External Systems

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

G2 JavaLink

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

G2 Diagnostic Assistant

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

Customer Support Services

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

To obtain customer support online:

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

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

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

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

➔ Use the following numbers and addresses:

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

Part I

Using GXL

Chapter 1: Introduction to GXL

Describes the features of the G2 XL Spreadsheet utility and the basic concepts of GXL spreadsheets and views.

Chapter 2: Getting Started

Describes how to install the GXL utility and introduces the features available from the top-level workspace.

Chapter 3: Using Spreadsheet Views

Describes the parts of a spreadsheet view and user interactions with the view.

Chapter 4: Using the Formula Bar

Introduces formulas and provides instructions for using formulas to calculate values in GXL spreadsheets.

Chapter 5: Working with Specifications

Describes how to build and use GXL specifications to create spreadsheets and views.

Chapter 6: Customizing the Toolbar

Describes how to customize the GXL toolbar display and how to include custom buttons on the toolbar.

Chapter 7: Programming GXL

Describes the programmatic interfaces to GXL.

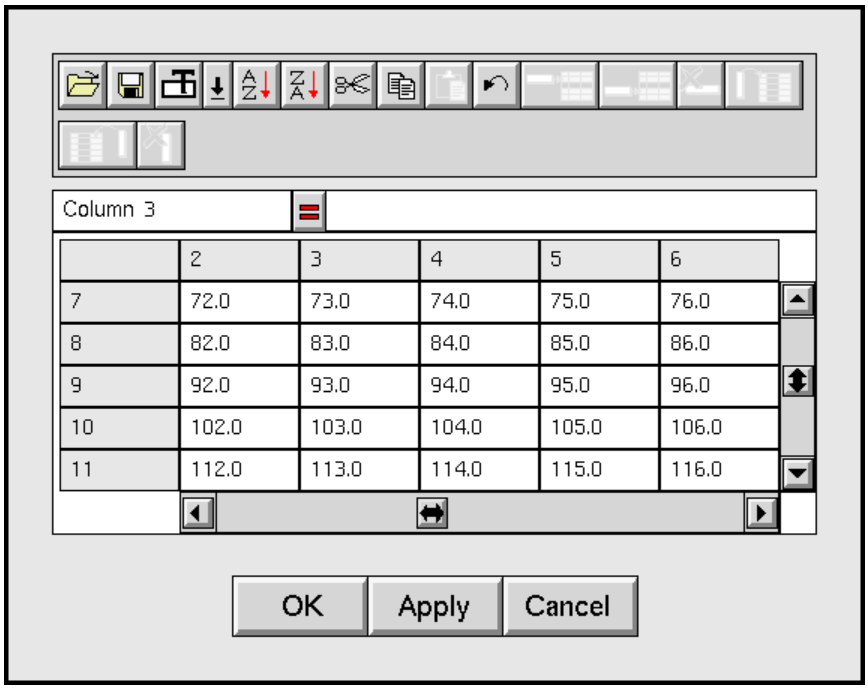
1

Introduction to GXL

Describes the features of the G2 XL Spreadsheet utility and the basic concepts of GXL spreadsheets and views.

What Is GXL?

The G2 XL Spreadsheet, or GXL, is a G2 utility module that allows you to create, display, and edit tabular data in familiar spreadsheet style. The following figure is an example of a view of a GXL spreadsheet.



You can create GXL displays dynamically or save them as permanent parts of your G2 application on any G2 workspace. These displays can accept user input from the mouse or keyboard, and you can update the displays in real time when data changes in your application. You can also dynamically construct GXL views on UIL dialogs, and use them for multiple-column input.

GXL and Telewindows

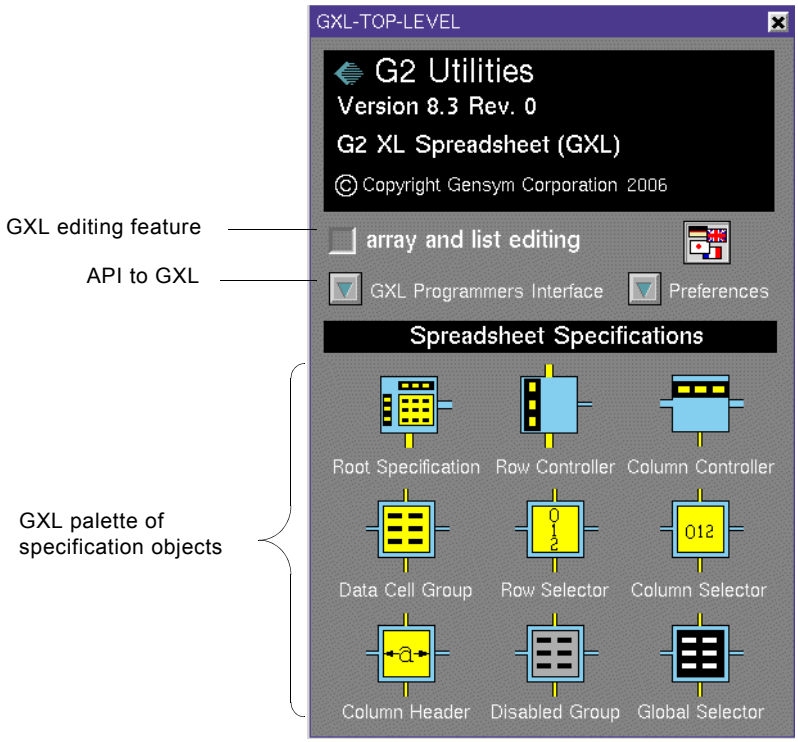
GXL has been designed with Gensym's Telewindows client-server architecture in mind. Several users can view the data in a single spreadsheet simultaneously. Each view of the spreadsheet can be scrolled independently, display different numbers of rows and columns, have different editing privileges, have toolbars containing different tools, and use different local languages.

GXL Features

GXL features include:

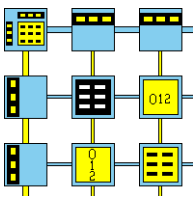
- G2 list and array editing with GXL spreadsheet views.
- Specifications for defining GXL spreadsheets and views.
- Application programmer's interface (API) to GXL.

All GXL features are available from the GXL top-level workspace:



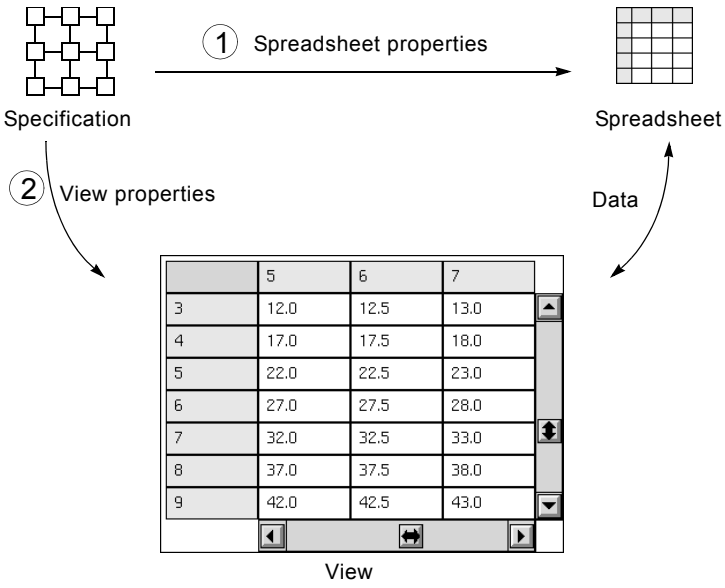
GXL Specifications

A **specification** is a graphical layout of objects that serves as a template for creating GXL spreadsheets and views. The following example illustrates a simple GXL specification:



Based on the layout of specification objects, their class, and their attribute values, a specification defines a GXL spreadsheet and a view of the spreadsheet.

Specification objects define two categories of attributes: those related to the spreadsheet and those related to its view. As the following figure illustrates, GXL first references the spreadsheet properties of the specification to create the spreadsheet, then references the view properties to create a view of the spreadsheet.



You build a specification on a workspace by cloning specification objects from the GXL top-level workspace. For information on creating spreadsheet specifications, see Working with Specifications.

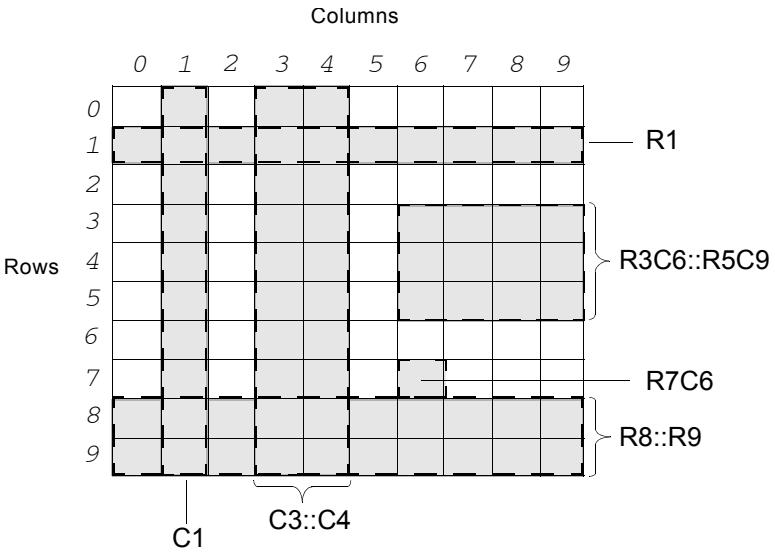
The GXL Spreadsheet

The **GXL spreadsheet**, the central object in GXL, is an instance of the class gxl-spreadsheet. A GXL spreadsheet contains data that is stored as a two-dimensional grid. You can create a GXL spreadsheet manually from a spreadsheet specification or programmatically by calling API procedures.

Spreadsheet Cells

Each data location in the spreadsheet is referred to as a **cell**, where row (R) and column (C) coordinates define the location within the two-dimensional grid. The row and column numbering of GXL spreadsheets always begins at 0. As the following figure illustrates, you can refer to a:

- Single cell (R7C6)
- Single row of cells (R1)
- Single column of cells (C1)
- Range of cells (R3C6::R5C9)
- Range of rows (R8::R9)
- Range of columns (C3::C4)



Every GXL spreadsheet has definite row and column dimensions, and the maximum size is 65535 rows and 65535 columns. Memory requirements are discussed in Appendix A, GXL Memory Requirements.

Cell Data Types

Each cell in a spreadsheet has a specific **data type**, which dictates the valid contents of the cell. Valid data types are:

- Floats
- Integers
- Quantities (floats or integers)

- Symbols
- Text
- Truth-values

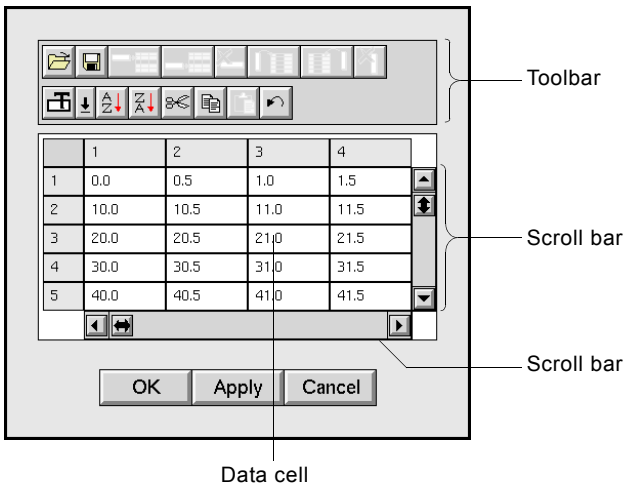
A cell can also be empty and contain no value.

The GXL Spreadsheet View

A **spreadsheet view** is the graphical user interface to the data contained in a spreadsheet. A view displays:

- The data in the cells of the spreadsheet.
- Tools that enable you to interact with the data in the spreadsheet.

For example:

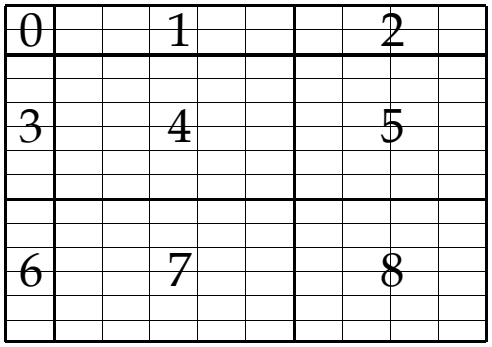


For information on using GXL tools to interact with spreadsheet data, see Using Spreadsheet Views.

You can customize GXL views by specifying various properties of the cell group specification objects, such as cell dimensions, font size, color pattern, and editing privileges. Typically, a spreadsheet view displays a subset of the cells in a spreadsheet. By scrolling the view you can see any data cell within a cell group of the spreadsheet. For information on specifying specification properties that define a spreadsheet view, see Working with Specifications.

Cell Groups

A **cell group** defines a rectangular section of the spreadsheet, in which all cells have the same data type and respond in a particular way to user input. The division of a spreadsheet into cell groups is similar to the panes of a window, for example:



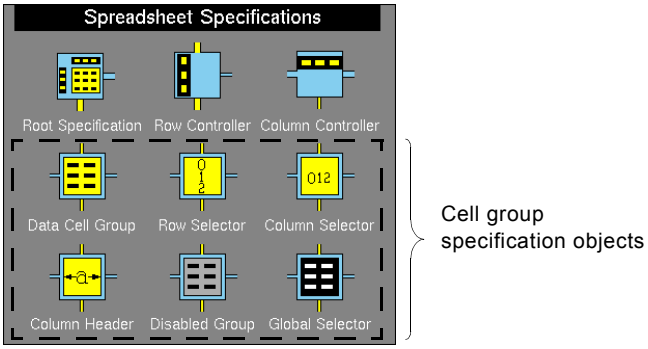
Notice that:

- Cell groups are always numbered left to right and top to bottom, starting from 0.
- All cell groups that are aligned top-to-bottom have the same number of columns.
- Cell groups that are side-by-side have the same number of rows.

When interacting with GXL programmatically, you always refer to a cell group by its number.

Cell Groups and GXL Specifications

The GXL palette of spreadsheet specifications contains a set of specification objects, of which six are **cell group specification** objects:

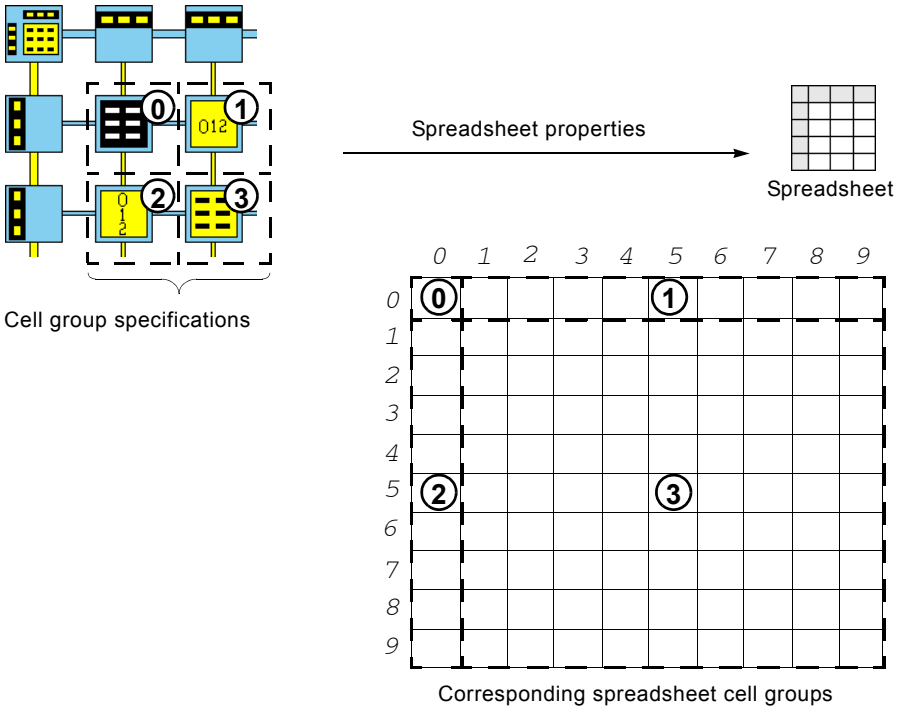


These are the cell group specification objects:

Cell Group Specification	Description
	Cells within a **Data Cell Group** accept and display a particular type of data, which can be integers, text, symbols, floats, truth-values, or quantities.
	Cells within a **Row Selector** cell group are used for selecting rows of the spreadsheet and are usually numbered.
	Cells within **Column Selector** cell group are used for selecting entire columns of the spreadsheet and are usually numbered.
	Cells within a **Column Header** cell group are used for specifying a label for one or more columns of the spreadsheet.
	Cells within a **Global Selector** cell group are used for selecting the entire spreadsheet.
	Cells within a **Disabled Group** fill areas of the spreadsheet that do not respond to user input and contain no data.

Cell Groups and Spreadsheets

When building a specification, you construct a rectangular grid of cell group specification objects that define the corresponding arrangement of cell groups within the spreadsheet, as the following figure illustrates:

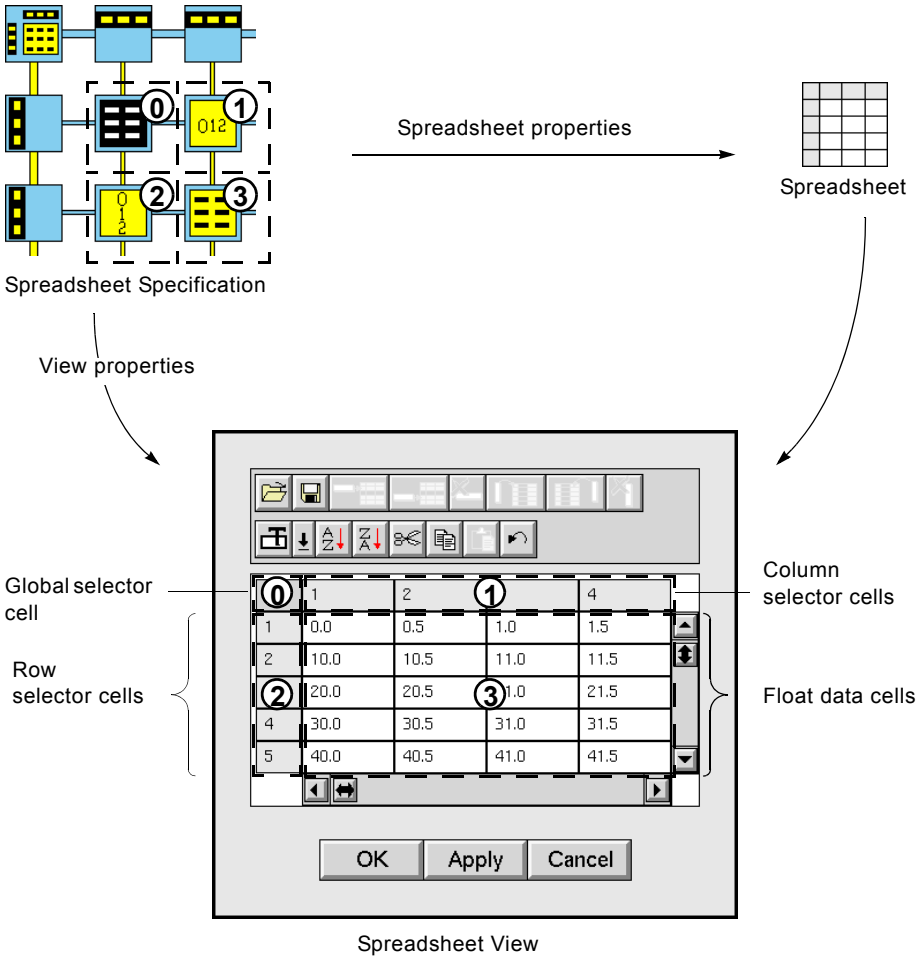


In this example, the cell group specification layout relates to the spreadsheet as follows:

- The Global Selector specification (0) defines characteristics of the first cell of the spreadsheet (R0C0).
- The Column Selector specification (1) defines characteristics of the second through tenth spreadsheet cells in the first row of the spreadsheet (R0C1::R0C9).
- The Row Selector specification (2) defines characteristics of the first spreadsheet cell in the second through tenth rows of the spreadsheet (R1C0::R9C0).
- The Data Cell Group specification (3) defines characteristics of a 10X10 grid of spreadsheet cells (R1C1:: R9C9).

Cell Groups and Spreadsheet Views

To understand how cell groups relate to a view of a particular spreadsheet, consider the following figure:



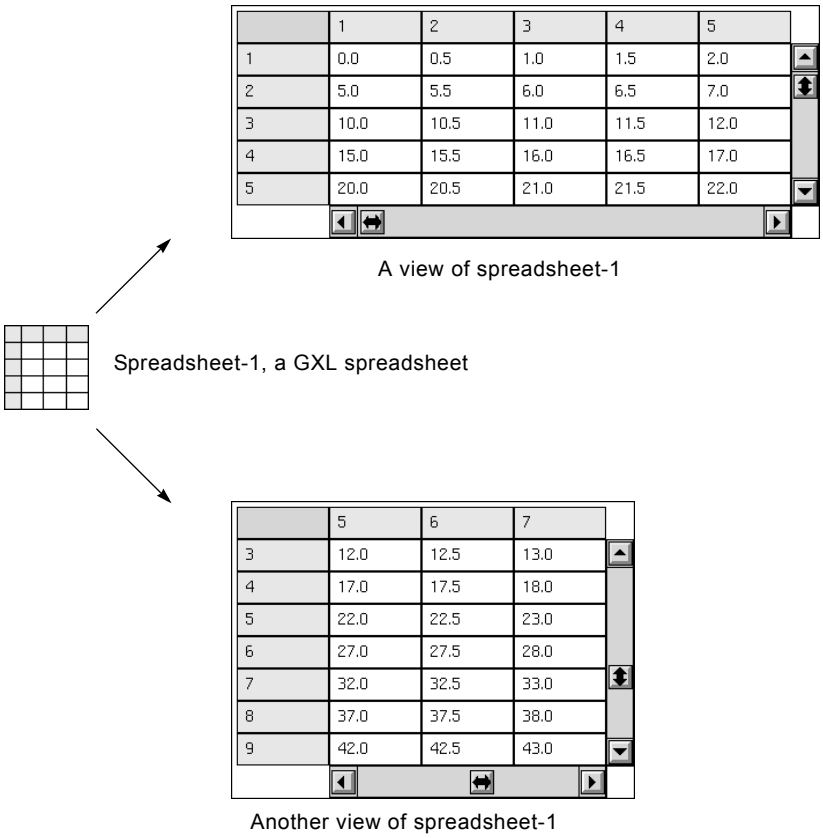
The cell group specification layout relates to the cell groups of the spreadsheet view as follows:

- The Global Selector specification (0) defines characteristics of cell group 0, which contains a single global selector cell.
- The Column Selector specification (1) defines characteristics of cell group 1, which contains column selector cells.
- The Row Selector specification (2) defines characteristics of cell group 2, which contains row selector cells.
- The Data Cell Group specification (3) defines characteristics of cell group 3, which contains float data cells.

For information on creating GXL specifications, see Working with Specifications.

Relationship between GXL Spreadsheets and Views

To support multiple clients, GXL distinguishes the spreadsheet, which is the G2 object that contains the data, from the view, which is the graphical user interface to the data that is presented to the user. A one-to-many relationship exists between spreadsheets and views, as illustrated in the following figure:



Multiple views can appear on the same or different G2 windows, and no limit exists to the number of views of a single spreadsheet.

However, because all the views of a spreadsheet share a common data source, all updates to the data in the spreadsheet are immediately reflected in each view. Values entered in one view are automatically propagated to all views of the same spreadsheet. This model of client-server interactions is suitable when changes in the data are to be displayed simultaneously for all clients.

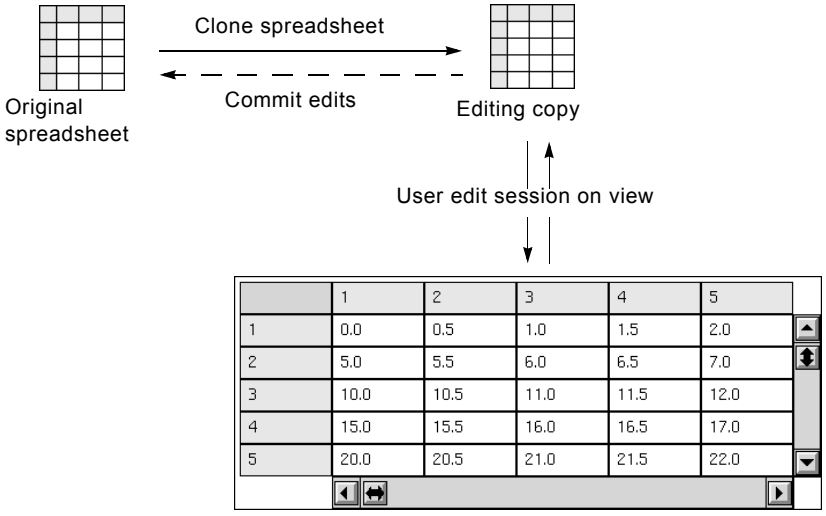
GXL Spreadsheet Data Storage

The data in a spreadsheet and its views are synchronized. If a user enters data into a view, this data is immediately stored in the spreadsheet and simultaneously propagated to all views. By default, a spreadsheet stores internally the data it displays. However, it can obtain the data it needs through user-supplied procedures. These two modes of data storage have different performance and memory characteristics. For a discussion on GXL data storage, see Managing Spreadsheet Data Storage.

GXL Edit Sessions

In some cases, you might want to use the spreadsheet in a mode where changes to the data are not carried out until the user specifically decides to commit a batch of changes. In this mode, data modifications are held in a buffer, enabling the user to cancel out of the changes, without affecting the original data. This mode of operation is called an **edit session**.

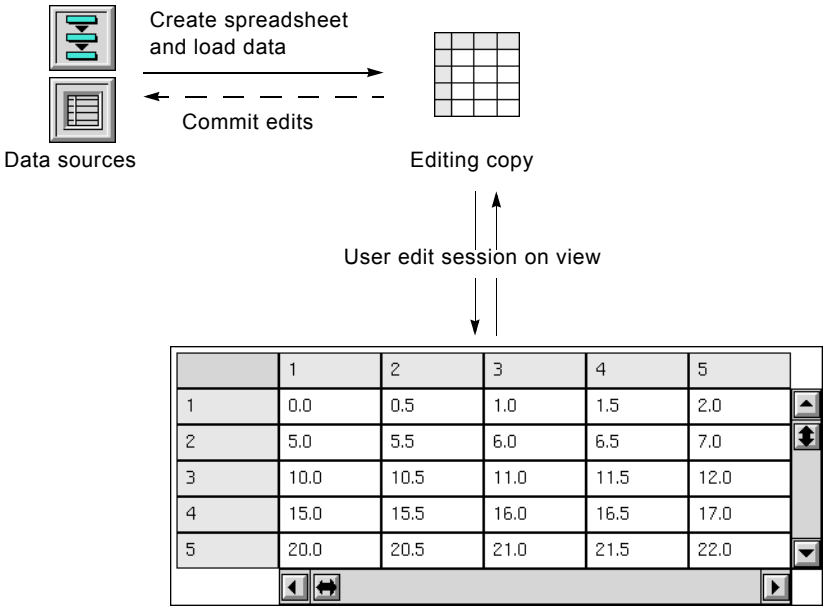
As the following figure illustrates, GXL provides an edit-session mode by creating a transient spreadsheet, which serves as a buffer to manage these edit sessions.



	1	2	3	4	5
1	0.0	0.5	1.0	1.5	2.0
2	5.0	5.5	6.0	6.5	7.0
3	10.0	10.5	11.0	11.5	12.0
4	15.0	15.5	16.0	16.5	17.0
5	20.0	20.5	21.0	21.5	22.0

When you edit the spreadsheet, GXL creates a transient editing copy of the spreadsheet by cloning and displays a view of the cloned spreadsheet. You make changes to the editing copy of the spreadsheet through this view. If you choose to commit the changes, GXL transfers the data from the editing copy to the original spreadsheet. If you cancel the edit, GXL simply deletes the editing copy and its view.

You can also set up an edit session on a data source that is not a spreadsheet, as the following figure illustrates:



Here, you could use one or more G2 objects as data sources, such as a text list and a float array. Using the programmatic interface to GXL, you begin the edit session by creating a spreadsheet with appropriate dimensions and then loading the data from the data sources.

The user then launches a view of the spreadsheet, enabling the user to alter the data in the spreadsheet. When the user commits the edit, you programmatically copy the data back to the original data sources and delete the spreadsheet and its view.

For information on data storage options when using GXL, see Managing Spreadsheet Data Storage.

GXL and Conventional Spreadsheets

GXL supports many operations that allow you to manually or programmatically change the spreadsheet structure and values. You can:

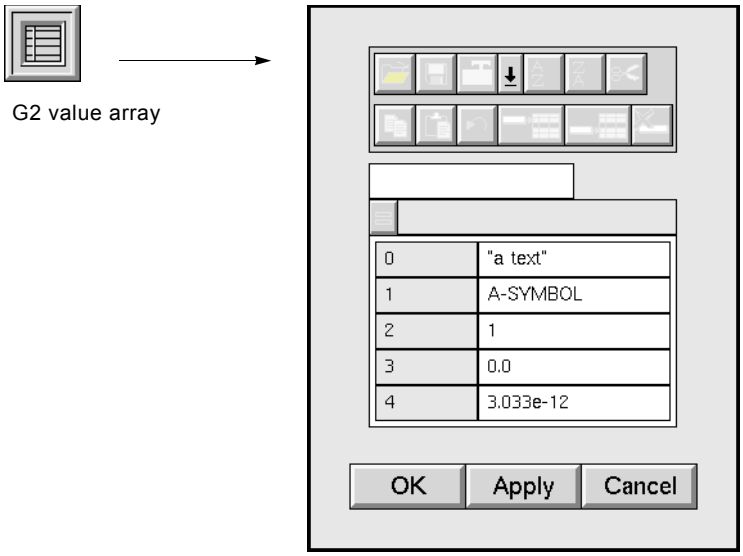
- Add and remove rows and columns.
- Cut, copy, and paste.
- Use formulas.
- Read and write data to files.

While many of the features of GXL are similar to conventional spreadsheets, a number of important differences exist between GXL and conventional spreadsheets:

- In GXL, formulas are not stored as part of the spreadsheet's permanent data structure. The use of formulas is limited to one-time calculation of cell values. Because formulas are not stored, a cell whose value has been calculated from other cells does not automatically update when those cells receive new values.
- A GXL spreadsheet has a definite size which is first established when the spreadsheet is created. The user must specifically add or remove rows and columns to modify the dimensions of the spreadsheet.
- Cells in GXL spreadsheets always have one of the following G2 data types: value, quantity, text, symbol, truth-value, float or integer. Although multiple cell types can be used in a single spreadsheet, they cannot be randomly interspersed.
- Within a single scrolling area, all cells must have the same height and width. You cannot dynamically resize cells on a view after the view is created. If data does not fit into a cell, GXL displays a truncation of the cell's data.
- In GXL, the cells that enumerate the rows and columns are stored as part of the spreadsheet's data, rather than being a part of a permanent "frame" outside of the spreadsheet. Inclusion of these cells is optional. This gives you the flexibility to specify the exact appearance of a spreadsheet view.

Editing G2 Lists and Arrays

Using the editing feature of GXL, you can view and edit G2 lists and arrays. When the GXL editing feature is active, choosing edit from a G2 list or array menu displays a view similar to the following:



If the list or array does not contain any data elements, GXL displays a view with an empty data cell.

The GXL editing feature is available from the GXL top-level workspace.

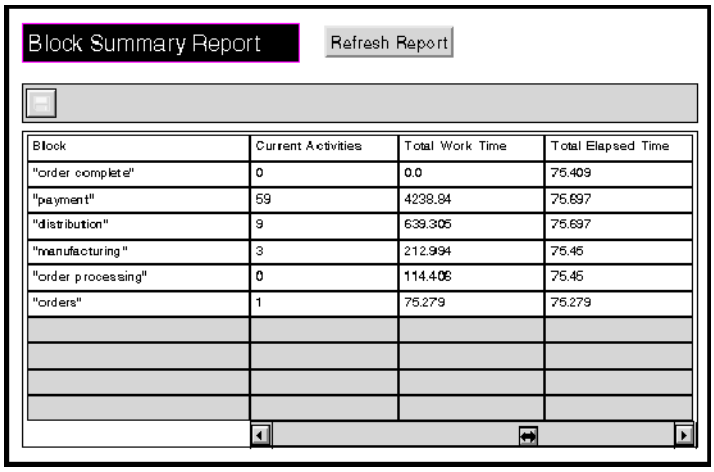
For information on...	See...
Editing G2 lists and arrays with GXL spreadsheet views	Using GXL to Edit G2 Lists and Arrays
Using GXL spreadsheet view tools to interact with list and array data	Using Spreadsheet Views

Application Programmer's Interface to GXL

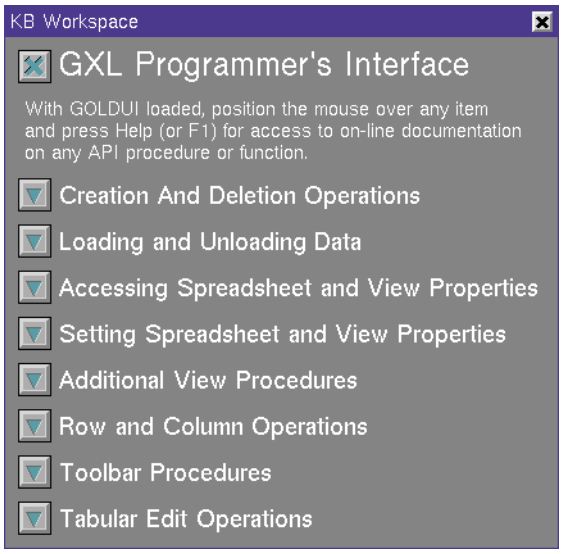
All programmatic interactions with GXL take place through a small set of public procedures, classes, and attributes. Collectively, these items are referred to as the **application programmer's interface** (API).

Using the GXL application programmer's interface within G2 procedures, you can programmatically create spreadsheet views, populate them with data, and display them on one or more G2 windows. For example, ReThink® uses GXL to

display summary reports of various statistics associated with model processes, as this figure shows:



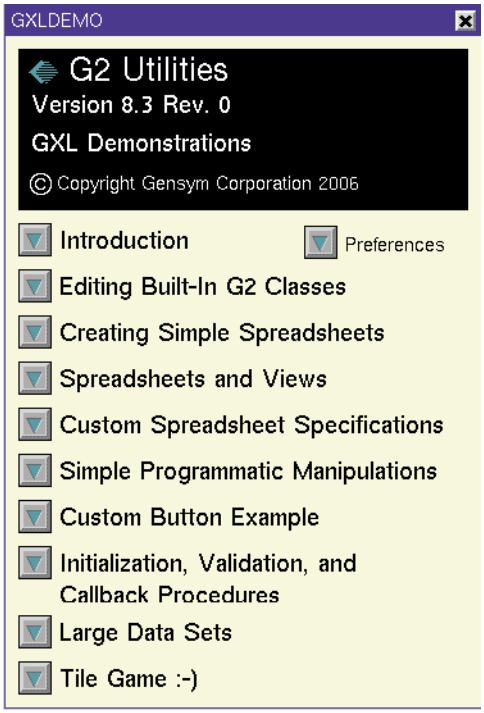
Workspaces containing the GXL programmer's interface of procedures and methods are available from the GXL top-level workspace.



For complete descriptions of these procedures and methods, see the corresponding chapters in Part II, API Reference.

GXL Demo KB

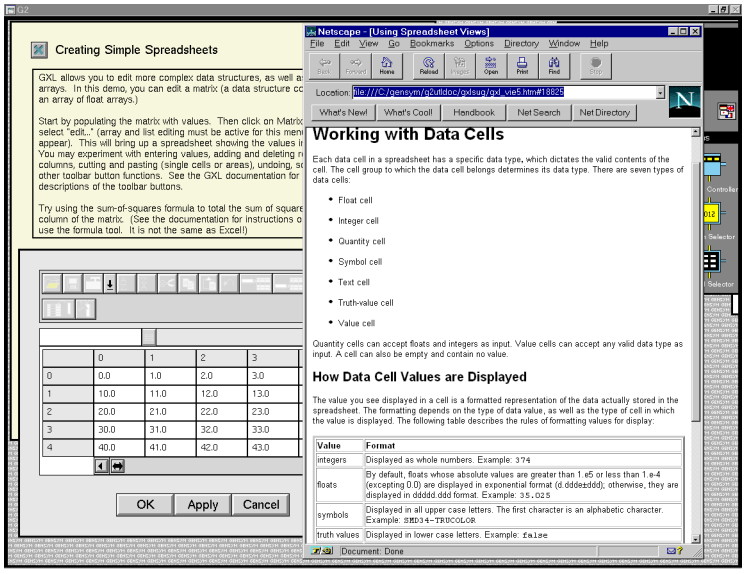
GXL provides a demo KB, which contains examples of GXL capabilities:



The examples show how GXL spreadsheets and views are constructed and how their behaviors are defined by setting attributes and using calls to API procedures. For information on installing the gxldemo KB and viewing the GXL examples, see The GXL Demo KB.

GXL Online Documentation

Using the G2 OnLine Documentation (GOLD) utility, you can view GXL documentation on a Web browser from G2, as the following figure illustrates:



A separate module, GOLDUI, provides the user interface for setting up and using GOLD. See the *G2 OnLine Documentation User's Guide* for instructions on installing and setting up GOLD for online viewing of GXL documentation.

2

Getting Started

Describes how to install the GXL utility and introduces the features available from the top-level workspace.

Installing GXL

The G2 XL Spreadsheet (GXL) is a knowledge base (KB) module, whose name is gxl. All components of GXL are identified by either the public gxl- prefix or private _gxl- prefix.

You install GXL by merging it into any modularized knowledge base. When you merge GXL, its required modules are automatically loaded into G2.

The filename of the GXL utility is *gxl.kb*. The default location of this KB is the *utils* subdirectory of the *kbs* directory under the *g2* directory.

To merge GXL into your KB:

1. Pause or reset your KB.
2. Choose Merge KB from the Main Menu to display the Load KB workspace.

   The merge in this KB option is enabled on the workspace.
3. Specify the location of the *gxl.kb* file and click End.

Tip When merging GXL, let G2 resolve conflicts by enabling the automatically resolve conflicts option.

When you merge GXL into your KB, it is not a required module unless it is specified in the Module Information table of your KB.

To make GXL a required module:

1. Choose Main Menu > System Tables > Module Information.
2. Specify GXL as a directly required module of the appropriate existing module of your KB.

For more information on merging KBs and making a KB a required module, see the *G2 Reference Manual*.

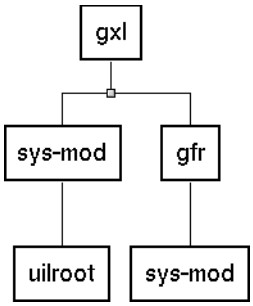
GXL Required Modules

When you merge GXL into your KB, its required modules are automatically loaded into G2. The following table describes these modules:

Module	File Name	Contents
gxl	*gxl.kb*	Definitions and API support for the G2 XL Spreadsheet utility.
gfr	*gfr.kb*	Definitions and API support for the G2 Foundation Resources (GFR) utility.
sys-mod	*sys-mod.kb*	The library of G2 system procedures.
uilroot	*uilroot.kb*	Definitions and API support for navigation buttons.

Note These module dependencies are subject to change in future versions of GXL.

This is the module hierarchy:



Requirements for Running GXL

The GXL module contains a table of version information that includes the minimum version of G2 in which the current version of GXL will run. This table of version information is available from the GXL top-level workspace:

To view the version information:

1 Choose Main Menu > Get Workspace > gxl-top-level.

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

Starting GXL

GXL works only when G2 is running. After merging GXL, resume or start G2.

To start GXL:

➔ Choose Resume or Start from the Main Menu.

Choosing a User Mode

You can use any mode user mode with GXL. GXL functions work the same in Administrator mode as in Developer mode.

Considerations and Restrictions

Drawing Parameters

To ensure the proper operation of GXL, you should verify the value for the drawing parameters that allow scheduled drawings and enable paint mode. These parameters are located in the Drawing Parameters system table.

To verify the drawing parameter settings:

1 Choose Main Menu > System Tables > Drawing Parameters to display the table.

2 Verify the values for the following parameters:

Drawing Parameter	Value
allow-scheduled-drawing	yes
paint-mode	yes

If necessary, change the value to yes.

Timing Parameters

To ensure optimum scrolling, you should set the minimal scheduling interval to either 0.1 or 0.05 seconds. The default is 1 second. This parameter is located in the Timing Parameters system table.

To change the minimal scheduling interval setting:

1 Choose Main Menu > System Tables > Timing Parameters to display the table.

2 Change the value of minimum-scheduling-interval to either 0.1 or 0.05 seconds.

Cloning Restrictions

Caution Certain cloning operations can corrupt your GXL application.

GXL does not support cloning of views. If you clone a view using Operate on Area or clone the workspace of the view, the cloned view will not function properly. When working with GXL:

- Do not clone any GXL item when G2 is reset.
- Do not clone GXL items by cloning a workspace containing GXL objects.
- Do not clone spreadsheet views.

Any of these actions will cause the relationships between specifications, spreadsheets, and views to become corrupted. These corruptions, if they occur, can only be repaired by deleting the affected items.

Programming Restrictions with G2 Actions

When you work with GXL programmatically, certain restrictions apply to the following G2 actions:

- Making GXL spreadsheets and views permanent.
- Deleting spreadsheets and views.
- Moving spreadsheet views.

In all cases you should use the API to GXL rather than the G2 action.

The following table lists the API to call to perform the appropriate action:

To...	Call this API...
Make a spreadsheet permanent	gxl-make-spreadsheet-permanent
Make a view permanent	gxl-make-spreadsheet-view-permanent
Delete a spreadsheet	gxl-delete-spreadsheet
Delete a view	gxl-delete-view
Move a spreadsheet view	gxl-move-spreadsheet-view

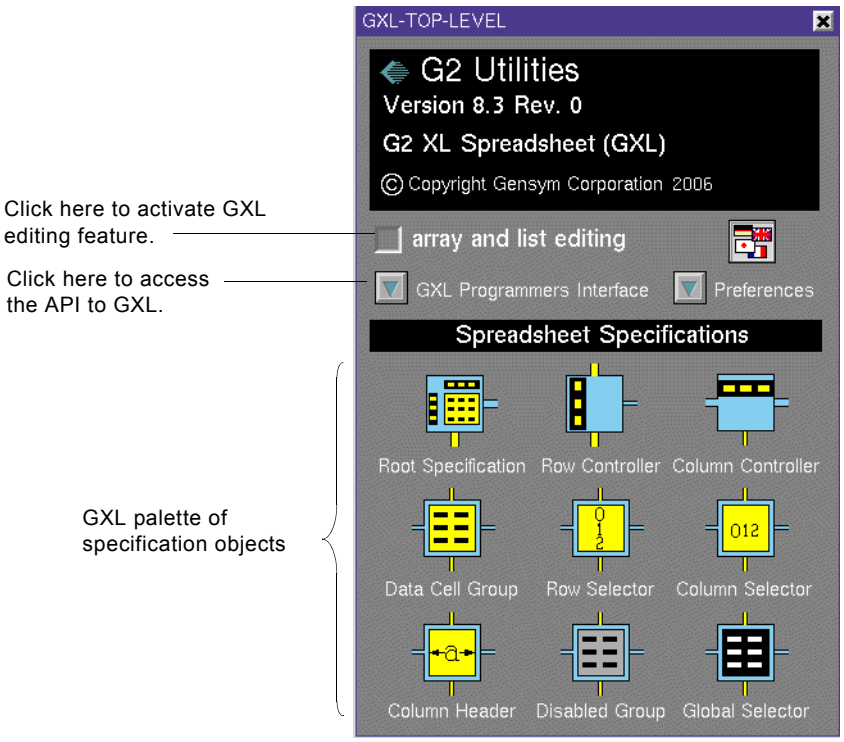
Accessing GXL Features

After merging GXL and starting G2, you can access these GXL features from the GXL top-level workspace:

- GXL array and list editing.
- The Application Programmer's Interface to GXL.
- The palette of GXL specification objects for building custom spreadsheets and views.

To display the GXL top-level workspace:

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



For information on...	See...
Array and list editing	Using GXL to Edit G2 Lists and Arrays
GXL Programmers Interface	Part II, API Reference
Spreadsheet Specifications	Creating a Custom Spreadsheet and View Working with Specifications

The GXL Demo KB

GXL provides a KB of examples that cover the main features of GXL. This KB, named *gxldemo.kb*, is located in the G2 *utils* subdirectory in the *kbs* directory under the *g2* directory.

To view the GXL examples:

1. Merge *gxldemo.kb* into your KB.
2. Resume G2.

After resuming G2, the GXL demonstrations workspace appears:

This workspace contains the following subworkspaces:

Topic	Contents
Introduction	Description of the demo KB.
Editing Built-In G2 Classes	Example describing how to activate the editing feature and instances of the lists and arrays you can edit with GXL.

Topic	Contents
Creating Simple Spreadsheets	Example showing how you can edit data structures with GXL and how to use the GXL formula bar.
Spreadsheets and Views	Example describing the distinction between spreadsheets and views and showing how specifications are used to create different views of the same data.
Custom Spreadsheet Specifications	Example showing how attributes of the specification objects define views of spreadsheet data.
Simple Programmatic Manipulations	Walk-through example of API procedures that create a view of a spreadsheet, load it with data, and manipulate the view.
Custom Button Example	Example showing how to create a custom button for the GXL toolbar.
Initialization, Validation, and Callback Procedures	Examples of callback procedures that respond to cell creation, cell selection, data entry, and data display.
Large Data Sets	Example describing how to display large sets of data using GXL's "on demand" mode.

To review a topic:

➔ Click the button next to the topic you want to view.

As you browse the demo KB, you can see how to construct GXL spreadsheets and views and how to define certain behaviors by setting attributes and using API calls.

Note Be sure you are in a non-administrator mode when you use the gxldemo KB. Otherwise, clicking a button displays a menu rather than the example workspace.

To close the GXL examples workspace:

➔ Choose Hide Workspace from the workspace menu.

To redisplay the GXL examples workspace:

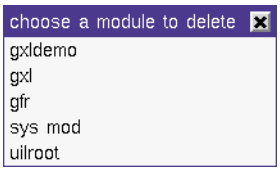
➔ Choose Main Menu > Get Workspace > gxldemo.

Removing the Demo Module

Since the GXL module of examples is not a required module, you may delete the it when you finish viewing the GXL examples.

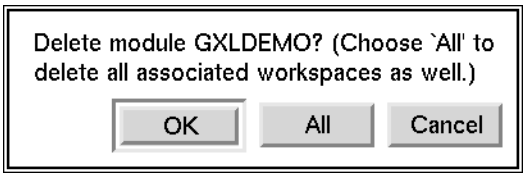
To delete the GXL examples module:

1 Choose Main Menu > Miscellany > Delete Module to display a menu of the modules currently loaded in G2:



2 Choose gxldemo.

The following dialog appears:



3 Click All to delete the module and all its associated workspaces.

Using Online Help

You can view GXL documentation online. Using the G2 OnLine Documentation (GOLD) utility, you can access GXL documentation from G2 and view it on a Web browser.

GOLD provides online viewing of HTML (HyperText Markup Language) files, which correspond to the printed product documentation. HTML files are standard files, which any HTML browser on any platform can display.

Using GOLD, you can:

- Display the table of contents or index of a book.
- Search the indexes of all available books for help topics.
- Display context-sensitive help on a particular item.

The browser of choice launches automatically to display the online help. Once the desired online file appears in the browser, you can navigate around the entire online documentation set by using:

- Hypertext, or "hot" links imbedded in the displayed document.
- Navigation buttons of the online document.
- Browser navigation buttons.

A separate module, GOLDUI, provides the user interface for setting up and using GOLD. For detailed instructions on installing the goldui KB and setting up GOLD for viewing GXL documentation online, see the *G2 OnLine Documentation User's Guide*.

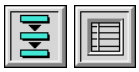
Accessing Online Documentation

When G2 is running, you can access the online documentation associated with GXL in the following ways:

- Choose Help from the G2 Main Menu.
- Position your cursor on a GXL item and press the F1 key for context-sensitive help.

Note For context-sensitive help to work correctly, the user mode can be any mode *except* administrator.

Using GXL to Edit G2 Lists and Arrays



One of the simplest applications of GXL spreadsheets is for editing system-defined classes of G2 lists and arrays. Using GXL spreadsheets, you can edit any of the following classes:

G2 Lists	G2 Arrays
value-list	value-array
symbol-list	symbol-array
text-list	text-array
truth-value-list	truth-value-array
quantity-list	quantity-array
float-list	float-array

G2 Lists	G2 Arrays
integer-list	integer-array
item-list	item-array

Note To edit item-list or item-array classes with GXL, each element in the item-list or item-array must be one of the other listed classes.

For information on G2 lists and arrays, see the *G2 Reference Manual*.

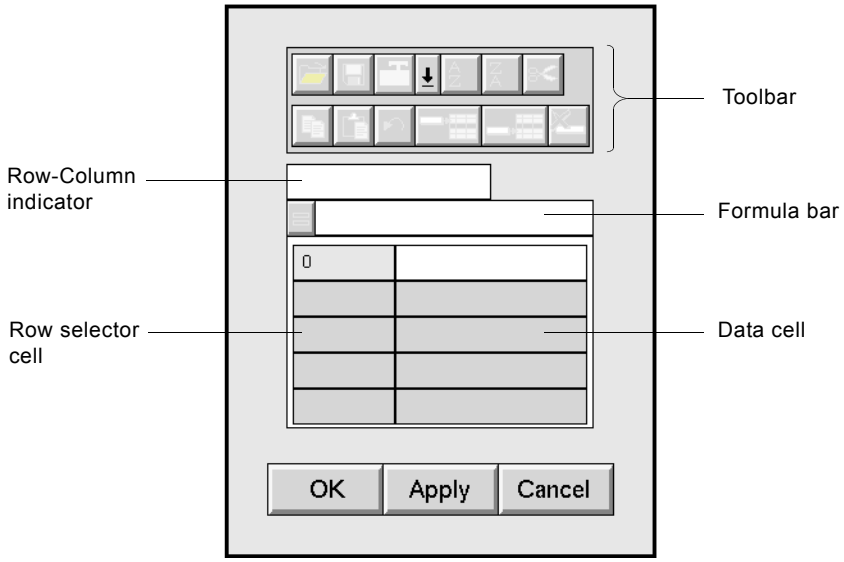
To enable GXL editing for G2 lists and arrays:

1. Get the gxl-top-level workspace.
2. Click the check box labelled array and list editing, as follows:

Note GXL features work only when G2 is running.

Spreadsheet View of G2 Lists and Arrays

When the GXL editing feature is active, choosing edit from the menu of a G2 list or array displays a spreadsheet view, similar to the following:



The main parts of the view display are:

Spreadsheet Item	Description
Toolbar	The toolbar contains a set of icon tools that allow you to manipulate and change the data in the spreadsheet view.
Row-Column indicator	This tool serves as a navigation aid by displaying information related to the location and type of cells in the spreadsheet.
Formula bar	This tool allows you to calculate cell values, using mathematical operations, built-in G2 functions, or functions you define.
Row selector cell	This is a special cell that allows you to select an entire row.
Data cell	The Data cell accepts and displays data of the type conforming to the array or list type: integers, floats, quantities, text, symbols, truth-values, G2 values.

Spreadsheet Item	Description
OK and Apply buttons	These buttons save the edits made in the spreadsheet view. The OK button also ends the edit session and closes the view.
Cancel button	This button closes the view. The spreadsheet returns to the original data of the cells, canceling any updates.

For detailed descriptions of the parts of a spreadsheet view, see Using Spreadsheet Views.

Editing a G2 Array

The following example shows how to edit a G2 float array, using GXL. After creating a G2 float array on a workspace, you display and edit the array using a GXL spreadsheet.

To create a G2 float array:

➔ Choose New Object > g2 array > value-array > quantity-array > float-array from the KB Workspace menu.

For information about how to create instances of G2 classes, see the *G2 Reference Manual*.

To edit a G2 array:

➔ Choose edit from the float array menu to display the spreadsheet view.

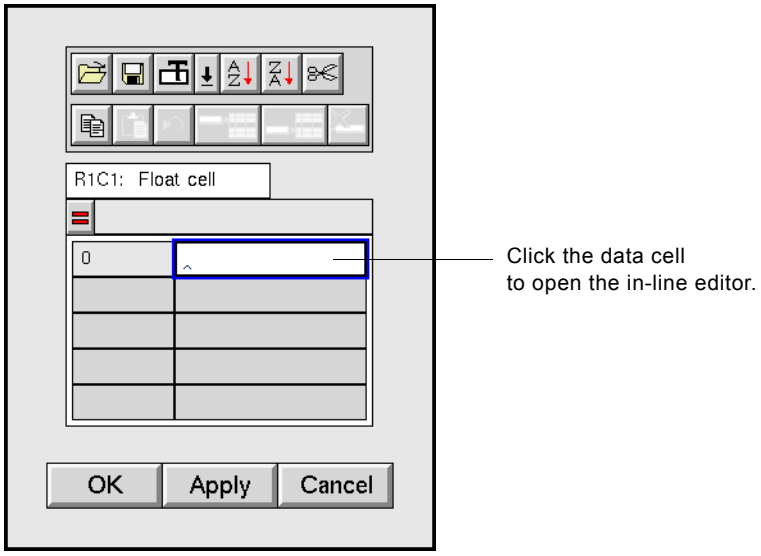
Entering Data into the Spreadsheet

Note For complete information on interacting with a GXL spreadsheet view, see Using Spreadsheet Views.

To enter data:

1 Click the data cell, which opens the in-line editor on the cell.

For example:



2 Type any float in the edit field.

Hint The standard G2 editor keyboard shortcuts work when entering data into a cell; for example, Control - A aborts the edit. While the editor is open, pressing Control - ? displays the valid keyboard shortcuts.

---

3 Finish editing the cell by pressing Enter to accept your input.

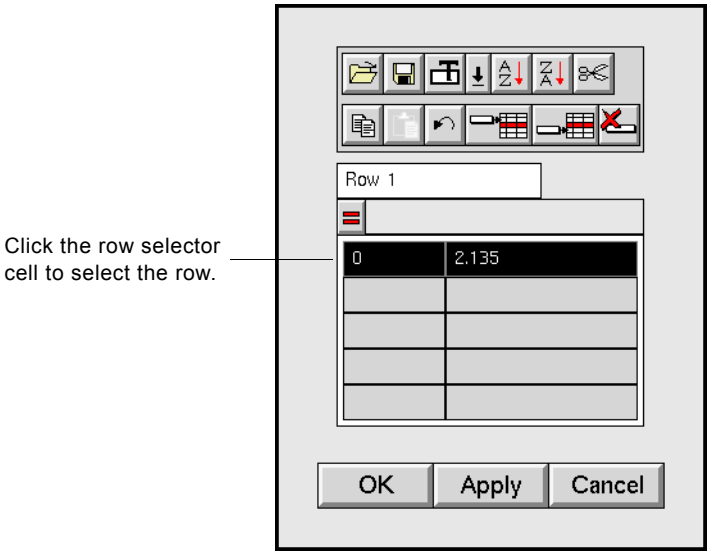
Adding Rows to the Spreadsheet

To add a second and subsequent elements to the float array, you must add new rows to the spreadsheet.

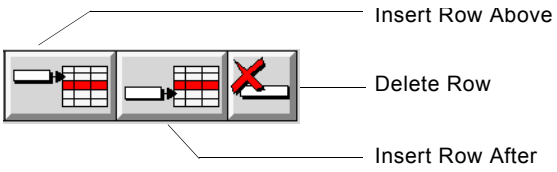
To add a row to the spreadsheet:

1 Click the first row selector cell to select the row.

For example:

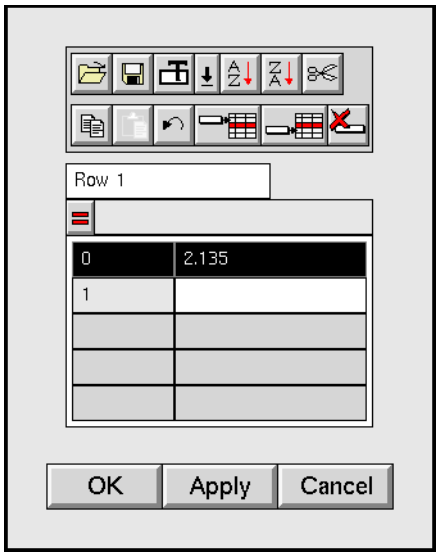


Selecting a row activates the following toolbar buttons:



2 Click the Insert Row After button.

GXL adds a new row to the spreadsheet, for example:



You can now enter a value into the second row.

If you make a mistake entering data, you can select the cell again and make the necessary correction. However, if you enter the wrong data type, the edit is not accepted.

For complete instructions on using GXL tools to input or update list and array values, see Using Spreadsheet Views.

Saving Spreadsheet Edits to the Array

During an edit session, your changes are stored in a buffer area. For any changes to take effect, you must save your edits. Saving your edits:

- Changes the dimension of the array to the current number of rows of the spreadsheet view.
- Assigns the values from the spreadsheet to the array.

To save the updates to the float array:

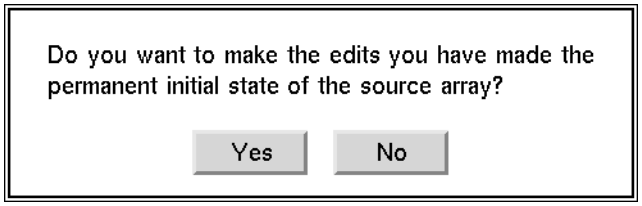
➔ Click either the OK button or the Apply button.

Button	Action
OK	Choosing OK ends the edit session and closes the view,
Apply	Choosing Apply keeps the view open, allowing further edits.

If you do not want to save your edits to the float array, click the Cancel button.

Making Your Edits the Initial Value of the Array

Clicking OK or Apply when editing the float array displays the following dialog:



If you want your edits to persist after resetting or restarting G2, you must make the values you have entered into the spreadsheet the initial values of the float array.

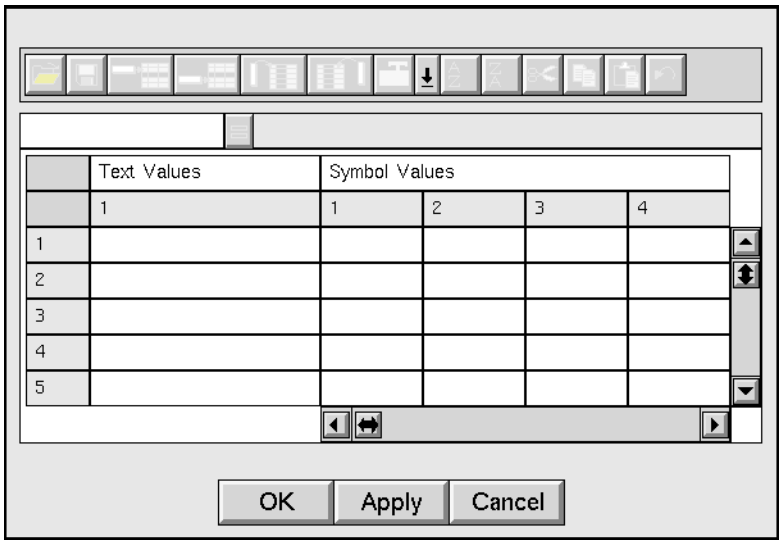
- If you choose Yes, GXL copies the values from the spreadsheet into the array as the initial values. The array contains these initial elements the next time you reset or restart G2.
- If you choose No, the values from the spreadsheet become the current values of the array, but the initial values of the array are not affected.

Note For information on programmatic control of the GXL editing feature, see Controlling GXL Editing Programmatically.

Creating a Custom Spreadsheet and View

To create a spreadsheet or a view, you construct a graphical layout of GXL specification objects. The specification layout serves as the template for defining and creating both a spreadsheet and a spreadsheet view.

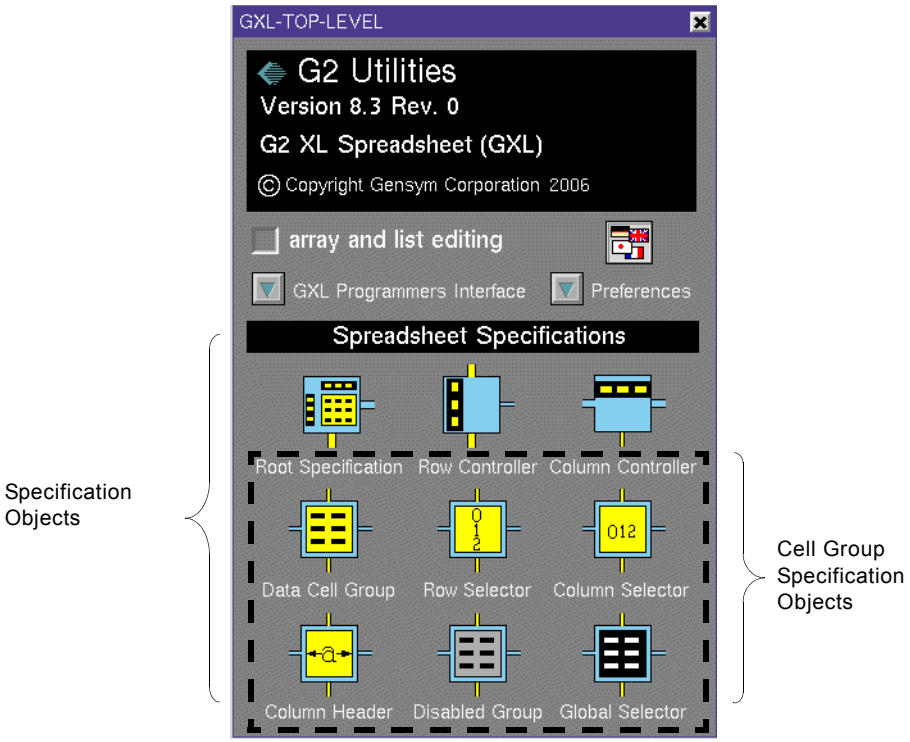
Suppose you want to create a spreadsheet with the following view:



This spreadsheet view has the following characteristics:

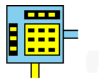
- The first row contains column labels.
- The second row contains column selector cells.
- The first column is 50 pixels wide and contains row selector cells.
- The second column is 180 pixels wide and contains text cells.
- There is a 1-column by 20-row grid of text cells, of which five rows are visible.
- There is a 10-column by 20-row grid of symbol cells, of which four columns and five rows are visible.
- Each column of symbol cells is 80 pixels wide.
- Because rows and columns cannot be deleted, the toolbar does not include the delete row and delete column buttons.

You build a specification on a workspace by cloning and connecting GXL specification objects, which are located on the GXL top-level workspace.



The GXL Specification

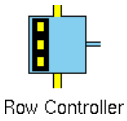
A specification layout consists of:


Root Specification

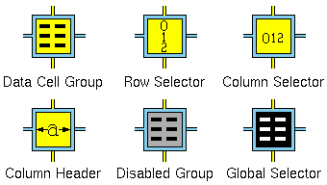
One **root specification** that defines certain properties of the spreadsheet as a whole.


Column Controller

A **column controller** for each column of cell groups. Each column controller defines properties that apply to all cell groups connected to it vertically on the specification layout.



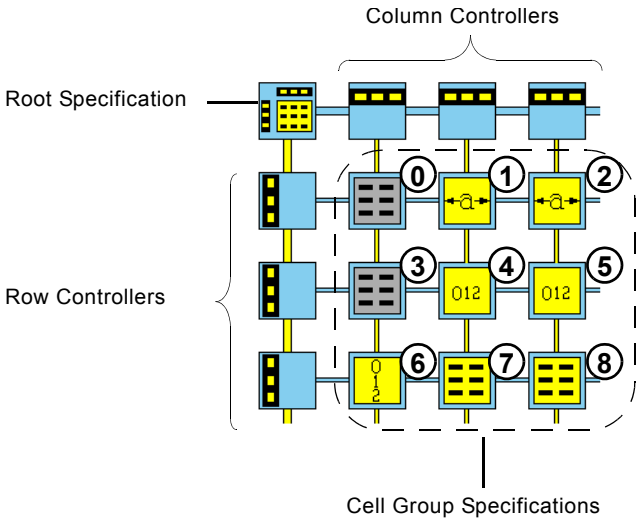
A **row controller** for each horizontal row of cell groups. Each row controller defines properties that apply to all cell groups connected to it horizontally on the specification layout.



A **cell group specification** object for each cell group. Each cell group specification defines the properties of all cells within the particular group.

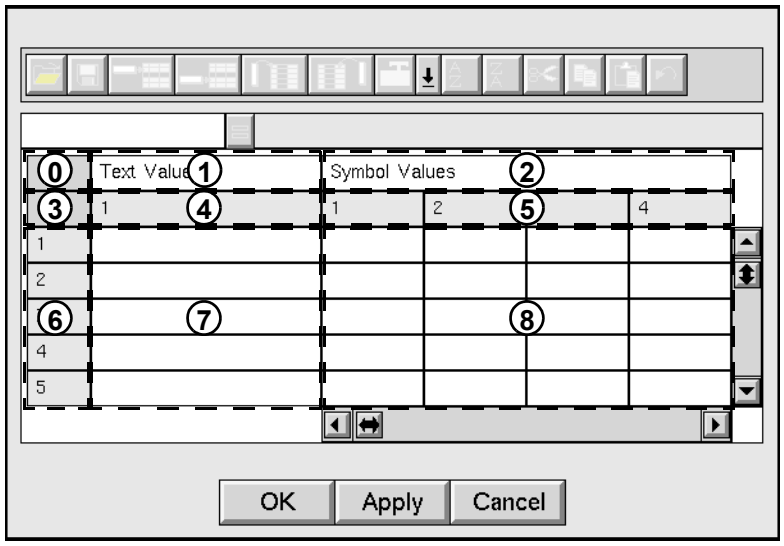
The cell group specifications are laid out in a rectangular grid that corresponds to the arrangement of cell groups within the spreadsheet. A specification layout must always have a full rectangular grid of cell group specification objects.

This is the layout of the spreadsheet specification that defines the view example:



Notice that cell groups are always numbered left to right and top to bottom, starting from 0.

Looking at the specification layout and view examples again, notice that each cell group specification object corresponds to a particular area of the view.



Attributes of a particular cell group define properties of the corresponding area of the spreadsheet view.

Looking at the view example, also notice that:

- Cell groups that are side-by-side have the same number of rows.
- Cell groups that are aligned top-to-bottom have the same number of columns, with the exception of column headers, which can straddle all columns of a cell group.

The row controller and column controller specifications control the number of rows and columns.

To create a spreadsheet and view:

1 Build a specification layout.

2 Define the properties of the spreadsheet and view.

3 Customize the toolbar, if necessary.

4 Create the spreadsheet object.

5 Display the spreadsheet view.

For complete information on specification objects and their attributes, see Working with Specifications.

Building a Specification Layout

When building a specification layout, remember that the specification defines both the spreadsheet and a view of the spreadsheet. The attributes of the specification objects define two categories of attributes: those related to the spreadsheet and those related to its view.

To construct a specification layout:

1. Create a new workspace to contain the specification.
2. Get the workspace named gxl-top-level, which contains the palette of specification objects.
3. Clone the following objects from the palette onto the new workspace:
   - One root specification
   - Three column controller
   - Three row controller
   - Two disabled cell groups (groups 0 and 3)
   - Two column headers (groups 1 and 2)
   - Two column selectors (groups 4 and 5)
   - One row selector (group 6)
   - Two data cell groups (groups 7 and 8)
4. Connect the objects as shown in the example.
5. Align the connected objects.

To connect specification stubs:

1. Click one of the connection stubs.

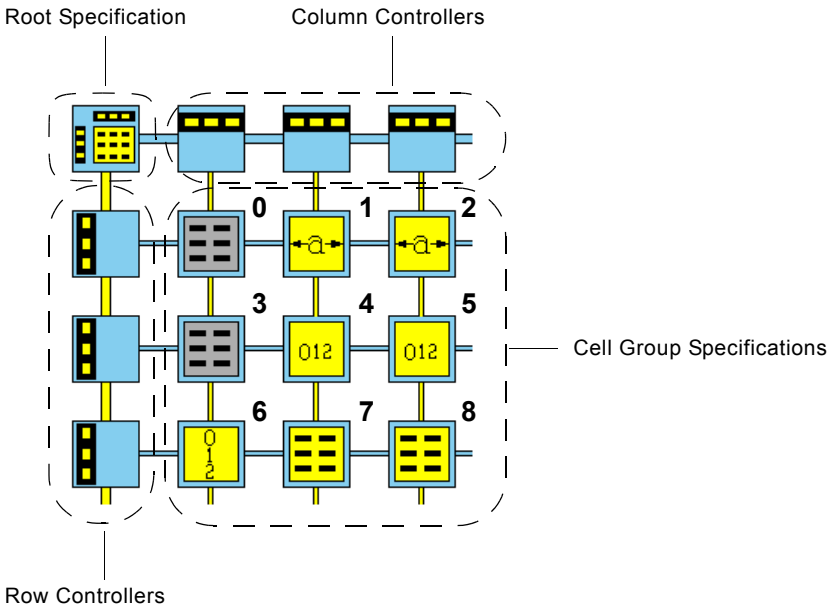
   When you move the mouse, the connection stub follows the cursor.
2. Position the stub directly over the adjacent stub.
3. Click once more to connect the two stubs together.

Caution When connecting specification objects, you must use existing stubs. Do not drag the connector directly into another object.

To Align the specification objects:

➔ Choose arrange from the root specification menu.

Your specification layout should look like this:



Defining Spreadsheet and View Properties

After you construct the layout, you need to specify the attribute values that define the properties of the spreadsheet and view.

To define the spreadsheet and view properties:

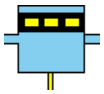
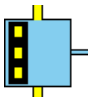
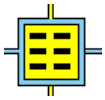
➔ Click the appropriate specification object and choose table from the menu to display its attribute table.

When specifying the spreadsheet and view properties, remember:

- The column controller attributes define properties for all cell groups connected to it vertically on the specification layout.
- The row controller attributes define properties for all cell groups connected to it horizontally on the specification layout.

Specifying Spreadsheet Properties

The following table summarizes the specification attributes controlling the spreadsheet properties that you must define:

Specification Object	Attribute	Description
 Column Controller	gxl-total-columns	Total number of columns.
 Row Controller	gxl-total-rows	Total number of rows.
 Data Cell Group	gxl-cell-type	Data type of its cells.

All other attributes of the specification objects related to spreadsheet properties remain set to their default values.

To specify the total number of columns:

➔ Change the gxl-total-columns attribute of the third column controller to 10.

To specify the total number of rows of data cells:

➔ Change the gxl-total-rows attribute of the third row controller to 20.

To specify the width of the columns:

➔ Edit the gxl-cell-width attribute of the column controllers as follows:

- Change the gxl-cell-width of the first column controller to 50.
- Change the gxl-cell-width of the second column controller to 180.

To specify the valid data type of a cell group:

➔ Edit the gxl-cell-type attribute of the data cell groups as follows:

- Change the gxl-cell-type of data cell group (7) to text-cell.
- Change the gxl-cell-type of data cell group (8) to symbol-cell.

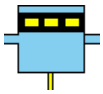
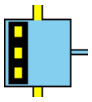
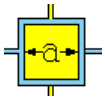
Specifying View Properties

The properties of the view example are:

- The first column, containing row selector cells, is 50 pixels wide.
- The second column is 180 pixels wide.
- Each column of symbol cells is 80 pixels wide, which is the default width.
- The first row contains the column headers “Text Values” and “Symbol Values.”
- Five rows of text data cells are visible.
- Four columns and five rows of symbol cells are visible.

If the spreadsheet contains more rows or columns of cells than the view can display, scroll bars appear, enabling you to view all of the cells.

The following table summarizes the specification attributes controlling the view properties that you must define.

Specification Object	Attribute:	Description
 Column Controller	gxl-cell-width	Column width.
	gxl-visible-columns	Number of visible columns.
 Row Controller	gxl-visible-rows	Number of visible rows.
 Column Header	gxl-cell-group-initialization-data	Column heading.

All other attributes of the specification objects related to view properties remain set to their default values.

To specify the number of the visible columns:

➔ Change the gxl-visible-columns attribute of the third column controller to 4.

To specify the number of the visible rows:

➔ Change the gxl-visible-rows attribute of the third row controller to 5.

To specify the column headers:

➔ Edit the gxl-cell-group-initialization-data attribute of the column header cell groups as follows:

- Change the gxl-cell-group-initialization-data of column header (1) to "Text Values."
- Change the gxl-cell-group-initialization-data of column header (2) to "Symbol Values."

When specifying the column heading, enclose the text string with double quotes (").

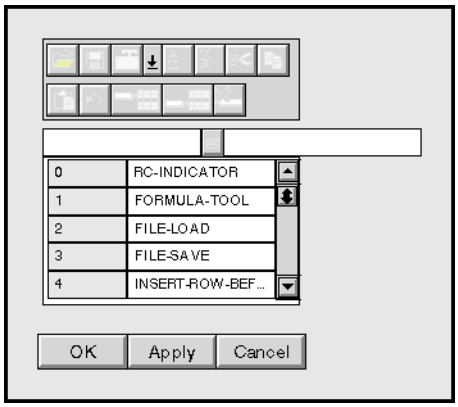
Customizing the Toolbar Display

Next, you decide which tools will be available on the spreadsheet view. Because deleting rows and columns from the spreadsheet is not allowed, you must remove the Delete Row and Delete Column buttons from the toolbar.

To determine which tools appear on the view:

1 Click the root specification and choose edit spreadsheet tools from the menu.

GXL displays a spreadsheet view that lists the default tools, for example:

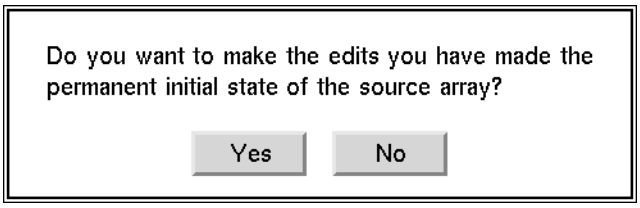


2 Select the row containing delete-row and click the Delete Row button on the toolbar.

3 Select the row containing delete-column and click the Delete Row button on the toolbar.

4 Click OK.

GXL displays the following dialog:



5 Click Yes to make your edits the initial state of the tool array.

Creating the Spreadsheet

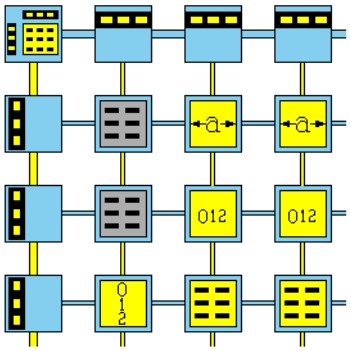
You create the spreadsheet from the root specification object.

To create the spreadsheet:

➔ Click the root specification and choose make spreadsheet from the menu.

A spreadsheet object appears to the left of the root specification:

Spreadsheet object



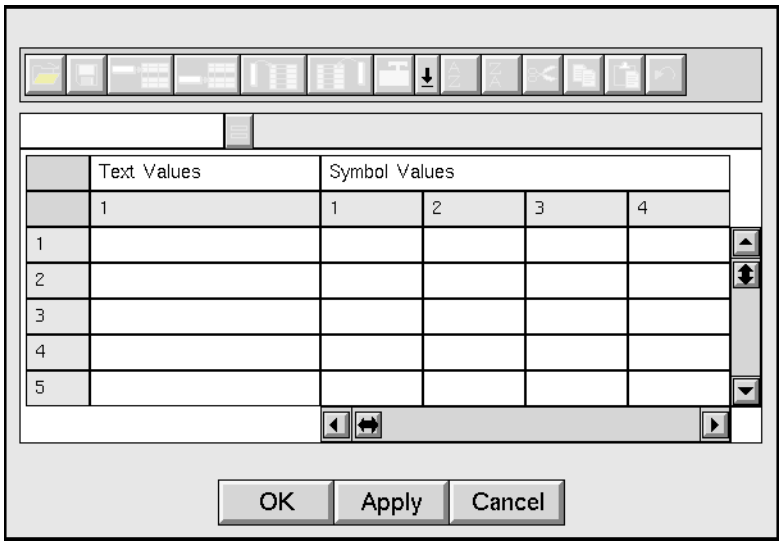
Displaying the Spreadsheet View

You display a view of the spreadsheet, using the spreadsheet object. However, GXL refers to the specification to define the view.

To display the view of the spreadsheet:

➔ Click the spreadsheet object, and choose edit from the menu.

Your view display should look like this:



The following table summarizes the location of information related to the creation of spreadsheet specifications:

For information on...	See...
Specification objects and their attributes	Working with Specifications
Customizing GXL tools	Customizing the Toolbar
Creating spreadsheets and views programmatically	Creation and Deletion Operations

3

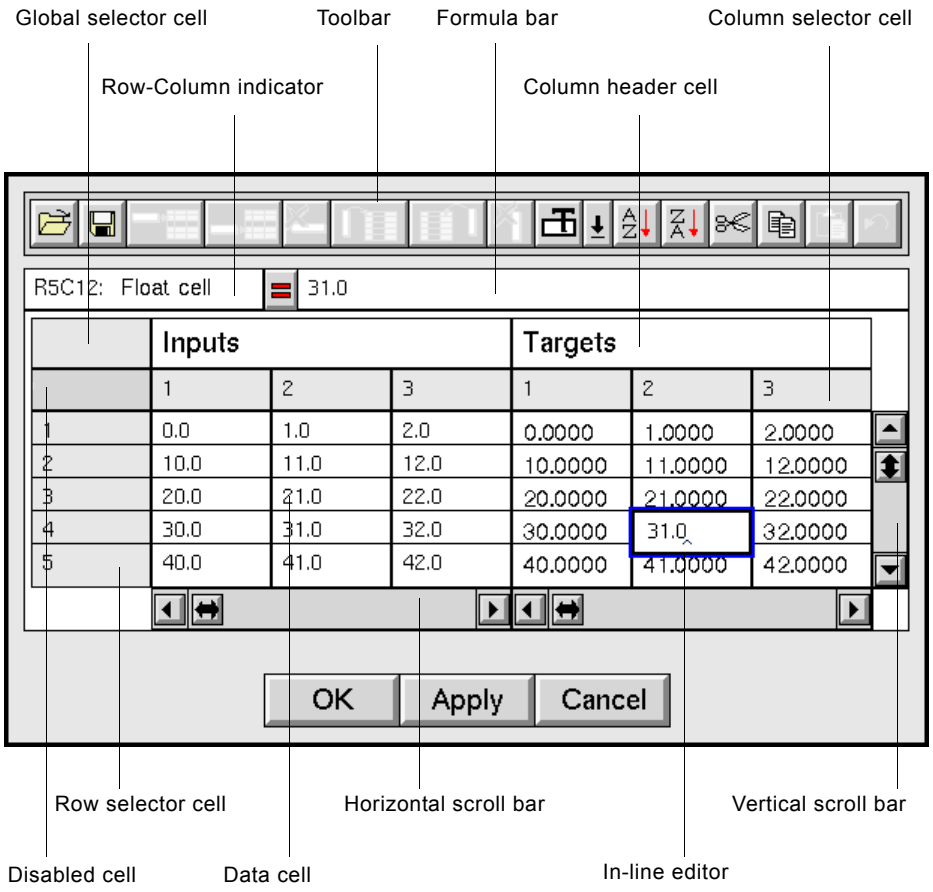
Using Spreadsheet Views

Describes the parts of a spreadsheet view and user interactions with the view.

Introduction

Using GXL, you can display and edit tabular data in familiar spreadsheet style. A GXL spreadsheet view consists of various types of cells and a set of tools that enable you to interact with the view and its data.

The parts of a typical view are illustrated in the following figure:



The following table describes the various types of cells:

Spreadsheet Item	Description
Column header cells	These cells provide a label for the contents of the column. Column headers can straddle more than one column.
Column/Row selector cells	These cells allow you to select entire Columns or rows. For information on using these cells, see Selecting Rows and Columns.
Global selector cell	This cell allows you to select the entire spreadsheet. For information on using this type of cell, see Selecting the Entire Spreadsheet.
Disabled cell	This cell fills areas of the spreadsheet. You cannot select or edit a disabled cell.
Data cells	These cells accept and display spreadsheet data of the following types: integers, floats, quantities, text, symbols, truth-values, and G2 values. For information on working with these cells, see Working with Data Cells.

The following table describes the spreadsheet tools:

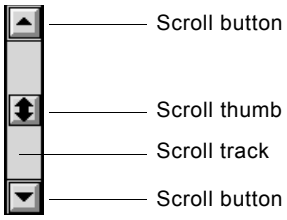
Spreadsheet Item	Description
Toolbar	The icon menu contains a set of tools that allow you to manipulate and change the spreadsheet view. For a complete listing of GXL spreadsheet tools and how to use them, see The Spreadsheet Toolbar.
Row-Column indicator	This tool serves as a navigation aid by displaying information related to the location and type of cells in the spreadsheet view.

Spreadsheet Item	Description
Formula bar	This tool allows you to calculate cell values, using mathematical operations, built-in G2 functions, or functions you define. For complete information on using the formula bar, see Using the Formula Bar.
Horizontal/Vertical Scroll bars	Scroll bars appear on the view whenever there are more rows or columns in the spreadsheet than are displayed on the view, enabling you to display any row or column. For information on using scroll bars, see Scrolling a View.
OK and Apply buttons	These buttons commit the edits made in the spreadsheet view. The OK button also closes the view. For information on using these buttons, see Saving Changes to the Spreadsheet View.
Cancel button	This button cancels the edit session and closes the view.
In-line editor	If a data cell is editable, an edit field opens when the cell is selected, allowing you to enter or change the data value in the cell. For information on using the editor, see Entering Values into Cells.

The specific layout of the view is configurable. For example, a view might not include row and column selector cells, or it might have multiple horizontal and vertical scroll areas.

Scrolling a View

Scroll bars appear on the view whenever there are more rows or columns in the spreadsheet than are displayed on the view. The following figure shows the parts of a scroll bar:



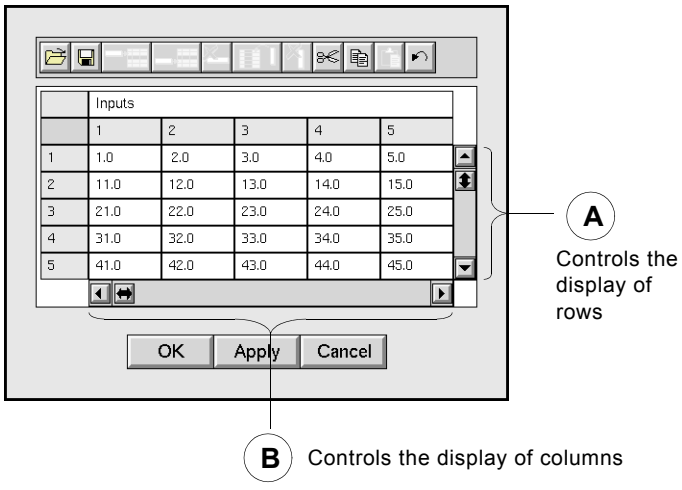
Vertical scroll bars control movement through rows and horizontal scroll bars control movement through columns. You can use the scroll bar four ways:

- Click on either scroll button to increment the scroll position by one row or column.
- Drag the scroll thumb to a new position to scroll by multiple rows or columns.

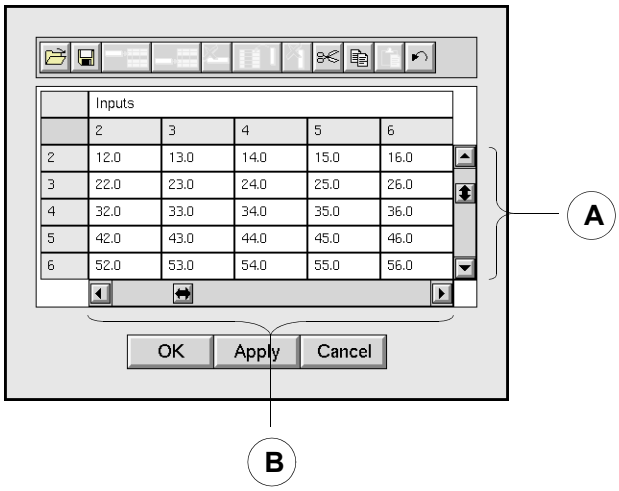
  When dragging the scroll thumb, the view does not scroll until you release the scroll thumb at the new position. If a row-column indicator is attached to the view, it displays the row or column corresponding to the current thumb position as you move the thumb, helping you to find the right position.
- Click directly on the scroll track to move one page forward or backwards.
- Hold either scroll button down to scroll continuously one row or column at a time. Scrolling stops when you release the mouse button or reach the limits of the scroll bar.

Note If you have more than one mouse button, use either button. G2 does not differentiate between mouse buttons.

A simple view with scroll bars looks like this:

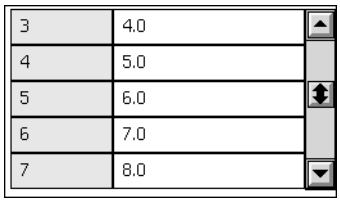


When you scroll downward using the vertical scroll bar (A), row 1 disappears and row 6 appears, and the cells appear to shift upward. The column selector cells remain visible. If you scroll to the right using the horizontal scroll bar (B), column 1 disappears and column 6 appears, and the cells appear to shift to the left, for example:

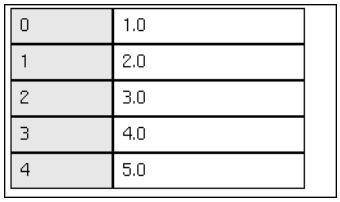


Dynamic Display of Scroll Bars

GXL adds and removes scroll bars dynamically if changing the dimensions of the spreadsheet makes scrolling necessary or renders it unnecessary. For example, if a view is configured to display five rows and the spreadsheet contains more than five rows, a scroll bar appears. If you then delete a number of rows, leaving five or fewer rows, the scroll bar disappears, as the following figure illustrates.



Scroll bar appears when the number of rows exceed the display configuration.



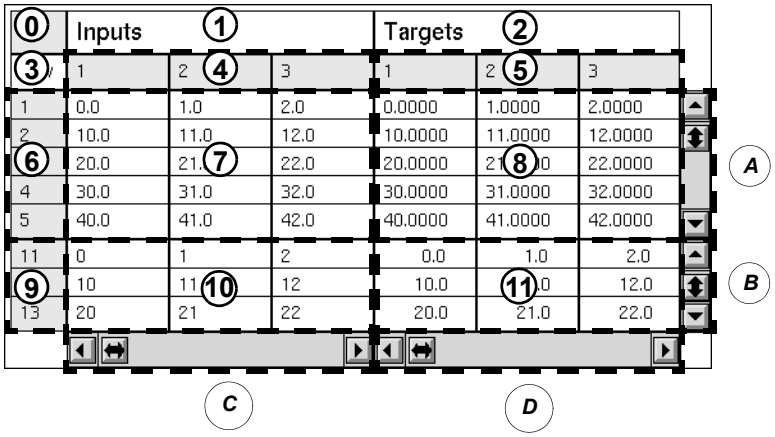
Scroll bar disappears when the number of rows are equal to or less than the display configuration.

Multiple Scroll Bars

When a view displays multiple horizontal and/or vertical scroll bars, Each scroll bar has a definite row or column range that determines how far you can scroll. You cannot scroll beyond the first row/column or the last row/column of the cell groups associated with the scroll area.

For example, in the following illustration scroll bars A and B control vertical movement through rows, while scroll bars C and D control horizontal movement through columns, where:

- A controls movement through the rows of Inputs and Targets in cell groups 6, 7, and 8.
- B controls movement through the rows of Inputs and Targets in cell groups 9, 10, and 11.
- C controls movement through the columns of Inputs (cell groups 4, 7, 10).
- D controls the movement through the columns of Targets (cell groups 5, 8, 11).



GXL does not permit multiple horizontal or vertical scroll bars per cell group, or scroll bars that span multiple cell groups.

Scrolling in a Telewindows Session

For the best scrolling performance under Telewindows, click on the scroll track to page forward/backward or move the scroll thumb.

Selecting Areas on the Spreadsheet View

When you want to operate on a cell or group of cells, you must select the cells that will be the target of your action. You can select:

- A single cell or a rectangular area of cells.

- A single row or column.
- A range of rows or columns.
- The entire spreadsheet.

GXL currently supports only *contiguous selection*, that is, continuous ranges of cells, rows, or columns. You cannot, for example, select rows 1, 3, and 5, but you can select rows 1 through 5.

Different types of cells respond differently to mouse gestures. Data cells, which include value, quantity, integer, float, text, truth-value, and symbol data types all behave similarly. Special cells, such as row and column selector cells, column headers, and global selector cells, have individual behaviors and are different than data cells.

The selection behavior of a cell also depends on two properties of the cell group to which a cell belongs: its *selectability* and its *editability*.

- If a cell is neither editable nor selectable, clicking or dragging the mouse on the cell does nothing.
- If a cell is selectable but not editable, you can select the cell, but you can not edit the contents of the cell.
- If a cell is selectable and editable, certain actions select the cell, and other actions open the in-line editor.

All cells within a cell group have the same selectability and editability properties.

Selecting Data Cells

In GXL, cells you select either appear as white text on a black background or appear with a wide blue border. The wide blue border indicates the in-line editor is active, enabling you to make changes to the contents of the cell. Only one cell in a G2 window can have an open editor.

To select a single data cell:

➔ Click the cell.

If the cell is selectable but not editable, it appears as white text on a black background:

0.478	0.101	0.564
0.05	0.508	0.23
0.738	0.316	0.963

A selected cell

If the cell is selectable and editable, the in-line editor opens in the cell:

0.478	0.101	0.564
0.05	0.50827798	23
0.738	0.316	0.963

A selected cell with active in-line editor

Selecting Multiple Cells

You can select multiple cells by using either of the following techniques:

- Dragging the mouse.
- Using the Shift key.

Using the Mouse to Select a Range of Cells

Note Dragging the mouse works only if you are selecting a range of cells that are currently visible in the view.

To select multiple cells by dragging the mouse:

1. Point to the cell at the start of the selection.
2. Press and hold down the mouse button while dragging the pointer to the cell at the end of the selection.
3. Release the mouse button to mark the end of the selection.

This selects all the cells in the rectangular area delimited by the starting and ending cells.

Using the Shift Key to Select a Range of Cells

To select multiple cells by using the Shift key:

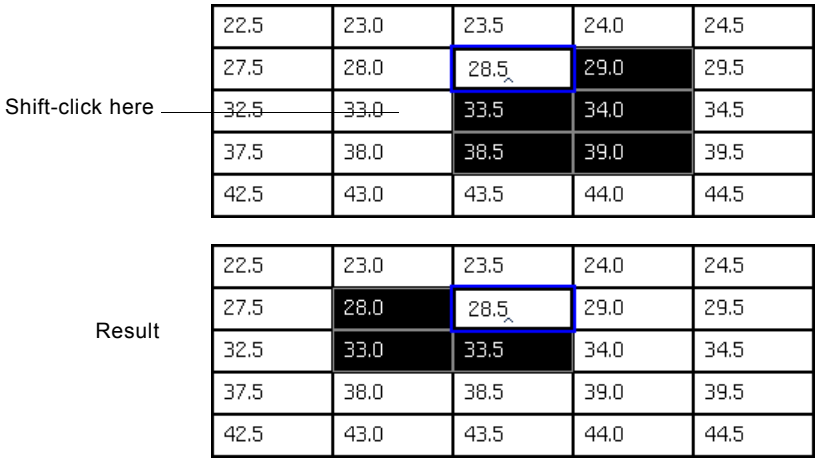
1. Click the cell that begins the selection to mark the start.
2. Press and hold down the Shift key.
3. Click the cell that ends the selection and release the Shift key.

This **shift-click** technique selects all the cells in the rectangular area delimited by the two cells. You may scroll the view after selecting the first cell, thus creating a selection area that is larger than the original view area.

Extending Cell Selections

If you already have selected multiple cells and you extend the selection by using the Shift key, the resulting selection depends on whether the in-line editor is open.

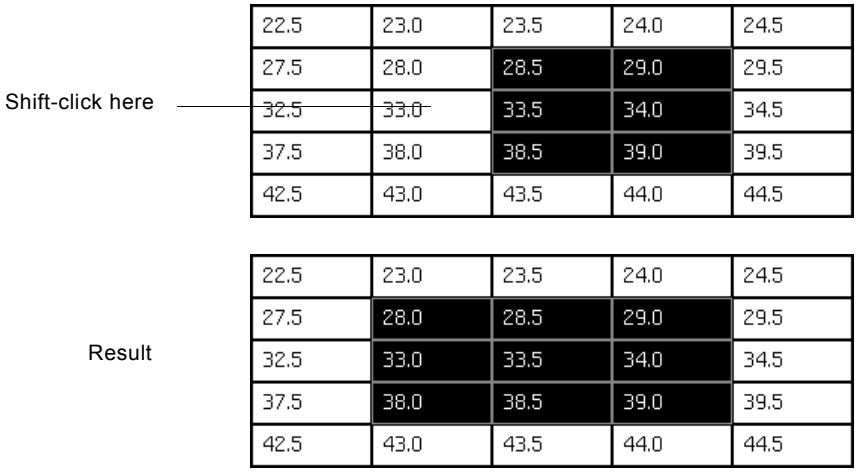
If the editor is open, selecting another cell with the Shift key results in a rectangular area of selected cells delimited by the edit cell and the cell where you clicked; for example:



22.5	23.0	23.5	24.0	24.5
27.5	28.0	28.5	29.0	29.5
32.5	33.0	33.5	34.0	34.5
37.5	38.0	38.5	39.0	39.5
42.5	43.0	43.5	44.0	44.5

22.5	23.0	23.5	24.0	24.5
27.5	28.0	28.5	29.0	29.5
32.5	33.0	33.5	34.0	34.5
37.5	38.0	38.5	39.0	39.5
42.5	43.0	43.5	44.0	44.5

If the editor is not open, selecting another cell with the Shift key results in a rectangular area of selected cells that encompasses the previously selected area and the cell where you clicked; for example:



22.5	23.0	23.5	24.0	24.5
27.5	28.0	28.5	29.0	29.5
32.5	33.0	33.5	34.0	34.5
37.5	38.0	38.5	39.0	39.5
42.5	43.0	43.5	44.0	44.5

22.5	23.0	23.5	24.0	24.5
27.5	28.0	28.5	29.0	29.5
32.5	33.0	33.5	34.0	34.5
37.5	38.0	38.5	39.0	39.5
42.5	43.0	43.5	44.0	44.5

Selecting Rows and Columns

When row and column selector cells are a part of the spreadsheet view, you can select rows or columns of cells. Row and column selector cells are normally light

gray and numbered consecutively. The numbering usually begins with 1 for a cell group.

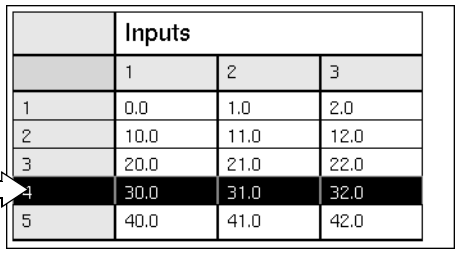
Column Header Cells

Column header cells, which provide a label for the contents of the column, can straddle more than one column. If you select or extend a range of columns that contains header cells, the header cells are not included in the selection. However, if a column header is editable, you may select and edit its contents.

To select a row of cells:

➔ Click the row selector cell of the row you want to select, for example:

Click here to select the entire row.

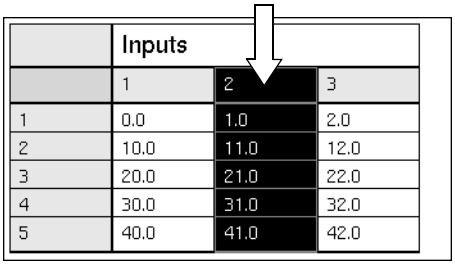


	Inputs		
	1	2	3
1	0.0	1.0	2.0
2	10.0	11.0	12.0
3	20.0	21.0	22.0
4	30.0	31.0	32.0
5	40.0	41.0	42.0

To select a column of cells:

➔ Click the column selector cell of the column you want to select, for example:

Click here to select the entire column.



	Inputs		
	1	2	3
1	0.0	1.0	2.0
2	10.0	11.0	12.0
3	20.0	21.0	22.0
4	30.0	31.0	32.0
5	40.0	41.0	42.0

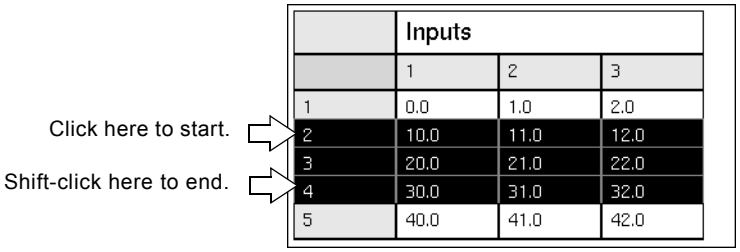
Selecting Multiple Rows or Columns

You can select multiple rows or columns by using the Shift key. If there is an open editor, the edit cell will define one edge of the selected region. You cannot simultaneously select a mixture of rows and columns.

To select multiple rows or multiple columns:

1. Click the row or column selector cell that marks the start of the selection.
2. Shift-click the row or column selector cell that marks the end the selection.

The following example shows the result of selecting multiple rows:



To select all of the rows or columns in a view:

1 Click the first row or column selector cell.

2 Shift-click the last row or column selector cell.

Selecting the Entire Spreadsheet

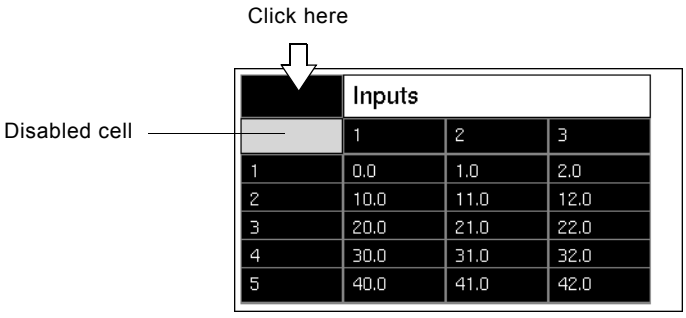
If the spreadsheet contains a global selector cell, you can use this cell to select all of the rows and columns of cells in the spreadsheet. The global selector cell is usually in the upper left hand corner of the view.

Disabled Cells

Some spreadsheets use a disabled cell to fill an area of the spreadsheet. These cells are usually solid light gray. You cannot select or edit a disabled cell.

To select the entire spreadsheet using the global selector:

➔ Click the global selector cell.



Notice that the global selector cell selects all row and column selector cells and all data cells. Although the view display does not indicate it, the global selector includes any disabled cells and column header cells in the selection.

Navigating the Spreadsheet

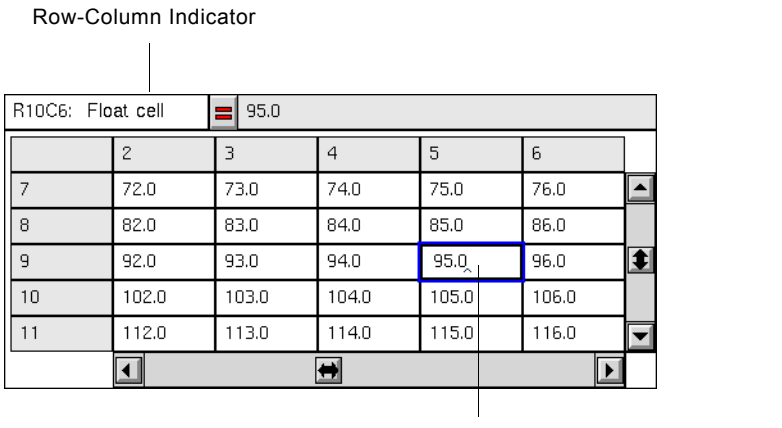
The **row-column indicator** is a GXL tool that displays information related to the location and type of spreadsheet cells. The row/column numbers in the row-column indicator are *absolute* spreadsheet coordinates, which may or may not match the numbered display of row and column selector cells.

The row-column indicator displays the coordinate location of a single cell in the format:

R*w*C*x*

where *w* and *x* represent the absolute row and column location.

In the following example, the coordinate location of the active cell in this spreadsheet view is row 9, column 5. However, the absolute spreadsheet coordinate location is row 10, column 6 (R10C6).



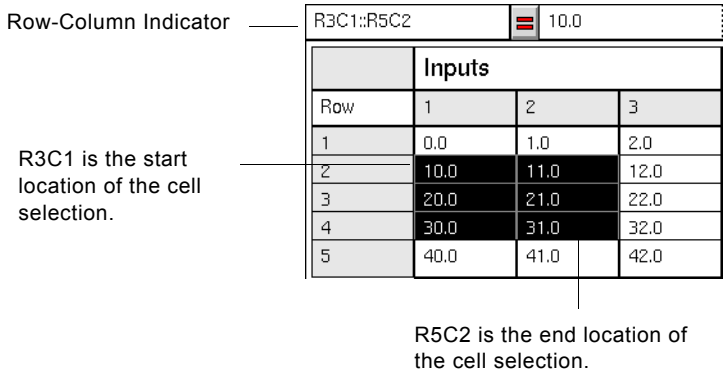
The row-column indicator also displays the type of data cell, which in this example, is a float cell.

When you select a range of cells, the row-column indicator displays the current selection limits in the format:

R*w*C*x*::R*y*C*z*

where *w*, *x*, *y*, and *z* represent row and column numbers.

For example, the selected range of cells in the following figure begins with the cell in row 2, column 1, and ends with the cell in row 4, column 2. However, the row-column indicator specifies the corresponding absolute spreadsheet coordinates of this selection, which are R3C1 and R5C2, respectively.



Working with Data Cells

Each data cell in a spreadsheet has a specific data type, which dictates the valid contents of the cell. The cell group to which the data cell belongs determines its data type. There are seven types of data cells:

- **Float cell**
- **Integer cell**
- **Quantity cell**
- **Symbol cell**
- **Text cell**
- **Truth-value cell**
- **Value cell**

Quantity cells can accept floats and integers as input. Value cells can accept any valid data type as input. A cell can also be empty and contain no value.

How Data Cell Values are Displayed

The value you see displayed in a cell is a formatted representation of the data actually stored in the spreadsheet. The formatting depends on the type of data value, as well as the type of cell in which the value is displayed. The following table describes the rules of formatting values for display:

Value	Format
integer	Displayed as whole numbers. Example: 374
float	By default, floats whose absolute values are greater than 1.e5 or less than 1.e-4 (excepting 0.0) are displayed in exponential format (d.ddde±ddd); otherwise, they are displayed in ddddd.ddd format. Example: 35.025
symbol	Displayed in all upper case letters. The first character is an alphabetic character. Example: SMD34-TRUCOLOR
truth-value	Displayed in lower case letters. Example: false
text	In text cells, text is displayed without enclosing quotes. Example: Satellite Office In value cells, text is displayed with enclosing quotes. Example: “Satellite Office”

If the formatted value is too large to fit into the cell, the value is truncated. Truncations are indicated by ellipses (...). Truncations do not affect the value actually stored in the spreadsheet. You can see the full value by activating the in-line editor on the cell.

When you enter values, G2 validates the input based on the type of data and cell type. For information on data validation, see Validating Data Input.

Entering Values into Cells

If a cell is selectable and editable, the in-line editor opens an edit field on the cell, enabling you to enter or change its value.

To enter a value into an editable cell:

1 Click the cell you want to edit.

An edit field opens on the cell and displays the full contents of the cell. The cursor position, represented by a carat symbol (^), is initially at the end of the edit field. For example:

0.478	0.101	0.564
0.05	0.50827798^23	
0.738	0.316	0.963

2 Enter data by typing keyboard characters.

3 Finish your edit by pressing enter to accept your input.

Editing Operations

The standard G2 editor keyboard shortcuts work when entering data into a cell; for example, Control - A aborts the edit, Control - J starts a new line. While the editor is open, pressing Control - ? displays the valid keyboard shortcuts.

You may scroll the view while you are editing a cell. However, if you scroll the edit field out of view, it cannot receive keyboard input until you scroll it back into view.

You can move the edit cursor to another position within the field by using either of the following techniques:

- Point to the position and click the mouse.
- Use the left and right arrow keys.

Creating an Empty Cell

If a cell has a value, you can make it an empty cell by editing the cell's contents.

To create an empty cell:

➔ Select the cell and use the backspace key to delete the value.

Note In text cells, there is no visual distinction between empty cells and cells containing the empty string.

Ending an Edit

The following actions end the edit in the selected cell:

Action	Results
Press Enter	Pressing Enter finishes the edit and opens the editor in the cell below the edited cell within the cell group.
Press the Tab key	Pressing the Tab key finishes the edit and opens the editor in the cell to the right of the edited cell within the cell group.
Select another cell	Selecting another cell finishes the edit and moves the editor to the new selected cell.

Validating Data Input

When you end the edit, G2 validates the new entry and might perform type coercion to make your data input conform with the cell type. The rules for validation and coercion by cell type are:

Cell Type	Validation Rules and Coercion
quantity float integer	Accepts any entry beginning with a numeric character, and converts the entry to a quantity, using the quantity function. • If the result is an integer and the cell is a float cell, convert the result into a float. • If the result is a float and the cell is an integer cell, convert the result to an integer, using the round function.
symbol	Accepts any entry that can be converted into symbols, using the symbol function.
truth-value	Accepts an entry if it is either true or false. Also valid, the initial characters t(rue) and f(alse).

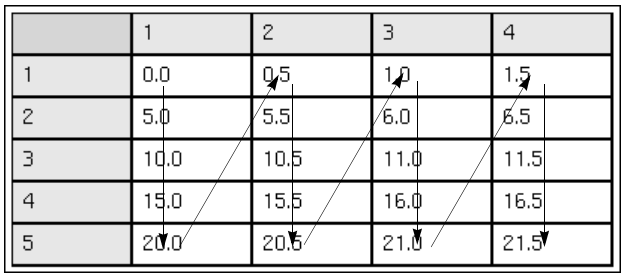
Cell Type	Validation Rules and Coercion
text	Accepts any entry, because any series of characters makes an acceptable string.
value	Converts an entry as follows: • If the entry begins and ends with quote marks, remove the beginning and ending quotes and store it as a text string. • If the entry begins with a numeric digit, convert it to a float or an integer, using G2's quantity function. • If the entry is true or false, store it as a truth value. • Otherwise, convert the entry to a symbol. If the entry cannot be converted into a symbol successfully, the entry is not accepted.

Moving the Editor within a Cell Group

When the in-line editor is active, you can press the Enter key or the Tab key to advance the editor. As you advance the editor, the view scrolls, if necessary, to keep the edit cell in the view. When you reach the last cell in the cell group, pressing Enter or Tab does not advance the editor.

Using the Enter Key to Move the Editor

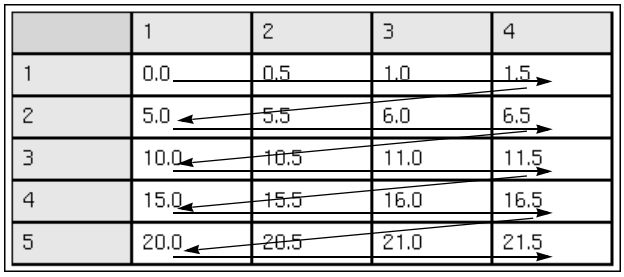
Pressing the Enter key moves the editor to the cell directly below the current cell. When you reach the last row of a cell group, pressing the Enter key advances the editor to the first row of the next column of the cell group, as the following figure illustrates:



How the Enter key advances the editor.

Using the Tab Key to Move the Editor

Pressing the Tab key advances the editor to the cell directly to the right of the current cell. When you reach the last column of a cell group, pressing the Tab key advances the editor to the first column of the next row of the cell group, as the following figure illustrates:



How the Tab key advances the editor.

Saving Changes to the Spreadsheet View

OK

Apply

During an edit session, GXL stores your input in a buffer area, as discussed in GXL Edit Sessions. When you finish updating values in the cells of the spreadsheet view, you can permanently save these changes to the spreadsheet, using the OK or Apply button. Selecting either of these buttons:

- Changes the dimension of the original spreadsheet to the current number of rows and columns in the spreadsheet view.
- Updates the values in the spreadsheet from the buffer.

Choosing the OK button also closes the view, while choosing the Apply button leaves the view display open.

Cancel

If you want to close the view without saving the changes, use the Cancel button.

To save data updates to the spreadsheet:

➔ Click OK or Apply.

Note If you use Apply to save your edits and later use Cancel, the data reverts to the most recent Apply.

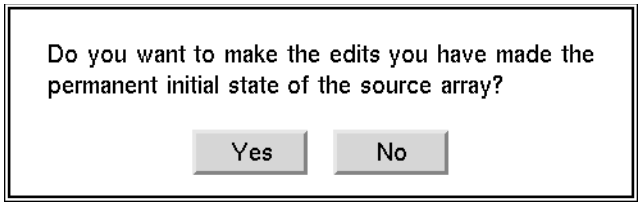
To close the view without saving the updates:

➔ Click Cancel.

Making Your Edits the Initial Value of an Array

G2 arrays can have initial values that set the array values when G2 is started. If you want your edits to persist through a G2 reset or restart, you must make the values you have entered the initial values of the spreadsheet.

Clicking OK or Apply when editing a value array displays the following dialog:



- If you choose Yes, GXL copies the values from the spreadsheet into the array as the initial values. The array contains these initial elements the next time you reset or restart G2.
- If you choose No, the values from the spreadsheet become the current values of the array, but the initial values of the array are not affected.

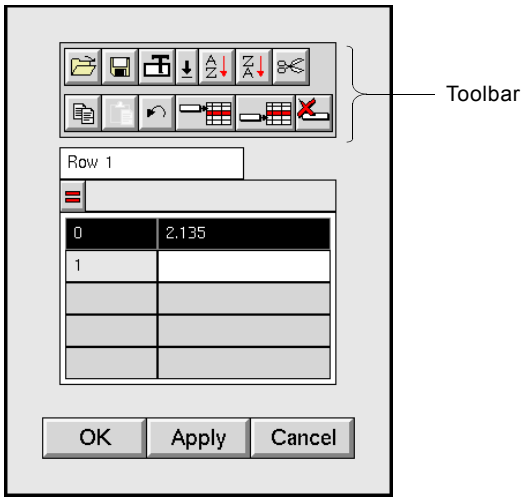
Note Because the persistence of values in a list are set by the list-is-permanent attribute of the list, the dialog offering to make your edits the initial values does not appear when ending an edit session on a list.

The Spreadsheet Toolbar

GXL spreadsheet views usually display a toolbar whose buttons provide an easy way to:

- Change the dimension of the spreadsheet.
- Load and save data to and from text files.
- Change the colors of cells.
- Sort areas of the spreadsheet.

The toolbar may contain the full set of GXL spreadsheet tools or an appropriate subset of these tools. The toolbar can also contain custom tools that meet specific needs of the GXL view display. Normally, the toolbar is located above the data area; for example:

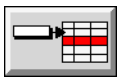
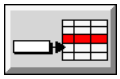
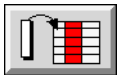
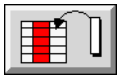


Buttons in the toolbar have two states: enabled and disabled. In the disabled state, the icon symbol for the button appears in light gray, and you cannot select the button. When the button is enabled, the icon symbol is readily visible and you can select the button. The state of the button reflects the selection of cells in the spreadsheet. For example, the Delete Row button is only enabled when one or more rows are selected.

Each button on the toolbar has a popup help text, which appears when you hold the mouse button down over the button.

The default GXL toolbar buttons are:

Toolbar Button	Function
	Loads data from a file into the selected area.
	Saves the current selection to a file.
	Applies the chosen color pattern to selected cells.
	Specifies the text, background, and border color of cells.

Toolbar Button	Function
	Sorts the rows of the selection in ascending order according to the contents of a key column.
	Sorts the rows of the selection in descending order according to the contents of a key column.
	Cuts and transfers the current selection to the clipboard.
	Copies the contents of the current selection to the clipboard.
	Copies the contents of the clipboard into the selected cells.
	Reverses the last operation.
	Inserts an empty row above the currently selected row.
	Inserts an empty row below the currently selected row.
	Deletes the selected rows.
	Inserts an empty column before the currently selected column.
	Inserts an empty column after the currently selected column.
	Deletes the selected columns.

Saving Spreadsheet Data to a File

You use the Save to File button to save spreadsheet data to a file. This button is active whenever any cell or area of the view is selected.

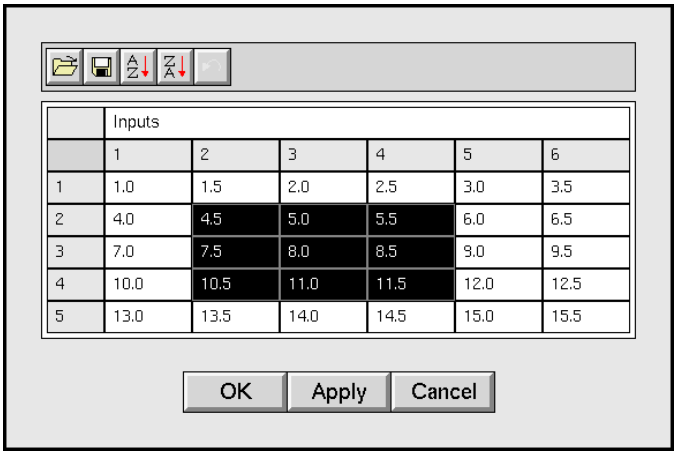
You can save a rectangular area of data or all of the data in the spreadsheet view. For information on selecting data, see Selecting Areas on the Spreadsheet View.

The data is saved in comma-separated value (csv) format.

To save a selected area of the spreadsheet data:

1 Select the rectangular area containing the data you want to save.

For example:

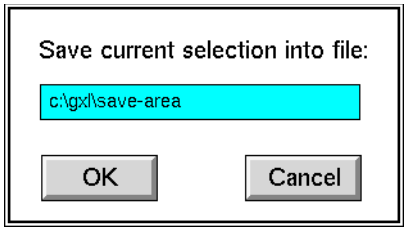


Tip To save all data cells in the view, select the first data cell in the upper left corner and shift-click the last data cell in the lower right corner of the spreadsheet view.

2 Click the Save to File button.

3 Specify the pathname of the file to which you want to save the data in the edit field of the dialog that appears.

For example:



4 Press Enter to accept the filename.

5 Click OK to save the data to the file.

Here are the contents of the file:

```
4.5,5.0,5.5
7.5,8.0,8.5
10.5,11.0,11.5
```

How GXL Saves Selections to a File

The data that GXL saves to a file depends on the area selected in the spreadsheet view. The previous example showed the result of saving a selected area of data cells. The following examples show the results of saving selected rows, columns, and the entire view.

Saving Selected Rows

If you use row selectors to select entire rows of data and save the selection to a file, the first data element of each row saved is the contents of the row selector cell. For example, suppose you select rows 2 through 5 to save:

	Inputs					
	1	2	3	4	5	6
1	1.0	1.5	2.0	2.5	3.0	3.5
2	4.0	4.5	5.0	5.5	6.0	6.5
3	7.0	7.5	8.0	8.5	9.0	9.5
4	10.0	10.5	11.0	11.5	12.0	12.5
5	13.0	13.5	14.0	14.5	15.0	15.5

Here are the contents of the file:

```
2,4.0,4.5,5.0,5.5,6.0,6.5
3,7.0,7.5,8.0,8.5,9.0,9.5
4,10.0,10.5,11.0,11.5,12.0,12.5
```

Saving Selected Columns

If the view contains both column headers and column selectors, using column selectors to select data also saves the contents of the column headers. For example, suppose you select columns 2 through 4:

	Inputs					
	1	2	3	4	5	6
1	1.0	1.5	2.0	2.5	3.0	3.5
2	4.0	4.5	5.0	5.5	6.0	6.5
3	7.0	7.5	8.0	8.5	9.0	9.5
4	10.0	10.5	11.0	11.5	12.0	12.5
5	13.0	13.5	14.0	14.5	15.0	15.5

Here are the contents of the file:

```
"Inputs","Inputs","Inputs"
2,3,4
1.5,2.0,2.5
4.5,5.0,5.5
7.5,8.0,8.5
10.5,11.0,11.5
13.5,14.0,14.5
```

Using the column selectors to select entire columns of data saves the contents of the column headers, as well as the column selectors, along with the data. Notice that even though the column header straddles the columns in the view, the header text is repeated for each column saved.

Saving the Entire View

If the global selector is available, you can use this cell to select all cells of the view. For example:

	Inputs					
	1	2	3	4	5	6
1	1.0	1.5	2.0	2.5	3.0	3.5
2	4.0	4.5	5.0	5.5	6.0	6.5
3	7.0	7.5	8.0	8.5	9.0	9.5
4	10.0	10.5	11.0	11.5	12.0	12.5
5	13.0	13.5	14.0	14.5	15.0	15.5

Here are the contents of the file:

```
,"Inputs","Inputs","Inputs","Inputs","Inputs","Inputs"
,1,2,3,4,5,6
1,1.0,1.5,2.0,2.5,3.0,3.5
2,4.0,4.5,5.0,5.5,6.0,6.5
3,7.0,7.5,8.0,8.5,9.0,9.5
4,10.0,10.5,11.0,11.5,12.0,12.5
5,13.0,13.5,14.0,14.5,15.0,15.5
```

Notice the first two rows of saved data: the first data element of each row is a comma (,). This indicates that the first cell is empty. The first cell of the first row selected is the global selector cell, and the first cell of the second row is a disabled cell. Neither of these cells contain data values.

Loading Data from a File into a Spreadsheet View

You use the Load from File button to load the contents of a file into a selected area of a spreadsheet. This button is active whenever any cell or area of the view is selected.

The file must be in the comma-separated values (csv) format, where:

- Values on the same line are separated by commas, for example:

  ```
  11.0,12.0,13.0
  ```

- Quotation marks enclose text strings, and embedded quotes within text strings are doubled, for example, the string abc “def” gh is represented as:

  ```
  "abc ""def"" gh"
  ```

- Empty cells are represented by consecutive commas, for example:

  ```
  20.0,21.0,,23.0,24.0,25.0,,27.0,28.0,29.0
  ```

- Lines may contain different numbers of values, for example:

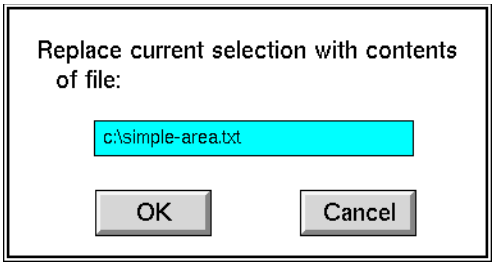
  ```
  30.0,31.0,32.0,33.0,34.0,35.0,36.0
  11.0,13.0,15.0,17.0,19.0
  20.0,21.0,,23.0,24.0,25.0,,27.0
  ```

To load data from a file:

1. Select the area of the spreadsheet that you want to load with data from the file.
2. Click the Load from File button.

3 Specify the pathname of the file containing the data you want to load in the edit field of the dialog that appears.

For example:



4 Press Enter to accept the filename.

5 Click OK to load the data from the file.

Considerations

When using the Load from File button to load data into a spreadsheet:

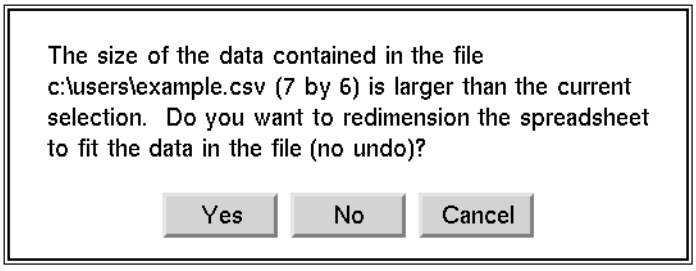
- Type checking is performed.
- No undo exists for this operation.

How GXL Loads the Data into the Spreadsheet

GXL reads the file to determine its dimensions. During the loading of the file, a progress dialog appears, which allows you to monitor the progress of the file load and cancel the file load, if desired.

If the data in the file fits into the selection, the data is entered by row into the selected area. Any area of the selection that does not receive new values from the file are cleared.

If the dimensions of the data in the file exceed the selection area, the following dialog appears:

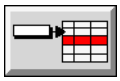


Your options are:

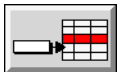
- Click Yes to insert additional rows and columns after the last row and/or column of the selection.
- Click No to load only the file data that fits into the selected area. Thus, some data in the file will not be loaded.
- Click Cancel to cancel the load operation.

Adding and Deleting Rows on the Spreadsheet

You use the Insert Row Before, Insert Row After, and Delete Row buttons to add and remove rows of cells on the spreadsheet view as follows:



The Insert Row Above button inserts a single row before the currently selected row.



The Insert Row Above button inserts a single row before the currently selected row

The Delete Row button deletes the currently selected rows.

These buttons are active whenever one or more rows are selected. You can reverse the actions of these buttons with the Undo button.

When adding rows to the view, GXL renumbers the rows, if necessary.

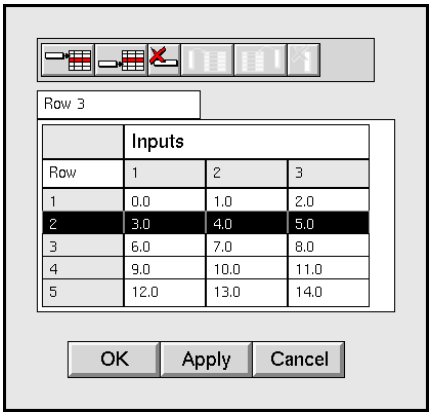
If a cell group boundary exists where you want to add a row, remember: GXL adds the row to the spreadsheet is in the same cell group as the currently selected row.

To add a row to a view:

1 Click the row where you want to add one or more new rows.

The cells in the selected row appear as white text on black background.

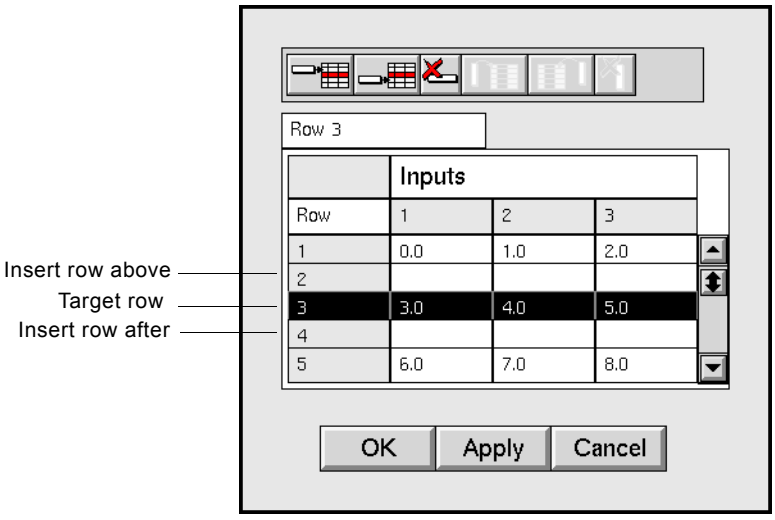
For example:



2 Do one of the following:

- Click Insert Row Above to add a new row before the selected row.
- Click Insert Row After to add a new row after the selected row.

The new row is in the same cell group as the selected row, and its cells are initially empty. If necessary, GXL renumbers the rows; for example:



Deleting Rows

Deleting a row removes the row and its data from the view. You can delete one or more contiguous rows at a time. If necessary, GXL renumbers the remaining rows.

Note If you delete all of the rows in a cell group, you cannot add them back interactively.

To delete a row on a view:

1 Select the row you want to delete.

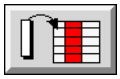
2 Click the Delete Row button to remove the row from the view.

To make the deletion permanent:

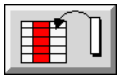
➔ Click OK or Apply.

Adding and Deleting Columns on a Spreadsheet

You use the Insert Column before, Insert Column After, and Delete Column buttons to add and remove columns in the spreadsheet as follows:



The Insert Column Before Selection button inserts a single column before the currently selected column.



The Insert Column After Selection button inserts a single column after the currently selected column.

The Delete Column button deletes the currently selected columns.

These buttons are active whenever one or more columns are selected. You can reverse the action of these buttons with the Undo button.

When adding columns to the view, GXL renumbers the columns, if necessary.

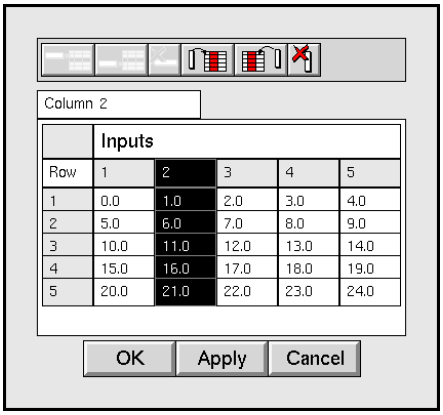
If a cell group boundary exists where you want to add a column, remember: GXL adds the column to the spreadsheet in the same cell group as the currently selected column.

Adding Columns

To add a column to a view:

1 Click the column where you want to add one or more new columns.

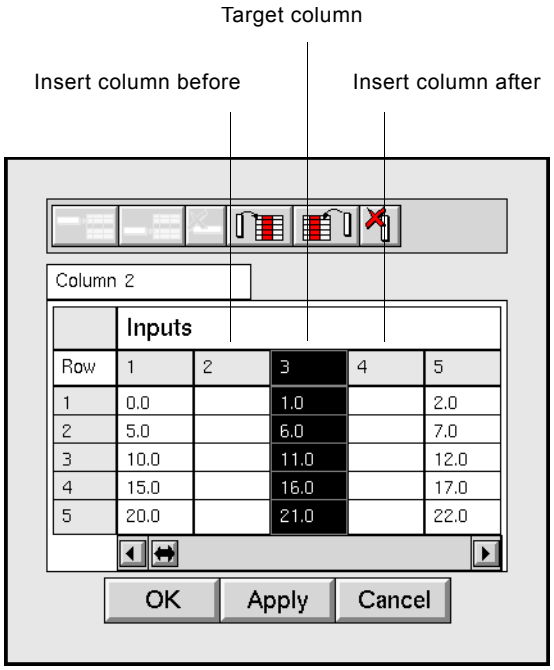
The cells in the selected column appear as white text on black background, for example:



2 Do one of the following:

- Click Insert Column Before to add a new column before the selected column.
- Click Insert Column After to add a new column after the selected column.

The new column is in the same cell group as the selected column, and its cells are initially empty. If necessary, GXL renumbers the columns. For example:



Deleting Columns

Deleting a column removes the column and its data from the view. You can delete one or more contiguous columns at a time. If necessary, GXL renumbers the remaining columns.

To delete columns on a view:

1 Select the columns you want to delete.

2 Click the Delete Column button to remove the columns from the view.

To make the deletion permanent:

➔ Click OK or Apply.

Changing the Color Patterns of Cells

You use these two buttons to change the color pattern of selected cells:

The Change Cell Color button applies the color pattern you have chosen to the selected cells.

The Specify Cell Color button specifies the desired text, background, and border colors of cells.

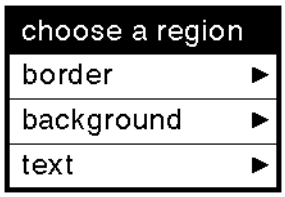
You can specify cell color patterns at any time. The Change Cell Color button is active whenever one or more cells are selected. You can reverse the action of the Change Color button with the Undo button.

Caution Never use white text on black background as a color combination in your spreadsheet. This color pattern is reserved for selected cells.

To specify the color pattern:

1 Click the Specify Color button.

A menu appears with three choices that correspond to the three color regions of a spreadsheet cell:



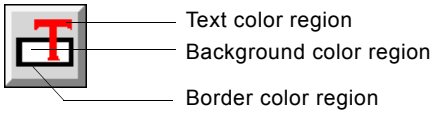
2 Choose the region of the cell whose color you want to change.

The G2 standard color palette appears.

3 Choose a color from the color palette.

The selected color becomes the color of the corresponding region on the Change Color button.

For example, if you select red as the text color, the "T" in the icon of the Change Color button changes to red:



To apply the color pattern to one or more cells:

➔ Choose the desired cells and click the Change Cell Color button.

Cutting, Copying, and Pasting

You use the Cut, Copy, and Paste buttons to transfer cell contents to and from spreadsheet cells, using a buffer area known as the GXL clipboard.

The Cut button copies the data values and color patterns of the selected cells to the clipboard and clears the contents of the current selection, leaving the selected cells empty.

The Copy button copies the data values and color patterns of the selected cells to the clipboard.

The Paste button transfers the contents of the clipboard to the selected cells, overwriting the current contents.

Selections can include a single or multiple cells, one or more rows or columns, or the entire spreadsheet.

In addition to cutting and pasting values within the same spreadsheet, you can use the clipboard to transfer data between spreadsheets. You can transfer values cut from one view to another view on the same window. There is one GXL clipboard per G2 window.

Note When you reset G2, the contents of the clipboard are cleared.

These buttons are active whenever one or more cells are selected. You can reverse the action of the Cut and Paste buttons with the Undo button.

Cutting Data from the Spreadsheet

To remove data from the spreadsheet:

1 Select the cell or cells whose contents you want to remove.

2 Click the Cut button.

GXL removes the contents of the cells and places the contents on the clipboard.

Copying Data from the Spreadsheet

To copy data from the spreadsheet:

1 Select the cell or cells whose contents you want to copy.

2 Click the Copy button.

GXL places a copy of the contents on the clipboard.

Pasting Cut or Copied Data

When pasting cut or copied data selections:

- The cell types and data types must be compatible.
- The dimensions of the data must be exactly the same.

GXL displays a warning message if it cannot successfully complete a paste operation.

Data Compatibility

When you attempt to paste data into a spreadsheet, the formats of the data you paste must be compatible with the destination cells. You cannot, for example, paste a text value into an integer cell. However, if you attempt to paste an integer into a text cell, G2 converts the integer into a text string and the paste will succeed.

If the cell type or data type on the clipboard is incompatible with the target selection, a dialog similar to the following appears:

Data Dimensions

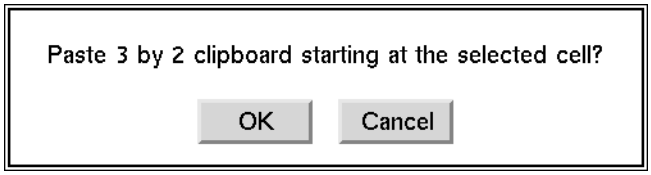
Data dimensions on the clipboard must match the target dimensions. If the data dimensions of the paste selection are incompatible with the target dimensions, a dialog similar to the following appears:

Clicking OK cancels the paste operation.

To paste the cut or copied data:

1. Select the cells where you would like the clipboard data to be inserted.
2. Click the Paste button.

If you select a single target cell and the contents on the clipboard is larger, a dialog similar to the following appears:



If you click OK, the selected cell becomes the upper left hand corner of the paste region. GXL copies the contents of the clipboard into the view, beginning with the selected cell.

Reversing the Last Toolbar Operation

Most actions performed by toolbar buttons and the formula tool are reversible. You use the Undo button to reverse the last action you performed on the spreadsheet. You can undo these actions:

- Adding rows before or after a selected row.
- Deleting rows.
- Adding columns before or after a selected column.
- Deleting columns.
- Sorting in ascending or descending order.
- Cutting a selection.

- Pasting a selection.
- Editing a cell.
- Applying a formula.

The Undo button is active only when an action can be undone.

You cannot undo selecting cells, scrolling, or tabbing the edit field.

Sorting Spreadsheet Data

You use the Sort Ascending button and Sort Descending button to sort the rows in a selection so that values in a key column appear in ascending or descending order.

If the selection includes more than one column, the key column is determined as follows:

- If an editor is open on any cell, the column containing the open editor becomes the key column.
- Otherwise, the left-most column of the selection becomes the key column.

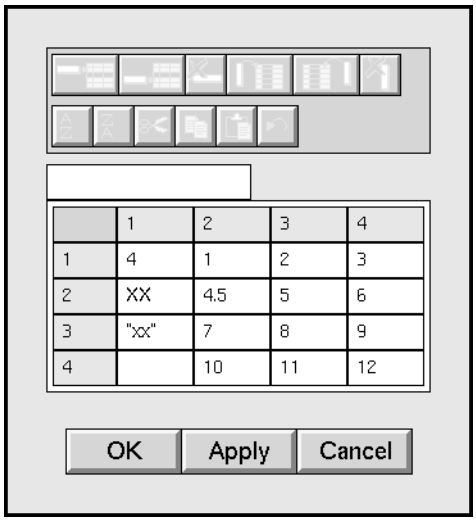
When performing the sort, GXL sorts the values in the key column in ascending or descending order, depending on which sort button you choose. The values that are not in the key column undergo the same permutations as the key column. Thus, the individual rows of the selection are unchanged by the sort, but they appear in a different order.

When you sort in ascending order, the key column is sorted in the following order:

1. Empty cells
2. Quantities (floats, integers), in ascending numerical order
3. Truth-values, with false before true
4. Symbols, in alphabetical order
5. Texts, sorted in ascending order as determined by the G2 greater than (>) operation

Sorting in descending order reverses this order.

For example, consider the following view whose data cells accept any value (value-cell type):



To sort data in ascending order:

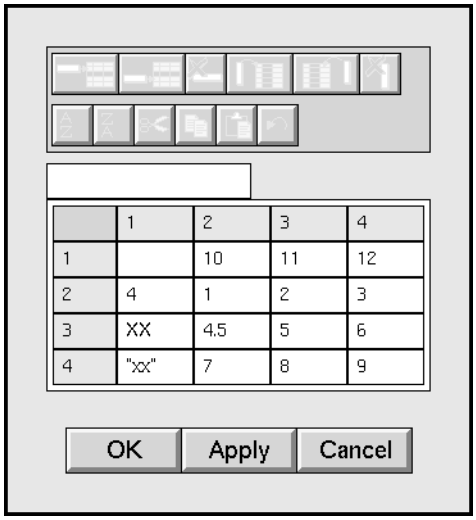
1. Click column 1 to select it.

   The selected column becomes the key sort column.

2. Shift-click column 4 to select all the data cells.

3. Click the Sort Ascending button.

Here are the results of the sort:



Other Operations on Views

Moving or Transferring Views

You can use Main Menu > Operate on Area tool to:

- Move a view to a new position on its workspace.
- Transfer a view between workspaces.

When using Operate on Area, be sure to *enclose all parts* of the view within the selection box. When you move a view on a workspace, G2 must be running. For more information on Operate on Area, see the *G2 Reference Manual*.

Deleting Views

You can delete a view by deleting the workspace of the view.

Cloning Views

GXL does not support cloning of views. If you clone a view using Operate on Area or clone the workspace of the view, the cloned view will not function properly.

4

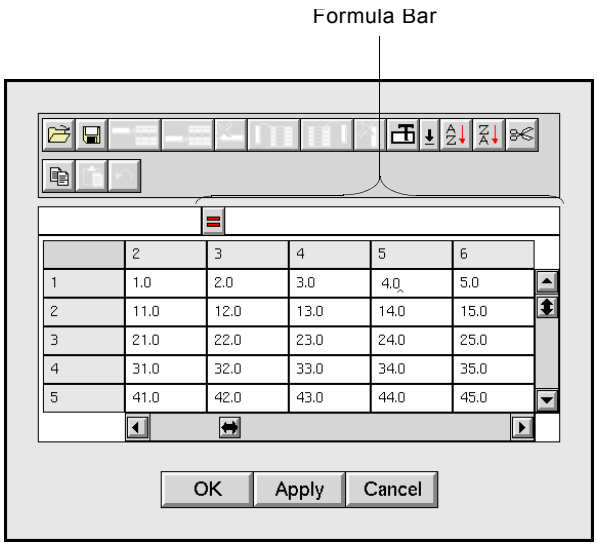
Using the Formula Bar

Introduces formulas and provides instructions for using formulas to calculate values in GXL spreadsheets.

Introduction

TheGXL **formula bar** allows you to calculate cell values, using mathematical operations, built-in G2 functions, or functions you define. You can use formulas to transform or rescale numerical data and to automatically calculate areas of the spreadsheet.

Here is an example of a spreadsheet view that contains a formula bar:



How Formulas Work in GXL

The role of formulas in GXL is more limited than in conventional spreadsheets. In conventional spreadsheets, formulas define the values of cells in terms of other cell values; dependencies are automatically updated when cells receive new values. In contrast, GXL formulas are not part of the definition of cells:

GXL cells contain values, not formulas.

GXL does not remember dependencies between cells in terms of formulas and cannot update values based on formulas that have been applied in the past.

Even though GXL does not provide true spreadsheet-like formula updating, formulas can be useful for many types of data manipulation. Additionally, because GXL provides optional callbacks to user-defined G2 procedures, you can provide event-based updating of spreadsheet values through callbacks, where necessary.

GXL Formula Syntax

The syntax of a formula is equivalent to that of a G2 expression. As you enter a formula, the formula is parsed. If GXL cannot parse the expression you enter, ellipses (...) appear in the formula. You can not enter the formula until you have corrected the syntax.

Note Entering Ctrl + A aborts a formula entry.

You can refer to single cells, rows, and columns, or ranges of cells, rows, and columns in formulas. The following table summarizes how you refer to these spreadsheet elements in formulas, where *w*, *x*, *y*, and *z* are represent zero-based row/column indices:

To refer to a...	Enter...	For example...
Single cell	R*w*C*x*	R2C3
Single row	R*w*	R2
Single column	C*x*	C3
Range of cells	R*w*C*x*::R*y*C*z*	R2C3::R6C4
Range of rows	R*w*::R*y*	R2::R6
Range of columns	C*x*::C*z*	C3::C4

For example, the syntax of the sum of cell (2, 3) and cell (6, 4) would be:

```
R2C3 + R6C4
```

When using a function that accepts a cell range, such as the GXL function gxl-sum, the syntax would be:

```
gxl-sum (R2C3::R6C4)
```

Note that the row/column numbers of formulas are *absolute* spreadsheet coordinates and are not necessarily the same as the numbers of the row/column selector cell. The latter are typically numbered from the beginning of a cell group.

Considerations

Make sure that you match dimensions correctly when defining formulas that mix reference to rows, columns, cells, and ranges. You cannot, for example, add a cell to a column, divide an area by a row, or multiply a range of columns by a constant.

However, if a function of a row, column, or range returns a scalar value (a single value), you can use this function value anywhere a reference to a single cell or a constant value would be acceptable. For example, you can use the function gxl-mean, which returns a scalar, in the expression R2C3/gxl-mean(C3).

Entering a Formula

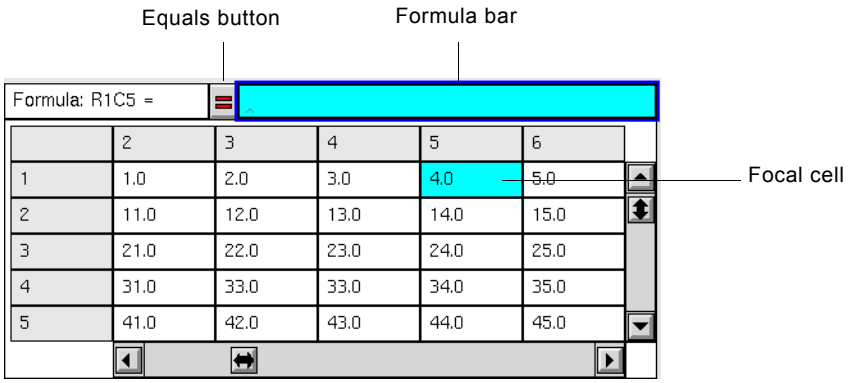
Before you can enter a formula, you must first select the target cell whose value is to be calculated. The cell you select is called the **focal cell**. You then activate the formula to enable formula entry. When you activate the formula bar:

- The color of the focal cell and the formula bar changes to cyan.
- The row-column indicator changes to display the following:

  Formula: [focal cell coordinates] =

- An editor opens on the formula bar.

For example:



You enter a formula through a combination of typing and selecting cells, rows, columns, and areas as follows:

- Clicking a cell adds the row/column reference of the cell to the formula.
- Clicking a row or column selector adds the appropriate row or column reference at the end of the formula.
- Clicking one cell and then shift-clicking another cell adds a range, indicated by the double colon (::), to the end of the formula.

  You cannot select ranges by dragging the mouse.

When selecting ranges, you can select a range of cells or rows or columns. However, GXL does not allow you to select ranges involving a mixture of cells, rows, and columns.

Note When you select a cell or a range, the reference to the cell or range is *always added at the end* of the formula, even if the text insertion cursor is elsewhere.

To enter a formula:

1. Select the focal cell, whose value you want to calculate.
2. Activate the formula bar by clicking either of the following:
   - Equals button
   - Formula bar
3. Enter the formula in the Formula bar.
4. Press Enter to evaluate the formula.

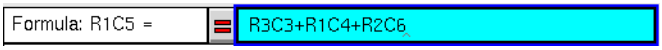
   The result appears in the focal cell.

For example, suppose you want to calculate the selected cell as the sum of several other cells.

To calculate the sum:

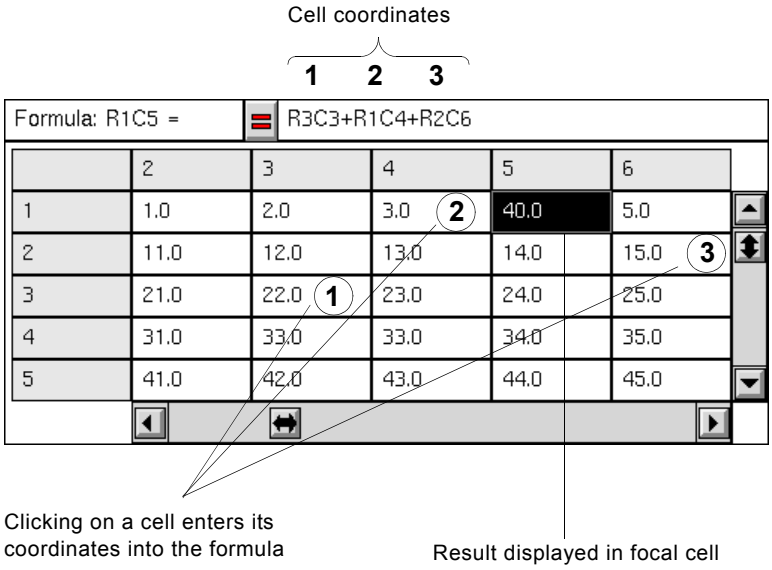
1. Select the focal cell.
2. Click the formula bar or the equals button to enter formula entry mode.
3. Click any cell and type a plus sign (+).
4. Click another cell and type a plus sign (+).
5. Click a final cell.

   The formula looks like this:



6. Press Enter to evaluate the formula. The result appears in the focal cell.

The following figure illustrates this example:



	2	3	4	5	6
1	1.0	2.0	3.0	40.0	5.0
2	11.0	12.0	13.0	14.0	15.0
3	21.0	22.0	23.0	24.0	25.0
4	31.0	33.0	33.0	34.0	35.0
5	41.0	42.0	43.0	44.0	45.0

Error Handling

If GXL cannot evaluate the formula you have entered, it clears the target cell, and signals an error with an audible beep. You can use the undo button to return the spreadsheet to its previous state and recover the lost value.

You might also receive a logbook message telling you that there are inconsistencies in a formula. You can safely ignore this logbook message.

Applying a Formula to Multiple Cells

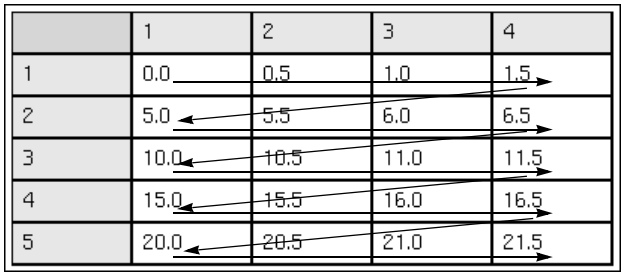
If you want to apply a formula to more than one cell, begin by selecting a range of cells. When you enter formula entry mode, the cell in the top left-hand corner of the selection is highlighted, indicating that it is the *focal cell*. You enter the formula as if you were calculating *only* the value of the focal cell. GXL applies the formula to all cells in the original selection by interpreting the formula as *relative references*.

Relative references work in the following way. Suppose the focal cell is R*w*C*x*. To apply this formula to another cell, R*y*C*z*, all the row references in the formula are incremented by (*y* - *w*), and all the column references by (*z* - *x*).

For example, if R2C4 is the focal cell and you enter the formula R2C2 + R2C3, and if you apply the same formula to R3C4, GXL will calculate it as R3C2 + R3C3.

Order of Calculation

When the target of a formula is a range of cells, the order of calculation used by GXL can affect the result. GXL always applies formulas cell-by-cell in a left-to-right, top-to-bottom manner, as this figure illustrates:



How GXL applies formulas over a range of cells.

Because of this calculation order, you should generally avoid formulas that place the result of a calculation into a cell of the spreadsheet that is involved in a range calculation.

For example, if you want to subtract the mean value of a column from each element of the column and if R1C1 is the focal cell, you might write R1C1 = R1C1 - gxl-mean(C1).

For the focal cell, this formula works correctly, but by the time the formula is applied to row 2 as R2C1 = R2C1 - gxl-mean(C1), the value in R1C1 has changed; hence the mean of the column will have changed, and the result will not be as intended. To perform this type of calculation correctly, pre-calculate the column mean and put the result into another cell.

Examples

Calculating Columns

The following example shows how GXL uses a formula to calculate a column of cells as the sum of two other columns. In this spreadsheet view of float data cells, GXL calculates column 4 as the sum of columns 2 and 3.

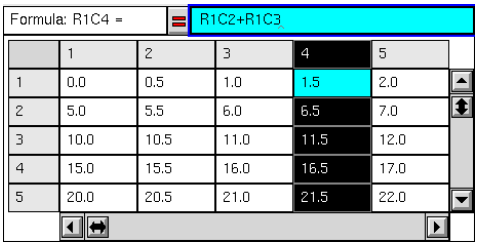
To calculate a column as the sum of other columns:

1. Select column 4.
2. Click the formula bar or the equals button to enter formula entry mode.

   GXL highlights the focal cell, which is R1C4 in the example.

3 Enter the formula R1C2 + R1C3.

For example:

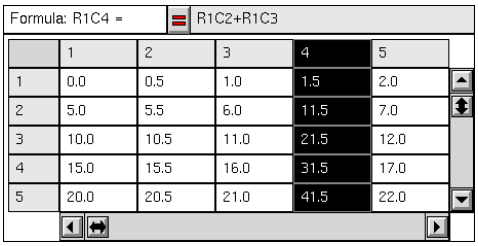


	1	2	3	4	5
1	0.0	0.5	1.0	1.5	2.0
2	5.0	5.5	6.0	6.5	7.0
3	10.0	10.5	11.0	11.5	12.0
4	15.0	15.5	16.0	16.5	17.0
5	20.0	20.5	21.0	21.5	22.0

4 Press Enter to apply the formula.

In the example GXL applies the formula, R1C2 + R1C3, to column 4.

Here are the results:



	1	2	3	4	5
1	0.0	0.5	1.0	1.5	2.0
2	5.0	5.5	6.0	11.5	7.0
3	10.0	10.5	11.0	21.5	12.0
4	15.0	15.5	16.0	31.5	17.0
5	20.0	20.5	21.0	41.5	22.0

Formulating Consecutive Integers

The following example shows how GXL uses a formula to fill a column with consecutive integers. In this spreadsheet view of integer data cells, GXL fills column 2 with consecutive integers:

To fill a column with consecutive integers:

1 Enter the integer 0 in R1C1.

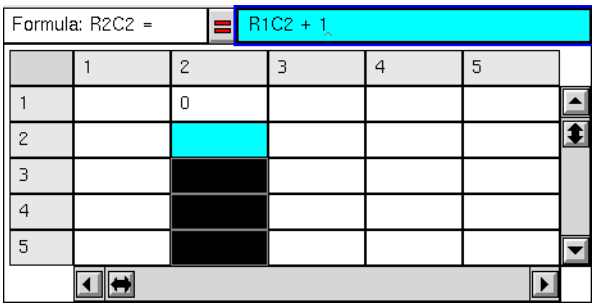
2 Select R2C2, and then scroll to expose the last cell in the column, and shift-select the last cell in the column, so that the remainder of the column is selected.

3 Click the formula bar or the equals button to enter formula entry mode.

GXL highlights the focal cell, which in the example is R2C2, the first selected cell.

4 Enter the formula R1C2 + 1.

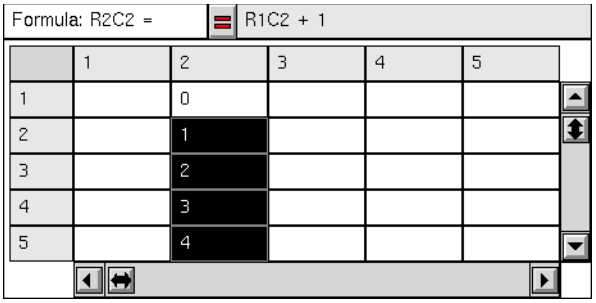
For example:



5 Press Enter to apply the formula.

GXL fills the cells in column 2 with consecutive integers.

Here are the results:



Using Built-in Functions

You can reference any G2 built-in function in formulas. For a complete list and description G2's built-in functions, see the *G2 Reference Manual*.

Additionally, GXL provides several basic functions that you can applied to ranges of cells, which include areas, rows, and columns. The following table summarizes the GXL functions:

Function Name	Calculation Performed
gxl-count	The number of values in a range.
gxl-maximum	The maximum quantity in a range.
gxl-mean	The average quantity in a range.
gxl-minimum	The minimum quantity in a range.

Function Name	Calculation Performed
gxl-quantity-count	The number of quantities in a range.
gxl-stdev	The standard deviation of the quantities in a range.
gxl-sum	The sum of quantities in a range.

For example, suppose you want to calculate a cell value as the square root of the mean sum of squares of the cells in column 3. The formula you enter looks like this:

```
sqrt(sum-of-squares(C3)/gxl-quantity-count(C3))
```

Note The sum-of-squares function, a user-defined function, is defined in Creating Your Own Functions.

When you select a range of cells, only data cells (text cells, integer cells, float cells, quantity cells, symbol cells, truth-value cells and value cells) within the range are included in function calculations. Selector cells, column headers, and disabled cells are ignored.

For example, if a column has 10 total rows and the first row is a column header and the second row contains column selector cells, then the gxl-count of the column is 8, not 10.

Creating Your Own Functions

You can add your own functions by creating function definitions in the standard G2 manner. See the *G2 Reference Manual* for details on creating function definitions.

Functions that apply to values in individual cells are straightforward to create. For example, if you want to define a text-maximum function, you can create the following function definition:

```
text-maximum(T1, T2) = (if T1 > T2 then T1 else T2)
```

You can reference this function in a formula by substituting cell references for T1 and T2, for example:

```
text-maximum(R2C3, R4C2)
```

The function is called with the values from the referenced cells.

To create functions that apply to ranges of cells, your function must accept a list of cell values as its only argument. For example, if you want to create a function that

finds the sum of squares over a range of cells, you create the following function definition:

```
sum-of-squares(List) = (the sum over each quantity Q in List of (Q*Q))
```

You can reference this function in a formula with any valid range arguments, for example:

```
sum-of-squares(R1C1::R8C5)
  or
sum-of-squares(C1::C5)
```

Here, GXL converts the range reference into a list of values from the data cells in the specified range and passes this list to the function for evaluation.

Note Because the order of values in the list is arbitrary, do not create formulas that depend on the ordering of the list.

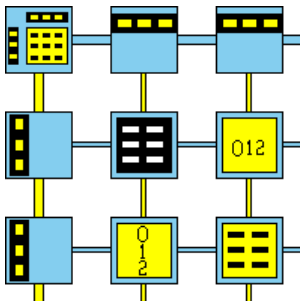
5

Working with Specifications

Describes how to build and use GXL specifications to create spreadsheets and views.

Introduction

A spreadsheet specification is a graphical layout of objects that serves as a template for the creation of spreadsheets and views. You build a specification on a workspace by cloning specification objects from the gxl-top-level workspace. The following figure illustrates a typical specification:



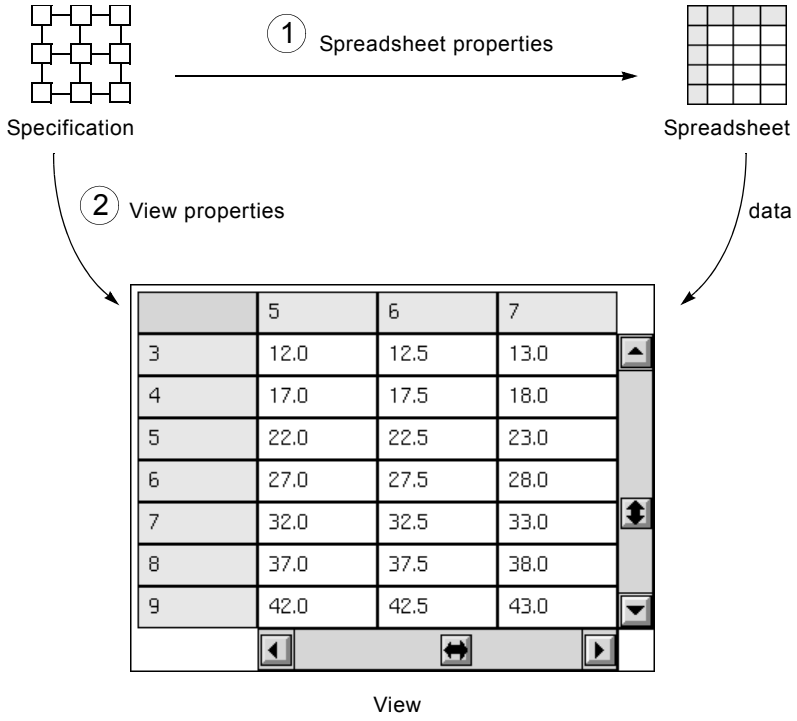
A specification layout serves the dual purpose of defining a spreadsheet and a view of the spreadsheet based on the specification objects, their class and their attribute values.

The attributes of the specification objects split into two non-overlapping categories:

- Attributes related to spreadsheet properties, such the type of data that a cell can contain, and whether data is stored in or external to the spreadsheet.
- Attributes related to view properties, such as the number of rows and columns of visible data, the size and color of the cells and their content, and whether a cell is selectable and editable.

For example, the gxl-total-rows attribute of a row controller specifies a spreadsheet property, while the gxl-visible-rows attribute specifies the appearance of a view.

The dual nature of a specification is reflected in the order of events when you create a spreadsheet and then a view of the spreadsheet. As the following figure illustrates, you first use the specification to create a spreadsheet. Then you use the spreadsheet and the specification to create a view.



View

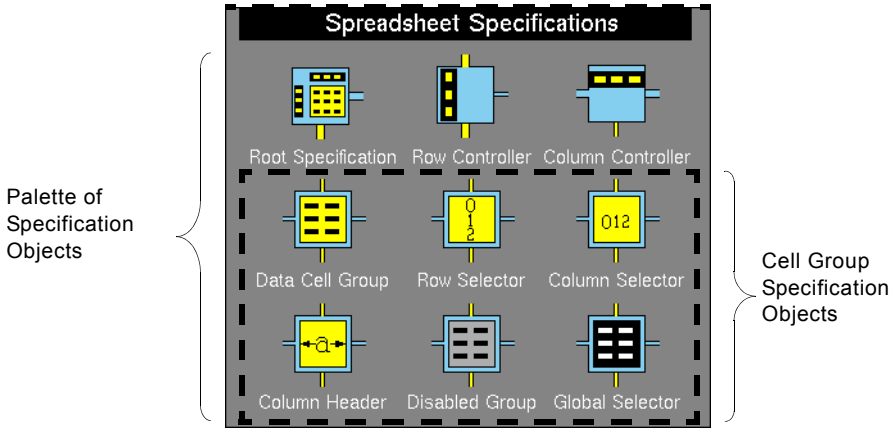
During the first phase, GXL references the spreadsheet properties of the specification. During the second phase, GXL references the view properties. It is quite possible to defer setting the view properties of the specification until after the creation of the spreadsheet.

GXL uses the information contained in a specification *only* at the time of spreadsheet or view creation. If you want to create multiple views of a single spreadsheet, each having a different layout, you can change the view-related attributes of the specification after creating one view and before creating another. Alternatively, you may use two different specifications to create alternative views, as long as each specification has the correct layout, which must be the same as the specification used to create the spreadsheet.

You may alter a specification manually or programmatically to create different spreadsheets and views from the same specification. Changing the attributes or structure of a specification *after* a spreadsheet or view has been created has no effect on existing spreadsheets or views. To change the properties of an existing spreadsheet or view, you must use GXL's API procedures.

The GLX Specification Layout

You specify the structure of your spreadsheet by constructing a specification layout. The layout consists of the following GXL specification objects cloned from the GXL top-level workspace:



A specification layout consists of:

- One root specification. which defines certain properties of the spreadsheet as a whole. The root specification is connected to the first row and column controllers.
- A column controller for each column of cell groups. The **column controller** specifies the common properties of the cell groups connected to it vertically, such as the total number of columns in the spreadsheet and the number of columns visible in the view.
- A row controller for each row of cell groups. The **row controller** specifies the common properties of the cell groups connected to it horizontally, such as the total number of rows in the spreadsheet and the number of rows visible in the view.
- A cell group specification object for each cell group in the spreadsheet. The cell group specifications are laid out in a rectangular grid that corresponds to the arrangement of cell groups within the spreadsheet. All cells within a cell group have the same value type and respond in a particular way to user input.

Cell Group Specifications

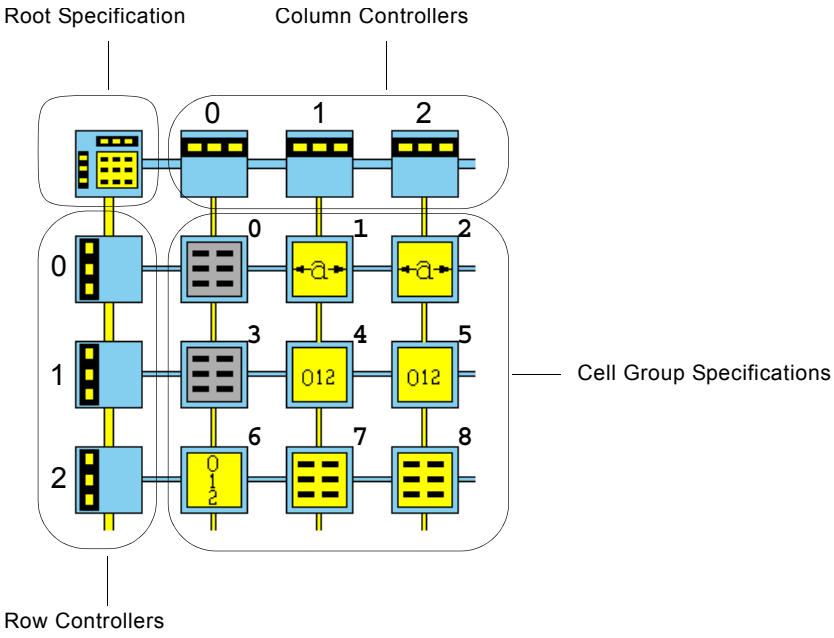
Cell group specifications define different rectangular areas of the spreadsheet. Within the specification layout:

- Cell groups are always numbered left to right and top to bottom, starting from 0.
- Cell groups that are side-by-side have the same number of rows.
- All cell groups that are aligned top-to-bottom have the same number of columns.

When working with cell groups programmatically, you always refer to a cell group by its number.

The cell group specifications are laid out in a rectangular grid that corresponds to the arrangement of cell groups within the spreadsheet. A specification layout must always have a full rectangular grid of cell group specification objects.

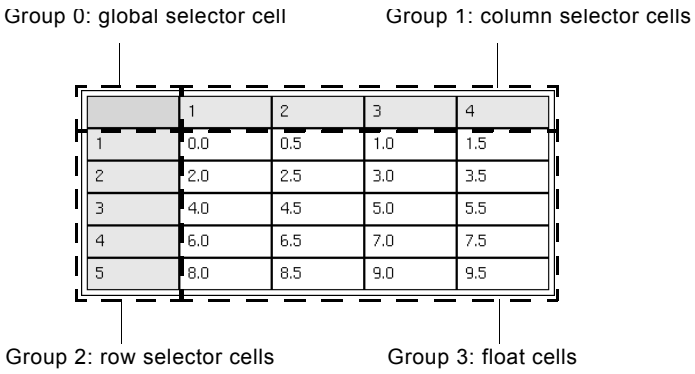
The following figure illustrates a typical specification layout.



Although this specification has an equal number of row and column controllers, leading to a square grid of cell group specifications, this is not a requirement. As long as there is at least one row controller and one column controller, there is no restriction on the number of row and column controllers.

Cell Groups and Spreadsheet Views

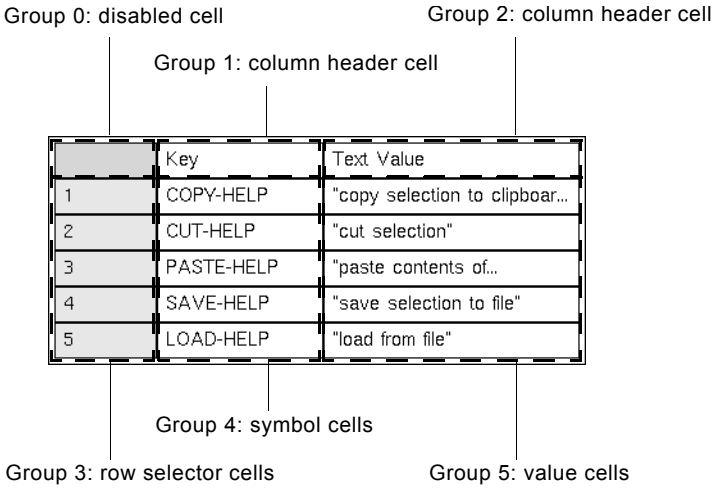
To understand how cell groups relate to a spreadsheet view, consider the following spreadsheet view:



	1	2	3	4
1	0.0	0.5	1.0	1.5
2	2.0	2.5	3.0	3.5
3	4.0	4.5	5.0	5.5
4	6.0	6.5	7.0	7.5
5	8.0	8.5	9.0	9.5

Displaying all the data in the spreadsheet, this spreadsheet is composed of four cell groups. Group 0 is composed of only one cell, a global selector cell. Group 1 is composed of four column selector cells. Group 2 is a set of row selector cells. Group 3 is a 5x4 grid of float cells.

The following figure illustrates a slightly more complicated example of cell groups. In this spreadsheet view, again displaying all the data in the spreadsheet, you see six cell groups:

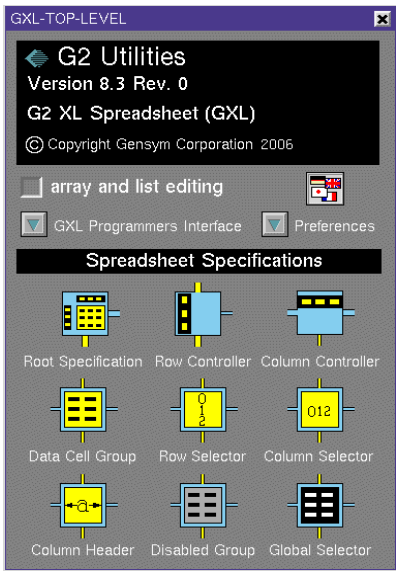


	Key	Text Value
1	COPY-HELP	"copy selection to clipboar...
2	CUT-HELP	"cut selection"
3	PASTE-HELP	"paste contents of...
4	SAVE-HELP	"save selection to file"
5	LOAD-HELP	"load from file"

In this case, the first three cell groups each contain only one cell. The remaining three cell groups each have five rows and one column, containing row selector cells, symbol cells, and value cells, respectively.

Building a Specification

You build a spreadsheet specification by cloning specification objects and arranging them in a layout on your workspace. The GXL top-level workspace contains a palette of the specification objects:



G2 must be running to clone specification objects from the palette.

Each specification object has a number of attributes, of which some define characteristics of the spreadsheet and others define characteristics of the view. You define the properties of the spreadsheet and its view by specifying values for the attributes.

See Specification Objects for descriptions of the attributes for each specification object. To check which attributes of an object define spreadsheet properties and which define view properties, see Defining Spreadsheet and View Properties.

To review an instructional example of building a specification of a GXL spreadsheet, see Creating a Custom Spreadsheet and View. The gxldemo KB also contains a number of examples that include different specifications for your review.

To clone objects from the palette:

1. Click the object you would like to clone.
2. Move the mouse to the desired location on the destination workspace and click again to deposit the object.

To help you make the correct connections, the connection stubs associated with row and column controllers and cell group specifications have different cross-

sections. G2 only lets you join compatible connection types, preventing you from making an erroneous connection.

Note When connecting specification objects, you must use existing stubs.

To connect the specification objects:

1 Click one of the connection stubs.

When you move the mouse, the connection stub follows the cursor.

2 Position the stub directly over the adjacent stub.

3 Click once more to connect the two stubs together.

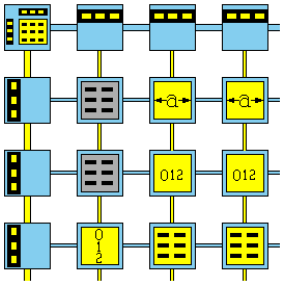
Caution Do not drag the connector directly into another object.

After making all the necessary connections, you can neatly arrange the specification.

To neatly arrange the specification:

➔ Choose arrange from the menu of the root specification.

The following figure is an example of a completed specification layout:



Specification Objects

This section discusses the types of objects that make up a specification and their attributes. In total, there are nine different specification objects. Six of these specify different types of cell groups, specifically, groups consisting of data cells, row selectors, column selectors, column headers, disabled cells, and global selectors. The other three specification objects - root specification, row controller, and column controller - specify collective properties of cell groups and the spreadsheet as a whole.

The Root Specification Object

The root specification object defines certain overall properties of the spreadsheet and spreadsheet view. A root specification is an instance of the class gxl-root-specification. The following table describes the attributes of the root specification:

Attribute	Description
gxl-spreadsheet-class	A symbol naming of the class of spreadsheet to be created.
Allowable values:	The symbol gxl-spreadsheet or a symbol naming a subclass of this class
Default value:	The symbol gxl-spreadsheet.
gxl-spreadsheet-view-class	A symbol naming the class of spreadsheet view to be created.
Allowable values:	The symbol gxl-spreadsheet-view or a symbol naming a subclass of this class
Default value:	The symbol gxl-spreadsheet-view

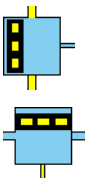
Attribute	Description
gxl-spreadsheet-tools	A symbol array containing symbols representing the tools to be included in the view.
Allowable values:	Any of the following symbols: rc-indicator, formula-tool, file-save, file-load, insert-row-before, insert-row-after, delete-row, insert-column-before, insert-column-after, delete-column, select-color, cut, copy, paste, undo, sort-ascending, sort-descending, ok, apply, cancel; and/or any symbol naming a subclass of gxl-toolbar-button
Default value:	A symbol array containing all the allowable values
Notes:	Choosing edit spreadsheet tools on the menu of the root specification object displays the gxl-spreadsheet-tools array in a spreadsheet view. For details, see Customizing the Toolbar.
gxl-toolbar-width	An integer indicating the toolbar width, in pixels. When set to the default value (0), the toolbar width equals the width of the view.
Allowable values:	Any positive integer
Default value:	0
gxl-selection-callback	A symbol optionally naming a procedure to be called whenever cells are selected on a view.
Allowable values:	Any symbol naming a procedure, or the symbol unspecified.
Default value:	The symbol unspecified
Notes:	Use this attribute to define a callback for every selection event, rather than using different callbacks with each cell group.

Attribute	Description
gxl-matrix-extension-procedure	A symbol optionally naming a procedure to be called when GXL needs to extend the size of a matrix. When set to the default, unspecified, GXL uses the built-in default method.
Allowable values:	Any symbol naming a procedure, or the symbol unspecified
Default value:	The symbol unspecified
Notes:	For a discussion on using matrix extension procedures and an example, see Matrix Extension Procedures.
gxl-cleanup-when-resizing-matrices	A truth-value that determines whether GXL deletes orphan vectors when reducing the length of the spine (item-array, -list) of a matrix.
Allowable values:	true or false
Default value:	true
gxl-make-edits-permanent	A symbol that determines if edits to data in GXL spreadsheets are permanent. When set to prompt, a dialog appears, allowing user to determine if edits are permanent.
Allowable values:	The symbol yes, no, or prompt
Default value:	The symbol prompt
gxl-store-data-internally	A truth-value that determines whether to store data associated with a GXL spreadsheet in the spreadsheet itself, or to make calls to data server procedures when data is needed for display or computation.
Allowable values:	true or false
Default value:	true

Attribute	Description
gxl-external-data-server	A symbol naming a procedure that GXL uses to determine the value stored in a spreadsheet cell.
Allowable values:	Any symbol naming a procedure that returns the data value at a specific row, column position of the spreadsheet, or the symbol unspecified
Default value:	The symbol unspecified
Notes:	This attribute is required when gxl-store-data-internally is set to false; otherwise it is not used. For a discussion on using an external data server procedure and an example, see External Data and Color Server Procedures.
gxl-external-color-server	A symbol naming a procedure that GXL uses to determine the colors of a cell of a spreadsheet.
Allowable values:	Any symbol naming a procedure that returns color values for a specific row, column position of the spreadsheet, or the symbol unspecified
Default value:	The symbol unspecified
Notes:	This attribute can be used when gxl-store-data-internally is set to false; otherwise it is not used For a discussion on using an external color server procedure and an example, see External Data and Color Server Procedures.
gxl-scroll-callback	A symbol naming a procedure to be called when a view of the spreadsheet is scrolled.
Allowable values:	Any symbol naming a procedure, or the symbol unspecified
Default value:	The symbol unspecified

Attribute	Description
Notes:	Using an optional scrolling callback is useful when you are storing the spreadsheet data externally. For a discussion on using scroll callbacks and an example, see Scrolling Callback Procedures.
gxl-allow-non-standard-row-heights	A truth-value that determines whether the minimum row heights of 28, 34, or 46 pixels for small, large and extra-large fonts, respectively, should be enforced.
Allowable values:	true or false
Default value:	false
Notes:	For a discussion on using nonstandard cell heights, see Customizing the Data Display in Cells.
gxl-return-and-tab-key-handler	A symbol naming a procedure to be called when the user finishes editing a cell by pressing either the tab or return key, which returns to GXL the row and column of the next cell to be edited, if any.
Allowable values:	A symbol naming a return and tab key handling procedure, or the symbol unspecified
Default value:	The symbol unspecified
Notes:	For a discussion return and tab key handling procedures and an example, see Return and Tab Key Handler Procedures.

Row and Column Controllers



The column controller and row controller objects specify attributes that are common to an entire column or row of the spreadsheet, even if the column or row might include more than one cell group. These controllers are instances of the classes gxl-column-controller and gxl-row-controller, respectively.

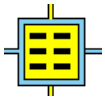
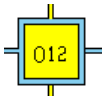
The attributes of column and row controllers are:

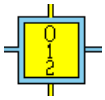
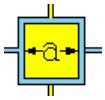
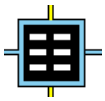
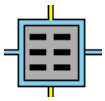
Attribute	Description
gxl-visible-rows **gxl-visible-columns**	An integer specifying the number of rows/columns in the cell group that are visible on the view.
Allowable values:	Any positive integer
Default value:	1
gxl-total-rows **gxl-total-columns**	An integer specifying the total number of rows/columns in the cell group.
Allowable values:	Any positive integer
Default value:	1
gxl-cell-height	An integer giving the cell height in pixels for the rows in the cell group.
Allowable values:	Any integer greater than or equal to 14
Default value:	28

Attribute	Description
Notes:	You can set cell height larger or smaller than the standard height for font size. See Customizing the Data Display in Cells for a discussion on using nonstandard cell heights.
gxl-cell-width	An integer giving the cell width in pixels for the columns in the cell group.
Allowable values:	Any integer greater than or equal to 32
Default value:	80.

Cell Group Specification Objects

There are six types of cell group specifications:

Specification Object	Description
 Data Cell Group	The Data Cell Group is an instance of the class gxl-cell-group. This specification defines data cell properties, including the data type, which can be one of the following: integer float quantity symbol text truth-value value
 Column Selector	The Column Selector is an instance of the class gxl-column-counter-cell-group. This specification defines special cells that have the property of selecting the entire column containing the cell when the cell is selected. These cells are often used to consecutively number the columns of a spreadsheet.

Specification Object	Description
 Row Selector	The Row Selector is an instance of the class gxl-row-counter-cell-group. This specification defines special cells that have the property of selecting the entire row containing the cell when the cell is selected. Cells of this type are often used to consecutively number the rows of a spreadsheet.
 Column Header	The Column Header is an instance of the class gxl-column-header-for-cell-group. This specification defines a special cell that spans the entire width of the cell group. Use this specification to create a cell that labels columns on a view.
 Global Selector	The Global Selector is an instance of the class gxl-global-selector-cell-group. This specification defines a special cell that has the property of selecting the entire spreadsheet when the cell is selected.
 Disabled Group	The Disabled Group is an instance of the class gxl-disabled-cell-group. This specification defines special cells that cannot contain data and are neither selectable nor editable. They are used to fill areas of the spreadsheet that do not contain data.

The six cell group specification objects share similar attributes, although not all cell group types have the complete set of attributes. For example, while all attributes are available for the data cell group, only the text, background, border, and font scale attributes are available for the disabled cell group. For a list of attributes for a particular specification object, see Defining Spreadsheet and View Properties.

The following table describes the attributes of cell groups:

Attribute	Description
gxl-cell-type	A symbol naming the type of cell to be used in the cell group.
Allowable values:	One of the following symbols: float-cell integer-cell quantity-cell symbol-cell text-cell truth-value-cell value-cell
Default value:	The symbol value-cell
gxl-font-size	A symbol indicating the font size to be used in the cells of the cell group.
Allowable values:	The symbol small, large, or extra-large
Default value:	The symbol small
gxl-default-background-color	The default background color of the cells within a cell group in the view.
Allowable values:	A symbol representing any G2 standard color
Default value:	Symbolic color depends on cell group: White for data cell and column header groups; Light-gray for a disabled cell group; Extra-light-gray for global, row, and column selector groups
Notes:	Use gxl-set-color-pattern-of-cell to override colors in a spreadsheet view.

Attribute	Description
gxl-default-text-color	The default text color of the cells within a cell group in the view.
Allowable values:	A symbol representing any G2 standard color
Default value:	Symbolic color depends on cell group: light-gray for disabled cell group; black for all other cell groups
Notes:	Use gxl-set-color-pattern-of-cell to override this attribute setting.
gxl-default-border-color	The default border color of the cells within a cell group in the view.
Allowable values:	A symbol representing any G2 standard color
Default value:	The symbol black
Notes:	Use gxl-set-color-pattern-of-cell to override this attribute setting.
gxl-cell-group-initialization-data	A value passed to the initialization procedure.
Allowable values:	See Notes
Default value:	The symbol unspecified
Notes:	For a discussion of this attribute and its values, see Initializing Cell Group Data.
gxl-cells-are-selectable	A truth-value indicating the whether the cells of the cell group will be selectable via the mouse.
Allowable values:	true or false
Default value:	true

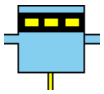
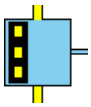
Attribute	Description
gxl-cells-are-editable	A truth-value indicating the whether the cells of the cell group will allow manually editing.
Allowable values:	true or false
Default value:	true
gxl-additional-validation-procedure	A symbol optionally naming a procedure used to validate manual entries.
Allowable values:	Any symbol naming a procedure, or the symbol unspecified
Default value:	The symbol unspecified
Notes:	For a discussion on validation procedures and an example, see Validation Procedures.
gxl-initialization-procedure	A symbol optionally naming a procedure used to initialize cell values, called when the spreadsheet is first created.
Allowable values:	Any symbol naming a procedure, or the symbol unspecified
Default value:	The symbol unspecified
Notes:	For a discussion initialization procedures and an example, see Initialization Procedures.
gxl-reinitialization-procedure	A symbol optionally naming a procedure used to be called when new rows or columns are added to the cell group, and when rows and columns are deleted.
Allowable values:	Any symbol naming a procedure, or the symbol unspecified
Default value:	The symbol unspecified

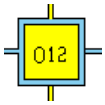
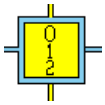
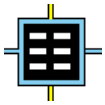
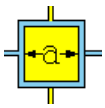
Attribute	Description
Notes:	For a discussion on reinstallation procedures, see Initialization Procedures.
gxl-callback-procedure	A symbol optionally naming a procedure to be called when a cell value is set using gxl-set-cell-contents or through manual entries.
Allowable values:	Any symbol naming a procedure, or the symbol unspecified
Default value:	The symbol unspecified
Notes:	For a discussion on using callbacks and an example, see Cell Callback Procedures.
gxl-selection-callback-procedure	A symbol optionally naming a procedure to be called when a cell is selected through manual or programmatic action.
Allowable values:	Any symbol naming a procedure, or the symbol unspecified
Default value:	The symbol unspecified
Notes:	For a discussion on selection callbacks and an example, see Selection Callback Procedures.
gxl-float-format	An object describing the desired formatting for floating point numbers in this cell group.
Allowable values:	An object of the class gxl-float-formatter

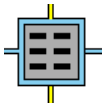
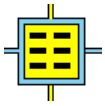
Attribute	Description
Default value:	An object of the class gxl-float-formatter, with these attributes: gxl-use-default = true gxl-minimum-width = 1 gxl-precision = 4 gxl-output-format = the symbol best gxl-remove-trailing-zeros = true
Notes:	For details, see Customizing the Appearance of Floating Point Numbers.
gxl-font-scale	Specifies the magnification of the font as a float value, where a value less than 1.0 shrinks the size of the font.
Allowable values:	Any positive or negative float number
Default value:	1.0
Notes:	For information on using this attribute, see Customizing the Data Display in Cells.

Defining Spreadsheet and View Properties

The attributes of specification objects define two separate categories of properties: those related to the spreadsheet and those related to its view. The following table summarizes which of the specification objects attributes define spreadsheet properties and which define view properties.

Specification Object	Spreadsheet Attributes	View Attributes
 Root Specification	gxl-spreadsheet-class gxl-selection-callback gxl-matrix-extension-procedure gxl-cleanup-when-resizing-matricies gxl-make-edits-permanent gxl-store-data-internally gsl-external-data-server gxl-external-color-server gxl-scroll-callback	gxl-spreadsheet-view-class gxl-spreadsheet-tools gxl-toolbar-width gxl-allow-nonstandard-row-heights gxl-return-and-tab-key-handler
 Column Controller	gxl-total-columns	gxl-visible-columns gxl-cell-width
 Row Controller	gxl-total-rows	gxl-visible-rows gxl-cell-height

Specification Object	Spreadsheet Attributes	View Attributes
 Column Selector  Row Selector  Global Selector	gxl-cell-group-initialization-data gxl-initialization-procedure gxl-reinitialization-procedure gxl-callback-procedure gxl-selection-callback-procedure	gxl-cells-are-selectable gxl-default-background-color gxl-default-border-color gxl-default-text-color gxl-float-format gxl-font-scale gxl-font-size
 Column Header	gxl-cell-group-initialization-data gxl-callback-procedure gxl-selection-callback-procedure	gxl-cells-are-editable gxl-cells-are-selectable gxl-default-background-color gxl-default-border-color gxl-default-text-color gxl-float-format gxl-font-scale gxl-font-size

Specification Object	Spreadsheet Attributes	View Attributes
 Disabled Group	None	gxl-default background-color gxl-default-border-color gxl-default-text-color
 Data Cell Group	gxl-cell-type gxl-cell-group-initialization-data gxl-additional-validation-procedure gxl-initialization-procedure gxl-reinitialization-procedure gxl-callback-procedure gxl-selection-callback-procedure	gxl-cells-are-editable gxl-cells-are-selectable gxl-default-background-color gxl-default-border-color gxl-default-text-color gxl-float-format gxl-font-scale gxl-font-size

To define the spreadsheet and view properties:

1. Click the appropriate specification object and choose table from the menu to display its attribute table.
2. Edit the table to specify values for those attributes you want to define.

For descriptions of specific attributes, see Specification Objects.

Initializing Cell Group Data

If you provide your own initialization procedure, the attribute gxl-cell-group-initialization-data can be any value, which is simply passed to your initializer.

Numbering Row and Column Selector Cells

For row and column selectors, the default initializer is gxl-serial-integer-initialization. This procedure expects gxl-cell-group-initialization-data to be an integer, which is 1 by default. The value of the gxl-cell-group-initialization-data becomes the value in the first row or column selector cell in the view. For example, if you want the row and column selectors to be numbered serially starting at 0, then set gxl-cell-group-initialization-data to 0.

Labeling Column Headers

For column headers, the gxl-cell-group-initialization-data contains the text you want to put in the column header cell. The value of this attribute is a text string. When specifying a column heading, enclose the text with double quotes (“).

For details on the use of initialization procedures, see Programming GXL.

Localizing Column Header Text

You can localize column header text before or after creating the spreadsheet. To do so before creating the spreadsheet, include the localized text in the Column Header specification object. To do so after creating the spreadsheet, use the gxl-set-group-column-header procedure.

For details on using the gxl-set-group-column-header procedure, see gxl-set-group-column-header.

Customizing the Appearance of Floating Point Numbers

You can specify how floating point numbers appear in views of the spreadsheet by using the gxl-float-format attribute of cell group specifications. This attribute contains a subobject with the following attributes:

Attribute	Description
gxl-use-default	A truth-value indicating if default formatting is used. If true, the default format described in The Spreadsheet Toolbar is used, and the remaining attributes of the float format are ignored.
Allowable values:	true or false
Default value:	true
gxl-minimum-width	Specifies the minimum number of characters in the formatted version of float. If the formatted float value has fewer characters than the number you specify, the text in the cell is padded on the left side with space characters.

Attribute	Description
Allowable values:	Any positive integer
Default value:	1
gxl-precision	Specifies either the number of digits to the right of the decimal point or the significant digits, depending on the gxl-output-format. If the output format is float or exponent, gxl-precision indicates the digits to the right of the decimal. Otherwise, gxl-precision is the number of significant digits.
Allowable values:	Any positive integer
Default value:	4
gxl-output-format	Determines the representation of the value. If this attribute is float, the value is displayed as a float; if exponent, exponential notation is used; if best, the value is displayed as an exponent if the result is too small or large for the specified precision.
Allowable values:	One of the following symbols: float, exponent, or best
Default value:	The symbol best
gxl-remove-trailing-zeros	A truth-value indicating whether zeros to the right of the last non-zero digit are to be stripped.
Allowable values:	true or false
Default value:	true

GXL determines the format when creating the view. Once created, existing views remain unaffected by any changes to its associated specification.

To change the formatting of floats on an existing view:

➔ Use the API procedure gxl-set-float-format-of-group-on-view.

You can also programmatically change the float format for an existing view, using the API procedure gxl-set-float-format-of-group-on-view. For a description of this procedure, see .

Customizing the Data Display in Cells

When GXL creates a spreadsheet view, it sets a minimum cell height to accommodate the font size as follows:

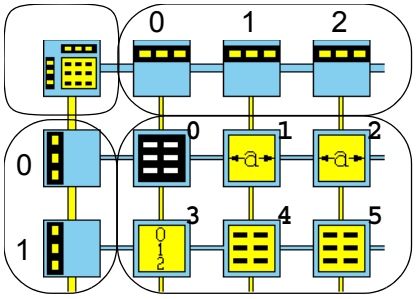
If the font size is...	The cell height is...
small	28 pixels
large	34 pixels
extra-large	46 pixels

These specification attributes control the font size and cell height:

- The gxl-cell-height attribute defines the height of the cells displayed for the cell groups associated with a particular row controller.
- The gxl-font-size attribute determines the size of the font displayed in cells of a particular cell group.

Even if you define a smaller cell height for a particular font size, the cell height defaults to the appropriate size. Additionally, if there are different font sizes within a row, the largest font size determines the height of the cells.

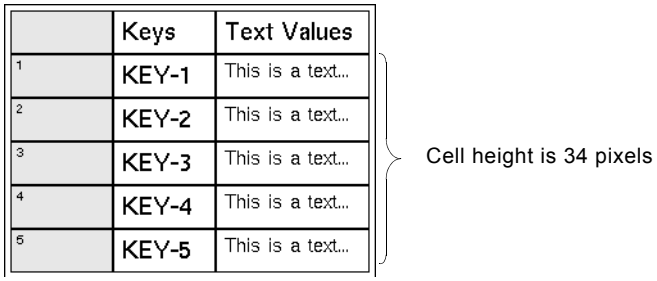
Consider the following specification:



Given these specification properties:

Specification Object	Attribute	Value
row controller (1)	gxl-cell-height	28
data cell group (4)	gxl-font-size	large
data cell group (5)	gxl-font-size	small

GXL creates the following spreadsheet view:



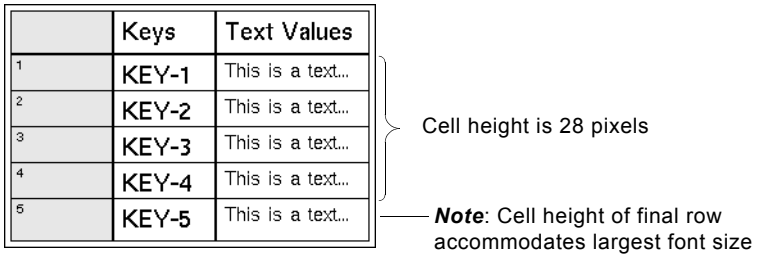
You can override this default action by enabling the display of non-standard cell heights.

To enable the display of non-standard cell heights:

➔ Change the gxl-allow-non-standard-row-heights attribute of the root specification to true.

Note If you use a nonstandard height, the last row always defaults to the standard cell height of the largest font size.

Using the same specification properties of the previous example, GXL creates the following spreadsheet view when gxl-allow-non-standard-row-heights is true.



Displaying More Information in Views

Using non-standard cell heights and a scaled font, you can display more rows of information on a spreadsheet view. The default font magnification is 1.0. You can make a specific font size appear larger or smaller by adjusting its magnification.

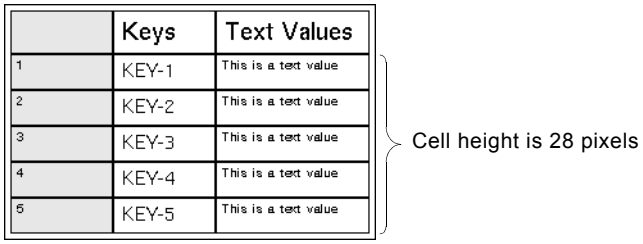
To change the magnification of a font size:

➔ Set the gxl-font-scale attribute of the cell group to a value that is greater than or less than 1.0.

For example, given these specification properties:

Specification Object	Attribute	Value
root-specification	gxl-allow-non-standard-heights	false
data-cell-group (4)	gxl-font-size gxl-font-scale	small 1.0
data-cell-group (5)	gxl-font-size gxl-font-scale	small 0.7

GXL creates the following spreadsheet view:



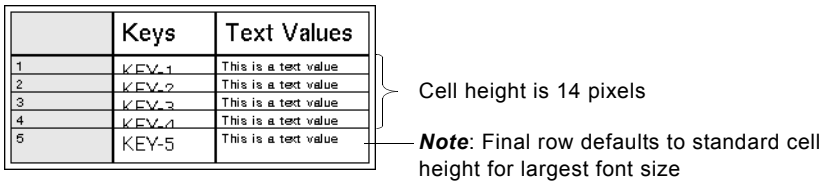
As the figure illustrates, a smaller font display results in more white space in the cell.

By adjusting the gxl-cell-height attribute of the row controller to a smaller pixel height you can display more rows of information. For example, given these specification properties:

Specification Object	Attribute	Value
root-specification	gxl-allow-non-standard-heights	true
row-controller (1)	gxl-cell-height	14

Specification Object	Attribute	Value
data-cell-group (4)	gxl-font-size gxl-font-scale	small 1.0
data-cell-group (5)	gxl-font-size gxl-font-scale	small 0.7

GXL creates the following spreadsheet view:



As you can see, reducing the cell height could result in clipping the bottom of the text display in a cell. Therefore, use care when setting the cell height and font magnification.

If you want to pack more information into a small space, try the following settings:

gxl-font-size	gxl-font-scale	gxl-cell-height
small	1.0	22
small	0.9	20
small	0.8	18
small	0.7	14

When non-standard heights are used, there is a small but noticeable cost in speed when scrolling or refreshing a view.

Assigning Color Patterns to Cells

For each cell in the spreadsheet, you can also specify the color of the data value, the background, and the border of the cell.

Three attributes of cell groups control the color display. The attributes are:

- gxl-default-text-color, which specifies the color of the data displayed in the cell.
- gxl-default-background-color, which specifies the background color of the cell.
- gxl-default-border-color, which specifies the border color of the cell.

The colors specified by these attributes apply to all of the cells in the corresponding cell group. The specified cell colors are displayed in all views of the cell.

Caution Never use white text on black background as a color combination in your spreadsheet. This color pattern is reserved for selected cells.

Controlling Cell Selection Behavior

The selection behavior of a cell depends on two properties of the cell group to which the cell belongs: its *selectability* and its *editability*. The cell group attributes that specify these properties are:

- gxl-cells-are-selectable, which controls selectability.
- gxl-cells-are-editable, which controls editability.

All cells within a cell group have the same selectability and editability properties. To be editable, a cell must be selectable.

You can programmatically control the selectability and editability of cells with the following API procedures:

- gxl-set-protection-of-group-on-view sets the editability and selectability of cell groups on a view.
- gxl-set-protection-on-entire-view sets these properties for all of the cells on a view.

For descriptions of these procedures, see Setting Spreadsheet and View Properties.

Customizing Selection Behavior

You can customize the selection behavior of cells within a cell group by creating a selection callback procedure and specifying the procedure name in the gxl-selection-callback-procedure attribute of the appropriate cell group.

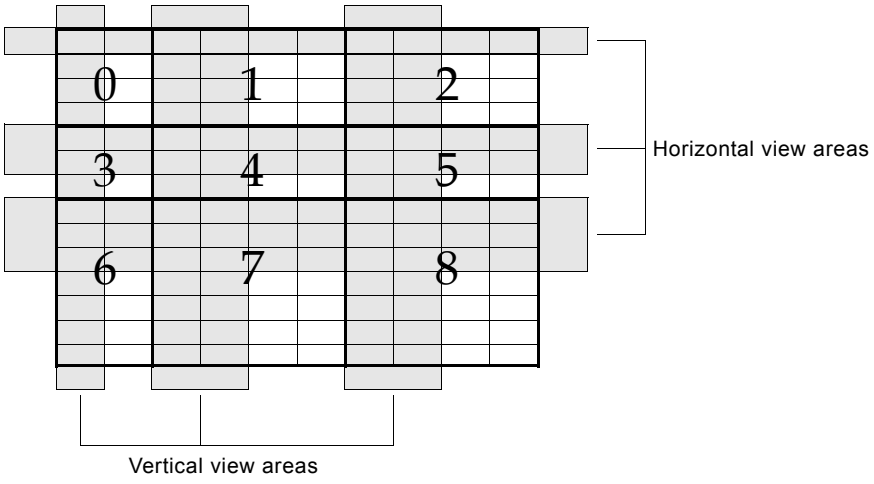
For information on creating and using selection callbacks, see Selection Callback Procedures.

Displaying View Areas

Row and column controller attributes determine the total number of rows and columns of a spreadsheet and the number of visible rows and columns of the view.

The gxl-total-rows and gxl-visible-rows attributes of a row controller object determine the total number of rows and the number of visible row for all cell groups connected to it horizontally. The gxl-total-columns and gxl-visible-columns attributes of a column controller object determine the total number of columns and the number of visible columns for all cell groups connected to it vertically.

Conceptually, this creates a number of horizontal and vertical **view areas** superimposed on the spreadsheet's data. If a cell is inside both a horizontal and a vertical view area, it is visible in the view. In the following figure, the shaded cells are exposed on the view:



The number of rows or columns in the view areas determines how many cells are shown in a view. It is feasible to have a view area with zero rows and columns, entirely hiding the contents of a cell group. Or, you may have a view area that exposes all rows or columns in a cell group, in which case scrolling is unnecessary. It is even possible to have a view area that is larger than the number

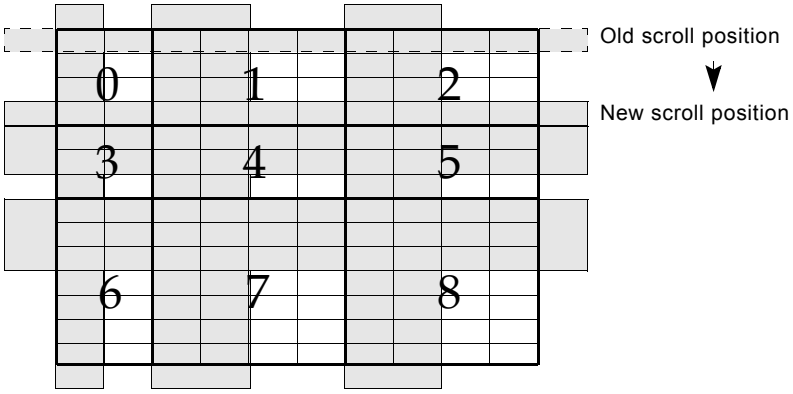
of rows and columns in the cell group. In this case, disabled cells (colored gray) are displayed in the view.

Scrolling in View Areas

If the size of a view area is such that it leaves cells hidden, then the view will include a scroll bar, which enables the user to reposition the view area. In effect, scrolling shifts the view areas and exposes different cells. Scrolling does not change the total number of cells visible in each group, which cannot be changed after a view is created. View areas cannot be scrolled beyond the last row or column of the cell groups they display.

Note There can be at most one horizontal and one vertical view area per cell group. Thus, you cannot introduce multiple scroll areas that show different parts of a single cell group.

As the next figure illustrates, scrolling the top horizontal scroll area down three positions exposes the last row in cell groups 0, 1, and 2. This view area cannot be shifted further, and thus cannot expose cell groups 3, 4, and 5.



Because view areas span the entire view either horizontally or vertically, you are always assured that cells appearing side-by-side in the view are in the same row of the spreadsheet. Likewise, cells that are vertically aligned in the view are in the same column of the spreadsheet.

Locating the Mouse on a View

GXL spreadsheet views have two attributes that together provide the cell location of the mouse. The attributes are:

- gxl-context-help-row
- gxl-context-help-column

These attributes are useful if you are extending the G2 OnLine Documentation (GOLD) module and writing a custom procedure to add symbolic keys, or HelpIDs, to the key table used by GOLD for context-sensitive help. For complete information on extending GOLD for your application, see the *G2 OnLine Documentation User's Guide*.

When a view is passed as the context object, these attributes give the cell location of the mouse when the user asks for help on the view. In most cases, the attributes contain a valid row and column.

If the values of gxl-context-help-row and gxl-context-help-column are -1, the values indicate that the location of the mouse click is out of the current bounds of the spreadsheet dimensions. Such is the case, for example, when the mouse is over a disabled cell.

6

Customizing the Toolbar

Describes how to customize the GXL toolbar display and how to include custom buttons on the toolbar.

Introduction

When you create a view, you can specify three aspects of the appearance of the toolbar:

- The width of the toolbar.
- The tools included in the toolbar.
- The order of the tools in the toolbar.

You can also include your own buttons in the toolbar and create custom buttons.

The root specification object contains the toolbar information for the spreadsheet specification.

Controlling the Toolbar Width

The gxl-tool-bar-width attribute of the root specification controls the width of the toolbar, where the width is measured in pixels. When set to the default value, 0, the width of the toolbar matches the width of the spreadsheet view.

To change the width of the toolbar on a view:

➔ Change the gxl-tool-bar-width attribute of the root specification to the desired width, measured in pixels.

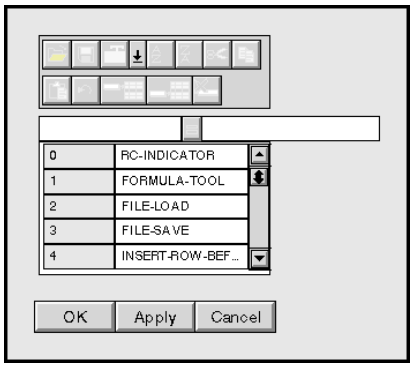
Changing the Display of Tools

The edit spreadsheet tools menu choice of the root specification enables you to change the number of tools displayed on the toolbar, as well as their order.

To change the display of tools on a view:

➔ Choose edit spreadsheet tools from the root specification menu.

The following view appears:



This view displays an editable list of all built-in tools by symbolic name. They are:

Symbol	Tool
rc-indicator	Row-column indicator
formula-tool	Formula bar and equals button
file-load	Load from file button
file-save	Save to file button
insert-row-before	Insert row before selection button
insert-row-after	Insert row after selection button

Symbol	Tool
delete-row	Delete row button
insert-column-before	Insert column before button
insert-column-after	Insert column after button
delete-column	Delete column button
select-color	Select and apply color buttons
cut	Cut button
copy	Copy button
paste	Paste button
undo	Undo button
sort-ascending	Sort ascending button
sort-descending	Sort descending button
ok	OK pushbutton (for edit session)
apply	Apply pushbutton (for edit session)
cancel	Cancel pushbutton (for edit session)

You can change the number of tool buttons included in the toolbar by removing from the list the symbolic entry of the tool you do not want.

To remove entries from the list of built-in tools:

➔ Select the row containing the entry you want to remove and click the delete row button.

Changing the Order of Tools Displayed

The tools appear in order horizontally on the toolbar. You can adjust the order in which the tools appear by changing their order in the list.

To change the order of the tools on the toolbar:

1 Edit rows as necessary and enter the symbolic name of the tool.

Hint Pressing Return advances the cursor and opens the editor on the next row. The list scrolls forward automatically.

---

2 When finished, Click OK and make the edits the permanent initial state.

For example, to move the Undo button to appear after the Sort Descending button, you would:

- Select the row containing the undo symbol and click the cut button.
- Click the delete row button to delete the row.
- Add a row after sort-descending.
- Select the new row and click the paste button to enter undo in the new row.

To verify your changes to the toolbar display:

➔ Click on the spreadsheet object associated with the root specification to display the view, and check the number and order of tools displayed.

Including Your Own Buttons in the Toolbar

GXL has an open design that allows you to add your own buttons to the toolbar. By creating custom buttons, you can add high-level functionality tailored to your specific purposes. You define the functionality of your button, using G2 subclassing and methods. Before attempting to create your own buttons, you should be familiar with creating classes, procedures, and methods in G2, as described in the *G2 Reference Manual*.

To create a custom toolbar button:

1 Create a subclass of gxl-toolbar-button with an appropriate icon and attributes.

2 Write methods for your class that implement the button functionality and animation of the button.

3 Add the custom button to the toolbar.

For an example of creating a toolbar button, see Creating a Custom Button.

Subclassing gxl-toolbar-button

The custom button you create inherits from the class gxl-toolbar-button. When creating a subclass of gxl-toolbar-button, you follow the usual G2 steps for creating an object definition. In most cases, class-specific attributes will not be required, but you can add them as needed.

Creating an Icon

You create an icon for your custom button in the usual manner. To maintain a consistent look with other buttons, keep the icon height at 32 pixels. You must retain the named icon regions that are inherited from the parent class:

face
light-shadow
dark-shadow

See the *G2 Reference Manual* for information on editing the icon.

Adding Help Text

Toolbar buttons can have a help text that appears when the user drags the mouse pointer over the toolbar button. To add help text to the custom button, you must use the text resource facility of Gensym Foundation Resources (GFR), which is a required module of GXL. For information on text resource groups and their use, see the *GFR Reference Manual*.

Creating Methods for Your Button

Three methods implement the functionality of a custom button. Their names are:

- gxl-perform-button-function
- gxl-set-button-state
- gxl-reflect-selection-state-in-button

You define the methods with these names for your custom button class.

gxl-perform-button-function

This method is called when the user selects the button. The signature is:

gxl-perform-button-function
(*button:* class gxl-control-button, *spreadsheet*: class gxl-spreadsheet,
view: class gxl-spreadsheet-view, *window*: class g2-window)

Generally, this method makes calls to the GXL programmer's interface to find the selection state of the view, to get and set data, and to interact with other application objects.

Note Do not include a call next method statement in your gxl-perform-button-function method.

gxl-set-button-state

This method is called in response to a change in the state of the button. The purpose of this method is to manage the icon colors of the button when the state of the button changes. The signature is:

```
gxl-set-button-state
    (button: class gxl-control-button, mode: symbol)
```

where *mode* s is one of the following symbols:

```
depress
release
disable
```

Based on the mode, you can change colors of your button.

When *mode* is disable, in most cases, you should change the color of icon regions to the color extra-light-gray, to be consistent with the other buttons on the toolbar. When the *mode* is depress or release, you should change the color of icon regions to the non-disabled colors.

Note You must include a call next method statement in your gxl-set-button-state method. The next method manages the regions of the icon named light-shadow, dark-shadow, and face, and sets the gxl-button-state attribute of the button.

gxl-reflect-selection-state-in-button

This method is called each time the user selects cells, rows or columns in the view associated with the button. The purpose of this method is to enable or disable the button, depending on the selection state of the view. The signature is:

```
gxl-reflect-selection-state-in-button
    (button: class gxl-control-button, view: class gxl-spreadsheet-view,
     row-is-selected: truth-value, column-is-selected: truth-value,
     cell-is-selected: truth-value, window: class g2-window)
```

Three arguments summarize the current selection state of the view:

- *row-is-selected*, which is true if any row is selected.
- *column-is-selected*, which is true if any column is selected.
- *cell-is-selected*, which is true if any cell is selected.

You set the state of your button by calling gxl-set-button-state, using the information contained in these arguments. If required, you can also find out more details about the selection state of the view, using API functions.

If you do not provide a method by this name, your button will always be enabled.

Note Do not include a call next method statement in the gxl-reflect-selection-state-in-button method.

Adding the Custom Button to the Toolbar

To add the custom button to the toolbar before creating a view:

➔ Add the class name to the spreadsheet tools array in the root specification.

You can add the name of the button to the tools array interactively by choosing edit spreadsheet tools from the root specification menu.

To add the custom button to an existing view:

➔ Use the API procedure gxl-add-tool-to-toolbar.

Creating a Custom Button

The following example, which is from the Custom Button Example of the gxldemo KB, creates an Acknowledge button that automatically fills in cells of a spreadsheet with acknowledgment information. Such a button might be part of a messaging system.

We begin with a spreadsheet that contains four columns: row selectors, messages, who responded, and response. For example:

	Message	Who	When
1	User message #1		
2	User message #2		
3	User message #3		
4	User message #4		
5	User message #5		

The desired behavior is that when a user selects a row containing an unacknowledged message, the Acknowledge button becomes highlighted. Selecting the Acknowledge button:

- Fills the Who field with the user name.
- Fills the When field with the current time.
- Unhighlights (disables) the Acknowledge button.

Defining the Custom Button Class

To define the acknowledge button class:

1 Create an object definition.

2 Specify the symbolic class name acknowledge-button.

3 Make gxl-toolbar-button the direct superior class.

4 Add a region to the inherited icon description called icon-symbol.

Do not delete or change the name of any of the inherited icon regions. Use the characters ACK as the icon symbol.

Adding Help Text

You add help text to the Acknowledge button, using the text resource facility of Gensym Foundation Resources (GFR). If you do not already have a text resource object to store text strings used in your application, create one now.

To set up your GFR text resources:

1 Get the GFR top-level workspace and clone the following objects onto your workspace:

Text resource group

Local text resource

2 Name the text resource group my-resources.

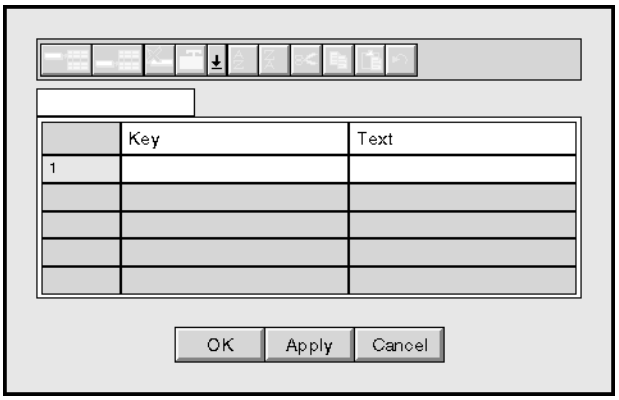
3 Change the gfr-resource-group attribute of the local text resource object to my-resources.

To add a help text string to your button:

1 Get the GXL top-level workspace and make sure that array and list editing is enabled.

2 Choose edit resource from the menu of the local text resource object.

The following spreadsheet appears:



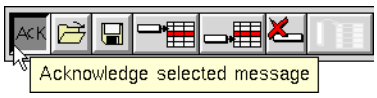
3 Enter acknowledge-button-help under the column labelled Key and acknowledge selected message under the column labelled Text.

4 Click OK and save your edits as the permanent values of your local text resource object.

5 In the attribute initializations of the object definition of the Acknowledge button, add the following:

```
gxl-help-text initially is acknowledge-button-help;
gxl-help-resource initially is my-resources
```

The Acknowledge button now looks like this when you drag the mouse pointer over the button:



If you create another custom button, you do not need to create another text resource group or local text resource object; simply add another key-text pair to the existing local text resource.

Creating Methods for the Acknowledge Button

You must create the following methods for the acknowledge-button class:

- gxl-perform-button-function
- gxl-set-button-state
- gxl-reflect-selection-state-in-button

gxl-perform-button-function Example

The gxl-perform-button-function method, which is called when the button is selected by the user, consists of the following steps:

1. Determine the user name and current time.
2. Determine which rows of the spreadsheet are selected.
3. Iterate over the selected rows and put the responder and time into cells.
4. Refresh the views associated with the spreadsheet.

Note that because gxl-set-cell-contents is being called more than once consecutively, the *update-views* argument to this procedure is false, that is, the views are not redrawn until all the data updates are finished.

```
gxl-perform-button-function (Button: Class acknowledge-button,
Sheet: class gxl-spreadsheet, View: class gxl-spreadsheet-view,
Win: class g2-window)

FirstRow, LastRow, Row: integer;
Timestamp: text = “[the current real time as a time stamp]”;
Responder: text = the text of the g2-user-name of Win;

begin
    FirstRow, LastRow = call gxl-get-selected-row-range (View, Win);
    if FirstRow = -1 then return; {No selection}

    for Row = FirstRow to LastRow do
        call gxl-set-cell-contents (Sheet, Row, 2, Responder, false, Win);
        call gxl-set-cell-contents (Sheet, Row, 3, Timestamp, false, Win);
    end;
    call gxl-refresh-all-views (Sheet, Win);
end
```

gxl-set-button-state Example

The gxl-set-button-state method is quite simple. When the button is disabled, the icon-symbol region is colored extra-light-gray, else the icon-symbol is black:

```
gxl-set-button-state(Button: class acknowledge-button, Mode: symbol

begin
    if Mode = the symbol disable
    then change the icon-symbol icon-color of Button to extra-light-gray
    else change the icon-symbol icon-color of Button to black;
    call next method;
end
```

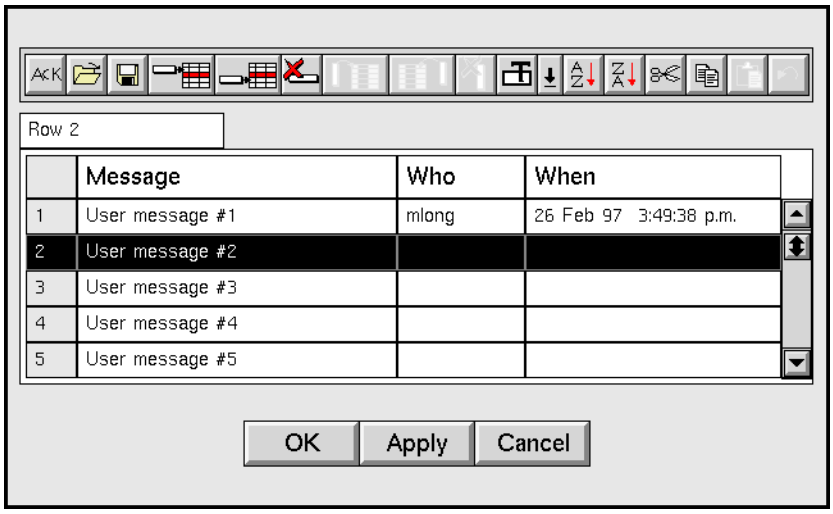
gxl-reflect-selection-state-in-button Example

The gxl-reflect-selection-state-in-button method is complicated by the requirement that the Acknowledge button should only be active when a row is selected that has not already been acknowledged, to prevent a message from being acknowledged more than once. Therefore, the gxl-reflect-selection-state-in-button method checks the cell contents in the responder column before setting the button state:

```
gxl-reflect-selection-state-in-button (Button: class acknowledge-button,
View: class gxl-spreadsheet-view, RowsSelected: truth-value,
ColsSelected: truth-value, CellsSelected: truth-value, Win: class g2-window)

FirstRow, LastRow, Row: integer;
Responder: text;
Sheet: class gxl-spreadsheet;

begin
    if not (RowsSelected)
    then call gxl-set-button-state (Button, the symbol disable)
    else begin
        FirstRow, LastRow = call gxl-get-selected-row-range (View, Win);
        Sheet = call gxl-get-spreadsheet-of-view (View, Win);

        for Row = FirstRow to LastRow do
            Responder = call gxl-get-cell-contents (Sheet, Row, 2, Win);
            If Responder /= "!nv"
            then begin
                call gxl-set-button-state (Button, the symbol disable);
                return:
            end;
        end;
        call gxl-set-button-state (Button the symbol release);
    end;
end
```

Note The text "!nv" is returned by gxl-get-cell-contents if the cell is empty.

This completes the implementation of the custom button. The following figure illustrates the results of selecting the Acknowledge button.



7

Programming GXL

Describes the programmatic interfaces to GXL.

Introduction

You access the full flexibility and power of GXL through its programmatic interface. This interface has been designed to give complete programmatic access to all the features of GXL.

Any function that you can access through the user interface, you can also access programmatically. This gives you, as a G2 developer, the ability to tightly integrate the spreadsheet into your G2 application.

The programmatic interface to GXL can be divided into two basic categories. The first category consists of procedures you write that are called by GXL. Examples include initialization procedures and cell callbacks. The second category, called the application programmer's interface (API) to GXL, includes procedures and methods provided by GXL that you can call at any time.

This chapter describes the various procedures that you provide. Part II, GXL Reference describes the GXL procedures and methods you can call from your application.

Controlling GXL Editing Programmatically

You can programmatically control the GXL editing feature as follows:

- To enable editing, conclude that the logical parameter gxl-matrix-and-array-editing-on = true.
- To disable editing, conclude that the logical parameter gxl-matrix-and-array-editing-on = false.

If you always want this feature to be active when G2 starts, create a rule that says:

```
initially conclude gxl-matrix-and-array-editing-on = true.
```

Controlling the Display of Rows and Columns

When editing G2 lists and arrays, you can control the number of rows and columns displayed in the spreadsheet view by concluding the desired values into the following integer parameters:

- gxl-default-rows-for-editing
- gxl-default-columns-for-editing

By default, the value for both parameters equals 5.

You use gxl-default-columns-for-editing only when editing two-dimensional data, such as a matrix.

Managing Spreadsheet Data Storage

By default, a spreadsheet stores a matrix of values internally. However, you can also store data externally and programmatically send data to the spreadsheet on an "as needed" basis.

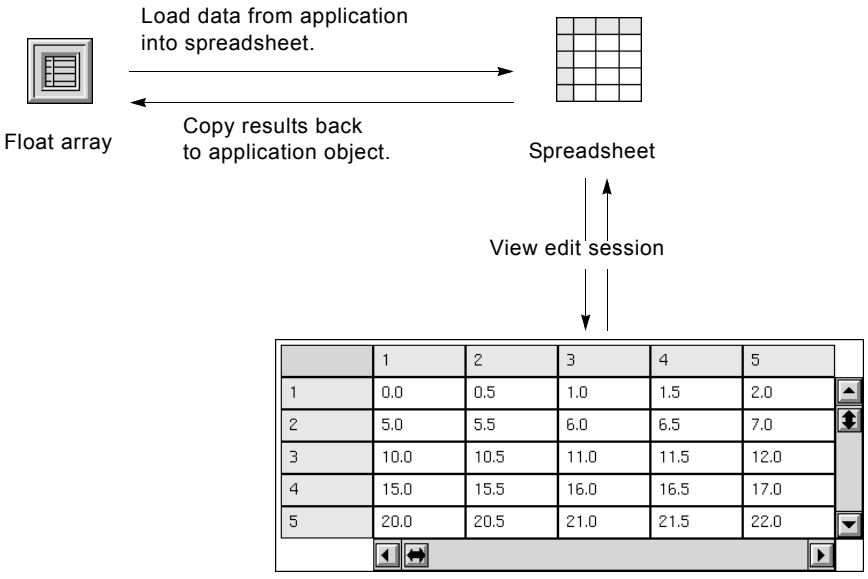
Internal Data Storage

When a spreadsheet stores data internally, you must programmatically:

- Load data into the spreadsheet from your application, before values can be displayed.
- Unload or copy data from the spreadsheet, for changes to be reflected in application objects.

In this mode of operation, the spreadsheet contains a duplicate copy of the information in the application.

As the following figure illustrates, if your application uses a spreadsheet to edit a float array, you first create a spreadsheet, then copy the values from the float array to the spreadsheet. When the user is finished editing the values in the spreadsheet, you then copy the values back to the float array.

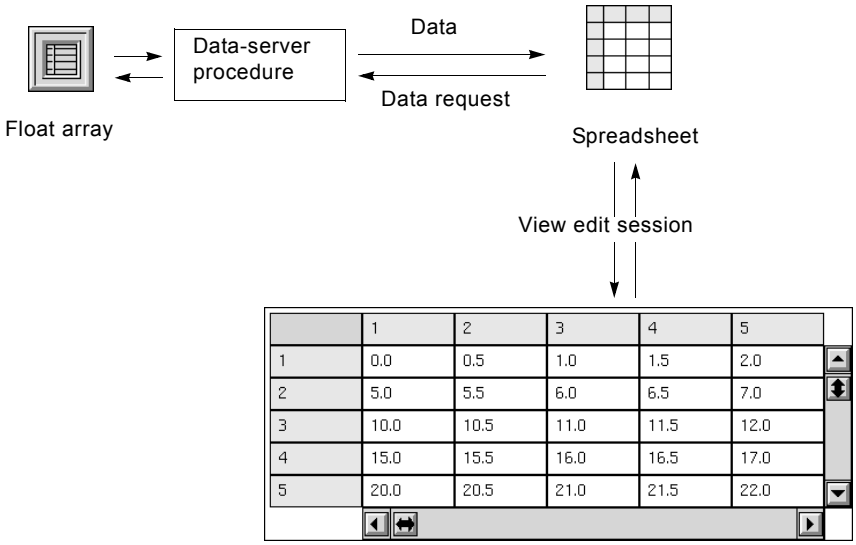


For very large applications, the work of copying the data to the spreadsheet, and storing the information in duplicate may create a performance or memory bottleneck.

External Data Storage

In the second mode of operation, the spreadsheet contains no data internally. Instead, the spreadsheet "pulls" data from the application on an as-needed basis, by calling a data-server procedure that you specify for the gxl-external-data-server attribute of the root specification. In this mode of operation only a single copy of the data exists at any time.

As the following figure illustrates, when a cell scrolls into view, the spreadsheet calls the data-server procedure to determine what value to display in the cell.



	1	2	3	4	5
1	0.0	0.5	1.0	1.5	2.0
2	5.0	5.5	6.0	6.5	7.0
3	10.0	10.5	11.0	11.5	12.0
4	15.0	15.5	16.0	16.5	17.0
5	20.0	20.5	21.0	21.5	22.0

In this mode of operation, the memory and time needed to create a copy of the application's data is avoided. However, there is extra overhead each time the spreadsheet needs to access data, so operations like scrolling are somewhat slower. This should be considered an "advanced" mode of operation because it requires the user to write more supporting code.

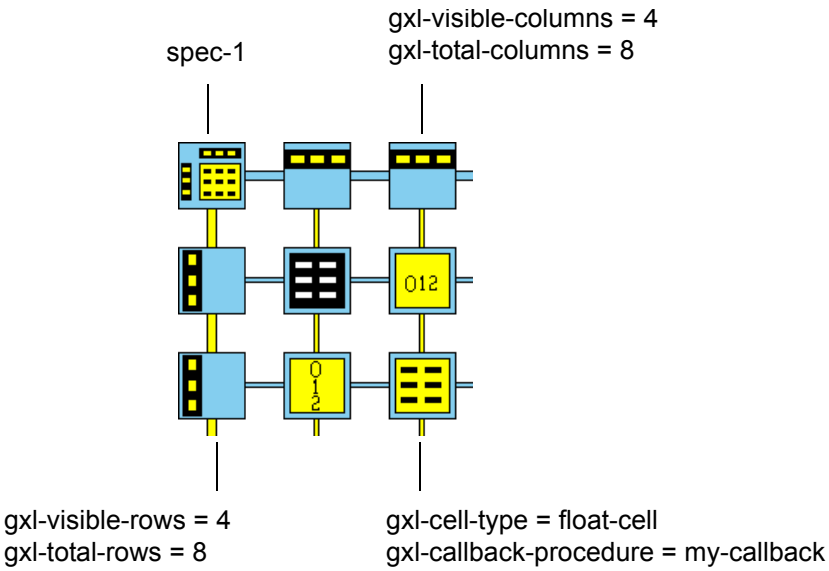
For information on the procedures you must supply if you configure a spreadsheet specification to obtain its data externally "on demand," see External Data and Color Server Procedures.

Manipulating a Spreadsheet Programmatically

Once a spreadsheet has been created, you can programmatically manipulate its properties, populate it with data, and run callbacks when the user enters data.

The following example shows how some of these techniques are used to create a spreadsheet whose cell colors reflect the numerical values contained in the spreadsheet. If the value is less than 50.0, the color will be green, if above 50.0 but below 100.0, the color will be yellow, and if the value is above 100.0, the color will be red.

First, create the following spreadsheet specification, and name the spreadsheet specification spec-1:



Using the technique shown in the previous example, remove the OK, Apply and Cancel buttons from the toolbar that will appear on the view.

Each cell group can name a cell callback procedure, which is a user-written procedure that GXL calls when data is entered into the cell group (see Cell Callback Procedures). In this example, the callback procedure named my-callback has been specified in the data cell group. Here is the callback procedure that updates the color of the cell based on the value contained in the cell:

```
my-callback(Sheet: class gxl-spreadsheet, Row: integer,
    Column: integer, NewValue: value, Window: class g2-window)

begin
    If Val < 50.0 then call gxl-set-color-pattern-of-cell
        (Sheet, Row, Column, the symbol green, the symbol
        black, the symbol black, true, Window)
    else if Val >= 50.0 and Val < 100.0 then call gxl-set-color-
        pattern-of-cell (Sheet, Row, Column, the symbol yellow,
        the symbol black, the symbol black, true, Window)
    else call gxl-set-color-pattern-of-cell(Sheet, Row, Column,
        the symbol red, the symbol black, the symbol black,
        true, Window)
end
```

Performing these steps results in a spreadsheet whose cell colors reflect the magnitude of the values they contain.

Suppose we want to create a view on an existing workspace, workspace-1, and keep the view as a permanent part of an application. The following code fragment creates a spreadsheet and a view from spec-1, and makes them permanent:

```
Sheet = call gxl-create-spreadsheet(spec-1, Win);
transfer Sheet to the workspace of spec-1 at (the item-x-position of
    spec-1 - 100, the item-y-position of spec-1)
call gxl-make-spreadsheet-permanent(Sheet, Win);
View = call gxl-create-spreadsheet-view(Sheet, spec-1, workspace-1,
    0, 0, Win);
call gxl-make-spreadsheet-view-permanent(View, Win);
```

The view can be displayed by showing workspace-1 on any G2 window. Generally, you will create a view for each G2 window where a view is required, because if you show a single view on multiple windows, the views cannot be scrolled independently.

User Procedures Called by GXL

You can provide a number of procedures that GXL will call at various stages during the creation or operation of a spreadsheet. These procedures include:

- Initialization and re-initialization procedures that set cell contents when the spreadsheet is created, or when dimensions of the spreadsheet are changed.
- Procedures that validate entries into cells.
- Callback procedures, called by GXL when cells are selected, when the contents of a cell changes, when the user finishes editing a cell, or when views are scrolled.
- Matrix extension procedures, used if you are unloading data from a spreadsheet into a matrix.
- Data server procedures, called when the spreadsheet needs to access a cell value, if the spreadsheet does not store its values internally.

All of these procedures are optional, except data server procedures when a spreadsheet does not store data internally. If you provide any of these procedures, you should test your procedures carefully. If there is a run-time error in your procedure, GXL may not be able to recover and continue processing.

Initialization Procedures

When a spreadsheet is created, you may want to initially fill parts or all of the spreadsheet with values. For example, you may want to have the days of the week or times of day entered into a row or column of the spreadsheet. Or, you may want to initialize a group of integer cells to 0, or default truth-value cells to true or false.

Specifying an Initialization Procedure

The gxl-initialization-procedure attribute of a cell group names the initialization procedure. Each cell group can have its own initialization procedure that is called when a spreadsheet is created.

If you use the built-in initialization procedure, gxl-default-initialization, each cell in the group will be initialized with a standard G2 default value according to the type of cell in the cell group, as summarized on .

The default value, the symbol unspecified, should be used if you do not provide an initialization procedure for the group.

To specify an initialization procedure for a cell group:

➔ Change the gxl-initialization-procedure attribute of the corresponding cell group specification object to the name of the procedure you want GXL to call.

If you provide your own initialization procedure, the signature of your procedure must be:

```
my-initializer
    (sheet: class gxl-spreadsheet, group-number: integer,
     init-data: value, first-row: integer, first-col: integer,
     n-rows: integer, n-cols: integer, window: class g2-window)
```

The arguments passed to your initialization procedure are:

Argument	Description
sheet	The spreadsheet that is being initialized.
group-number	The reference number of the cell group that is being initialized.
init-data	The value specified in the attribute gxl-cell-group-initialization-data of the cell group specification object of the cell group being initialized.
first-row	The first row of the uninitialized area of the cell group.
first-col	The first column of the uninitialized area of the cell group.
n-rows	The number of rows in the uninitialized area of the cell group.

Argument	Description
n-cols	The number of columns in the uninitialized area of the cell group.
window	The g2-window originating this call.

For an initialization procedure, the *FirstRow*, *FirstCol, NRows,* and *NCols* arguments are the starting coordinates and dimensions of the cell group, because the entire cell group is uninitialized. Inside your procedure, you can use the spreadsheet, group number, and other arguments, along with GXL's API to access properties of the spreadsheet and set values, colors, etc. of the cells in the cell group.

For example, suppose you want to initialize a cell group with the days of the week, from Monday through Friday. Assume the spreadsheet has been laid out so the cell group to contain these values has one row and five columns. Your initializer procedure should look like this:

```
days-of-week-initializer(Sheet: class gxl-spreadsheet, GroupNumber:
    integer, InitData: value, FirstRow: integer, FirstCol: integer,
    NRows: integer, NCols: integer, Win: class g2-window)

begin
    call gxl-set-cell-contents(Sheet, FirstRow, FirstCol, “Monday”,
        false, Win);
    call gxl-set-cell-contents(Sheet, FirstRow, FirstCol + 1, “Tuesday”,
        false, Win);
    call gxl-set-cell-contents(Sheet, FirstRow, FirstCol + 2, “Wednesday”,
        false, Win);
    call gxl-set-cell-contents(Sheet, FirstRow, FirstCol + 3, “Thursday”,
        false, Win);
    call gxl-set-cell-contents(Sheet, FirstRow, FirstCol + 4, “Friday”,
        false, Win);
    call gxl-refresh-all-views(Sheet, Win);
end
```

Reinitialization Procedures

You can provide an optional reinitialization procedure that GXL will call whenever the dimensions of a cell group change by the addition or removal of rows or columns. By default, anytime GXL adds rows or columns to a spreadsheet, they are initially empty. Additionally, if there are row or column selectors, GXL renumbers them, if necessary.

You use a reinitialization procedure whenever you want to initialize the values in new rows or columns, or perhaps to change values when rows and columns are deleted.

Specifying a Reinitialization Procedure

You specify a reinitialization procedure, using the gxl-reinitialization-procedure attribute of the appropriate cell group. Each cell group can have its own reinitialization procedure.

To specify a reinitialization procedure for a cell group:

➔ Change the gxl-reinitialization-procedure attribute of the corresponding cell group specification object to the name of the procedure you want GXL to call.

The default value is the symbol unspecified. Once the spreadsheet has been created from its specification, you can call gxl-set-cell-group-procedure-attribute to change the reinitialization procedure.

The signature of a reinitialization procedure is the same as an initialization procedure. You can, in fact, use the same procedure for both initialization and reinitialization of a cell group if you choose to do so. When a reinitialization procedure is called, the *first-row*, *first-column, nrows, and ncols* arguments represent the uninitialized area of the cell group (the coordinates of the new rows or columns). However, if the reinitialization procedure is being called because a row or column has been deleted, these four arguments are all passed to the procedure as -1.

Validation Procedures

When a cell receives a manual entry, the entry is automatically validated and illegal entries are rejected. The acceptable entry depends on the cell type. While value cells accept any entry, you cannot, for example, enter a text into an integer cell or a symbol into a truth value cell.

Sometimes, validation by data type is not enough. For example, you may want to have a text field that only accepts specific text values, such as the names of specific people working on a project. You may want to enter values from formula evaluation, or you may want to limit a quantity cell to only accept positive quantities.

To introduce your own validation, you specify a procedure that GXL will call after a user has entered a value into a cell, but before GXL actually assigns the new value to the cell.

Specifying a Validation Procedure

You specify the custom validation procedure that GXL will use to validate the data input for all cells of a cell group with the gxl-additional-validation-procedure attribute. Each cell group can have its own validation procedure.

To specify an additional validation procedure for a cell group:

➔ Change the gxl-additional-validation-procedure attribute of the corresponding cell group to the name of the procedure you want GXL to call.

In a validation procedure, you can:

- Perform any tests on the value, and accept or reject it. If you reject the value, GXL will optionally post an alert message to the user with a text you provide.
- Return a value different from the value passed to the procedure, which becomes the value actually put into the cell. You could use this capability to round off a float value to some fixed precision, to expand abbreviations, etc.

The signature of a validation procedure is:

```
my-validator
    (sheet: class gxl-spreadsheet, row: integer, column: integer,
     new-value: value, window: class g2-window
    -> value-or-alert: value, outcome: truth-value
```

The arguments provided to a validation procedure are:

Argument	Description
sheet	The spreadsheet that is receiving input from the user.
row	The row of the spreadsheet where the user input has been received.
column	The column of the spreadsheet where the user input has been received.
new-value	The new value that has been input by the user.
window	The g2-window originating this call.

The values that your procedure must return are:

Return Value	Description
value-or-alert	The value that is to be placed into the cell, if the value is accepted, or a text string to be used as an alert to the user, if the value is not accepted.
outcome	The truth-value indicating whether the value is accepted, where true indicates that the value has been validated.

The first return value is most often the same value that is passed to the procedure, if the validation passes. If the new value is rejected, you can return a text string which is posted to the user as a popup dialog. Typically, you advise the user why the input has been rejected, and inform them of the acceptable values. However, if you do not want a popup dialog to appear when the value is rejected, return an empty text string, "".

Here is a simple validation procedure which only accepts positive integers:

```
positive-integer-validator(sheet: class gxl-spreadsheet, row: integer,
    column: integer, newvalue: value, window: class g2-window) =
    (value, truth-value)

begin
if newvalue is not an integer or newvalue <= 0 then return
    "A positive integer was expected", false else return
    newvalue, true;
end
```

GXL always validates the basic value type before calling the user validation procedure. Therefore, your procedure will receive a *NewValue* which is commensurate with the cell type where the data was entered. For a discussion on data validation, see Validating Data Input.

Cell Callback Procedures

Cell callbacks are procedures you provide to respond to cells receiving new values. GXL calls these procedures whenever cells receive new values by manual input or via the API procedure, gxl-set-cell-contents.You can specify a different callback procedure for each cell group.

Callback Restrictions

When using callback procedures, consider the following restrictions:

- Callbacks are not used when you load data into a spreadsheet via gxl-load-data-into-defined-area or gxl-load-data-into-cell-group.
- If a user loads data from a file with the Load from File button on the toolbar, callbacks on the cells are not activated.
- There are no callbacks when cell colors are changed.

Specifying a Callback Procedure

You specify a callback procedure for a cell group before the spreadsheet is created by naming the procedure, using the gxl-initialization-procedure attribute of a cell group. Each cell group can have its own initialization procedure that GXL calls when creating the spreadsheet.

To specify the callback of a cell group before a spreadsheet is created:

➔ Change the gxl-callback attribute of the cell group specification object to the name of the procedure you want GXL to call

You assign a callback procedure to a cell group, not individual cells. The signature of a callback procedure is:

```
my-callback
    (sheet: class gxl-spreadsheet, row: integer, column: integer,
     new-value: value, window: class g2-window)
```

The arguments provided to a callback procedure are:

Argument	Description
sheet	The spreadsheet that received a new cell value.
row	The row coordinate of the spreadsheet that received a new value.
column	The column coordinate of the spreadsheet that received a new value.
new-value	The new value now contained by the cell.
window	The g2-window originating this call.

Inside a callback procedure, you can perform any G2 actions. You may want to update other cells in the same spreadsheet, or change the colors of the cell that received the value. Or, you could use the callback to trigger communications with a user, or do processing in other objects in the KB.

Note While processing is taking place inside your callback, the spreadsheet processing is suspended. Therefore, in general, you should not place wait states inside your callback. The safe way to introduce wait states into a callback is to have your callback start another procedure that contains the wait state.

Changing a Callback Procedure of an Existing Spreadsheet

To change the callback procedure of a cell group for an existing spreadsheet:

➔ use the API procedure gxl-set-cell-group-procedure-attribute.

Activating and Deactivating Cell Callbacks

Sometimes, you might want to set cell values without activating the callbacks. You can dynamically activate or deactivate cell callbacks in a spreadsheet by changing the gxl-callbacks-enabled attribute of the spreadsheet. By default, this attribute is true for any spreadsheet.

To deactivate all callbacks for an existing spreadsheet:

➔ Change the gxl-callbacks-enabled attribute of the spreadsheet to false.

Changing this attribute to true re-activates all callback procedures of the spreadsheet.

For an example of using a callback procedure, see Manipulating a Spreadsheet Programmatically.

Selection Callback Procedures

Selection callbacks are procedures you provide to respond when cells are selected. GXL calls your selection callback procedure in response to:

- Manual selection of one or more cells on a spreadsheet view.
- Programmatic selection, using gxl-set-selection-limits.

Specifying a Selection Callback Procedure

You can specify a different selection callback for each cell group, or specify a single callback for all selections made anywhere on a view.

To specify a selection callback of a cell group before a spreadsheet is created:

➔ Change the gxl-selection-callback-procedure attribute of the corresponding cell group specification object to the name of the procedure you want GXL to call.

The signature of the selection callback procedure that you provide is:

```
my-selection-callback
    (sheet: class gxl-spreadsheet, view: class gxl-spreadsheet-view,
    first-row: integer, first-col: integer, n-rows: integer,
     n-cols: integer, window: class g2-window)
```

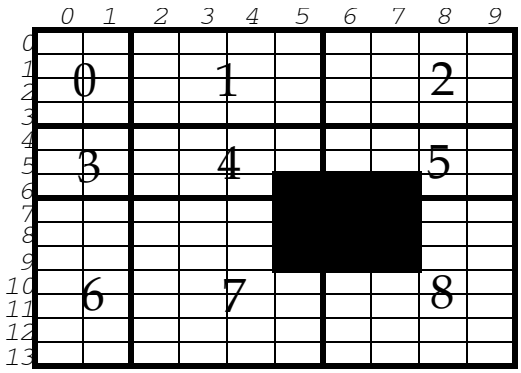
The arguments provided to the selection callback procedure are:

Argument	Description
sheet	The spreadsheet corresponding to the view where the selection has taken place.
view	The view where the selection has taken place.
first-row	The first row of the selection within the cell group requesting this callback.
first-col	The first column of the selection within the cell group requesting this callback.
n-rows	The number of rows in the selection within the cell group requesting this callback.
n-cols	The number of columns in the selection within the cell group requesting this callback.
window	The g2-window originating this call.

When the selection is a single cell or multiple cells within a single cell group, GXL calls the selection callback once. The arguments to the selection callback provide information on the extent of the selection.

When the selection spans multiple cell groups, each cell group involved in the selection receives one call to its selection callback, if it has one. The arguments of the callback indicate the cells within the cell group that have been selected.

For example, the following spreadsheet view displays a selection as indicated in black:



The four callbacks are:

- To the selection callback of cell group 4, where:

  first-row = 6, *first-col* = 5, *n-rows* = 1, *n-cols* = 1

- To the selection callback of cell group 5, where:

  first-row = 6, *first-col* = 6, *n-rows* = 1, *n-cols* = 2

- To the selection callback of cell group 7, where:

  first-row = 7, *first-col* = 5, *n-rows* = 3, *n-cols* = 1

- To the selection callback of group 8, where:

  first-row = 7, *first-col* = 6, *n-rows* = 3, *n-cols* = 2

To change the selection callback of a cell group for an existing spreadsheet:

➔ Use the API procedure gxl-set-cell-group-procedure-attribute.

Using a Single Selection Callback on a View

Alternately, you can define a selection callback procedure in the root specification that GXL calls in response to every selection event on a view of the spreadsheet.

To specify a single selection callback for use with every selection event:

➔ Change the gxl-selection-callback attribute of the root specification to the name of the procedure you want GXL to call.

Return and Tab Key Handler Procedures

When a user finishes manually editing a cell using the return key or the tab key, GXL advances to the next editable cell within the cell group. For a discussion of the default movement in response to the return key and tab key, see Moving the Editor within a Cell Group.

To summarize the default movement:

- In response to the return key, GXL advances to the cell directly below the current cell, unless it is in the last row of a cell group, in which case GXL advances to the first row of the next column in the cell group.
- In response to the tab key, GXL advances to the next column in the cell group, or to the first column of the next row in the cell group.
- GXl does not respond to pressing the return or tab keys if the cell is the last cell in the cell group.

You can override this behavior by providing a return and tab key handler procedure that explicitly directs GXL to the next editable cell.

Note that the only way you can make GXL cross cell group boundaries when advancing the editor is by providing your own return and tab key handler.

Specifying a Return and Tab Key Handler

You specify a custom return and tab key handler procedure in the root specification object of a spreadsheet specification.

To specify a return and tab key handler procedure:

➔ Change the gxl-return-and tab-key-handler attribute of the root specification to the name of the procedure you want GXL to call.

The signature of the return and tab key handler procedure that you provide is:

```
my-return-and-tab-key-handler
    (sheet: class gxl-spreadsheet, view: class gxl-spreadsheet-view,
    key: symbol, row: integer, column: integer, window: class g2-window)
    -> (row: integer, column: integer)
```

The arguments provided to the return and tab key handler procedure are:

Argument	Description
sheet	The spreadsheet corresponding to the view on which the return or tab key has been used.
view	The view on which the return or tab key has been used.
key	The symbol corresponding to the key used: return or tab
row	The row location of the cell whose edit has just been completed.
column	The column location of the cell whose edit has just been completed.
window	The g2-window originating this call.

The values that your procedure must return are:

Return Value	Description
row	The row location of the next cell to be edited.
column	The column location of the next cell to be edited

If the editor should not advance to another cell, the procedure should return -1, -1.

Scrolling Callback Procedures

In the root specification, you can name procedure that GXL calls whenever a view is scrolled to a new position. You use this procedure to respond to scrolling events in any way necessary in your application.

Specifying a Scrolling Callback Procedure

You specify a custom scrolling callback procedure in the root specification object of a spreadsheet specification.

To specify a scrolling callback procedure:

➔ Change the gxl-scroll-callback attribute of a root specification to the name of the procedure you want GXL to call in response to a scrolling event.

The signature of the scrolling callback procedure that you provide is:

```
my-scroll-callback
    (sheet: class gxl-spreadsheet, view: class gxl-spreadsheet-view,
    controller: integer, scroll-area: symbol, scroll-position: integer,
     window: class g2-window)
```

The arguments provided to the selection callback procedure are:

Argument	Description
sheet	The spreadsheet corresponding to the *View* where the scrolling event occurred.
view	The view where the scrolling event occurred.
controller	The reference number (starting at 0) of the row or column controller where the scrolling event occurred.
scroll-area	One of the following symbols: • row if the view has been scrolled to a new vertical position. • column if the view has been scrolled to a new horizontal position.

Argument	Description
scroll-position	The reference number (starting with 0) of the scroll position. The scroll position is 0 if you are at the top or left side of a scrollable area, and increments 1 per row or column exposed.
window	The g2-window originating this call.

For example, if we have six cell groups laid out 2 row controllers by 3 column controllers, then and we scroll cell groups 2 and 5 using the horizontal scroll bar at the bottom of the group, then the symbol in the call is COLUMN, and the controller number is 2.

Matrix Extension Procedures

A matrix consists of an item-array or item-list, or "spine," containing vectors which constitute the "ribs" of the matrix. Using gxl-unload-data-from-defined-area or gxl-unload-data-from-cell-group, you can copy data from a spreadsheet area or cell group into a matrix.

When copying spreadsheet data into a matrix, GXL might have to extend the matrix to accommodate the data in the spreadsheet. However, GXL does not have the necessary information to decide what class of vector it should create as ribs of the extended spine.

For example, even if the spreadsheet contains all integers, it is not clear whether GXL should create an integer-array or integer-list, or a user-defined subclass of integer-array or integer-list.

In general, therefore, you need to provide a procedure that GXL can call to extend a matrix when it is unloading data from the spreadsheet into the matrix. Your matrix extension procedure must add the proper number and type of arrays or lists to the spine of the matrix.

Specifying a Matrix Extension Procedure

You specify a custom matrix extension procedure with the gxl-matrix-extension-procedure attribute of the root specification.

To specify your matrix extension procedure:

➔ Change the gxl-matrix-extension-procedure attribute of the root specification to the name of your procedure.

If the gxl-matrix-extension-procedure attribute is set to the symbol unspecified, GXL will use the most specific type of array (not list) that accommodates the data contained in the corresponding row or column of the spreadsheet.

For details on the signature of the procedure you provide, and a discussion on copying spreadsheet data to a matrix, see Unloading Two-Dimensional (Matrix) Data.

External Data and Color Server Procedures

GXL spreadsheets can store data internally or depend on the application to provide data to the spreadsheet on demand. For a discussion on these two modes of data storage and their performance and memory characteristics, see Managing Spreadsheet Data Storage.

If you have configured your spreadsheet to obtain its data from the application on demand (root specification attribute gxl-store-data-internally = false), you must provide a procedure to get the externally stored value of a cell. You name this procedure in the gxl-external-data-server attribute of the root specification.

The gxl-external-color-server attribute of the root specification, which specifies a procedure that gets the color pattern of a cell, can be unspecified.

Specifying External Data and Color Server Procedures

You specify the custom external data and color server procedures in the root specification object of a spreadsheet specification.

To specify the external data server and color server:

➔ Change the following attributes in the root specification to the symbolic names of the procedures GXL should call.

- gxl-external-data-server
- gxl-external-color-server

External Data Server Procedure

GXL calls the external data server procedure whenever it needs to have the current value of a cell. This could be when the cell value is displayed, or when the value is referenced by sorting, or by a formula.

The signature of the external data server procedure that you provide is:

```
my-data-server
    (sheet: class gxl-spreadsheet, row: Integer, column: Integer)
    -> (contents: value)
```

The arguments provided to the external data server procedure are:

Argument	Description
sheet	The spreadsheet for which the current value is needed.
row	The row location of the cell for which the value is needed.
column	The column location of the cell for which the value is needed.

The value that your procedure must return is:

Return Value	Description
contents	The current value for the given *Row-Column* location.

External Color Server Procedure

GXL calls the color server procedure whenever a cell needs to be displayed on a view. The procedure returns the background, text and border colors as symbols.

The signature of the external color server procedure that you provide is:

```
my-color-server
    (sheet: class gxl-spreadsheet, row: Integer, column: Integer
    -> (background-color: symbol, text-color: symbol, border-color: symbol)
```

The arguments provided to the external color server procedure are:

Argument	Description
sheet	The spreadsheet corresponding to the view whose cell is to be displayed.
row	The row location of the cell to be displayed.
column	The column location of the cell to be displayed.

The values that your procedure must return are:

Return Value	Description
background-color	The background color of the cell.
text-color	The text color of the cell.
border-color	The border color of the cell.

If you want the default color pattern applied to the cell, your color server should return the symbols default, default, default.

When returning a color pattern for the cell, you cannot mix a return of default with other non-default colors.

Assigning Views to G2 Windows

Views are an assembly of objects on a conventional KB workspace. Using Telewindows, it is possible to show a workspace containing a view on multiple windows. However, to do so would defeat the purpose of the view, which is that each client should be able to scroll and perform other operations independently of other clients. The simple rule to follow is this:

> When you need to display data from a spreadsheet on multiple windows, give each window its own view.

There is one exception: if a view is read-only and cannot scroll, you can safely show it on multiple windows. In such a case, every client sees the same thing, eliminating any interference between different clients.

If you use the programmatic interface to GXL, the mechanism for assigning views to windows is through G2's show command. You can create any number of views of a spreadsheet using the API procedure, gxl-create-spreadsheet-view, which is described on . Then, you simply show the workspace of a particular view on the desired g2-window.

The Application Programmer's Interface

All interactions with the spreadsheet module take place through a small set of specially-designated "public interface" procedures, methods, classes, and attributes. These items are referred to as the Application Programmer's Interface (API).

Because G2 does not have a mechanism to distinguish public and private classes and attributes, GXL uses a naming convention to help you differentiate the API, from the internal parts of GXL. The convention is simply this:

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

Only items whose names begin with gxl- are part of the public interface to GXL. A public class may have private attributes. Unless otherwise documented, public classes must not be subclassed.

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

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

In some cases, the API allows you to do things you cannot do from the graphical user interface, for example, overriding the protections on a view, setting cell values without type checking, and suppressing view updating. Because of these capabilities, use of the programmatic API requires a greater understanding of the design concepts of GXL than does manual use. You should be thoroughly familiar with the nomenclature of GXL, and the basics of specifications, spreadsheets, and views before using the API.

All API procedures require a g2-window as their last argument. In general, this is the window where the call originated. The purpose of passing the window is to identify the client for whom the action is being carried out. User mode, language and other properties of the client are accessed through the window argument. When you begin a thread of processing from an action button or user menu choice, the window is accessible with the this window syntax in the action of the button or menu choice. Only when you start a procedure through a rule is there a problem associating a window with a thread of processing. In this case, use gfr-default-window as the window argument.

Workspaces containing the GXL programmer's interface of procedures and methods are available from the GXL top-level workspace.

Accessing the API

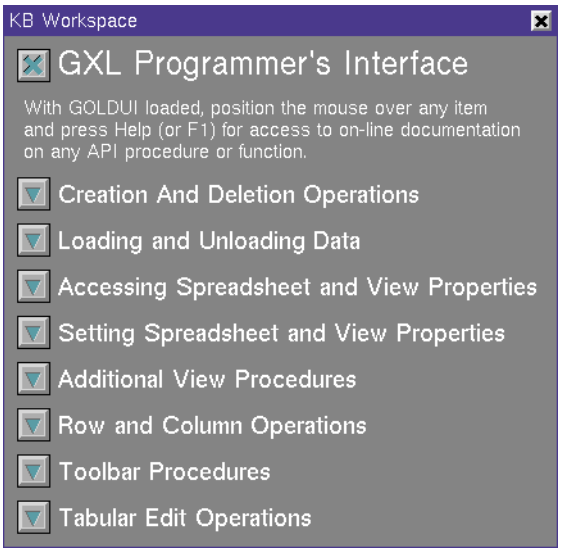
You can access the API procedures and methods from the GXL top-level workspace.

To access the API to GXL:

1 Choose Main Menu > Get Workspace > gxl-top-level.

2 Click the button next to Gxl-Programmers-Interface.

The following subworkspace appears:



3 Click the button next to the category you want to display.

Each category has a corresponding chapters in Part II, which describes each related API procedure or method in detail.

The API for...	Begins on...
Creation and deletion operations	
Loading and unloading data	
Accessing spreadsheet and view properties	
Setting spreadsheet and view properties	
Additional view procedures	
Row and column operations	

The API for...	Begins on...
Toolbar procedures	
Tabular edit operations	

Part II

API Reference

Chapter 8: Creation and Deletion Operations

Describes the API procedures for creating and deleting GXL spreadsheets and views.

Chapter 9: Loading and Unloading Data

Describes the API procedures for loading and unloading spreadsheet data.

Chapter 10: Accessing Spreadsheet and View Properties

Describes the API procedures for accessing GXL spreadsheet and view properties.

Chapter 11: Setting Spreadsheet and View Properties

Describes the API procedures for accessing GXL spreadsheet and view properties.

Chapter 12: Additional View Procedures

Describes miscellaneous API procedures related to GXL views.

Chapter 13: Row and Column Operations

Describes the API methods for adding and deleting rows and columns, and API procedures for sorting.

Chapter 14: Toolbar Procedures

Describes the API procedures for managing the appearance of toolbars on GXL spreadsheet views.

Chapter 15: Tabular Edit Operations

Describes the API procedures for launching and managing GXL edit sessions.

8

Creation and Deletion Operations

Describes the API procedures for creating and deleting GXL spreadsheets and views.

Introduction

This chapter describes the procedures of the Application Programmer's Interface to GXL that you can use to programmatically create and delete spreadsheets and views.

The basic procedure for creating a spreadsheet is gxl-create-spreadsheet. Once a spreadsheet has been created, you can create one or more views using gxl-create-spreadsheet-view.

Only permanent spreadsheets and views can survive G2 resets. To make a spreadsheet or view permanent, use the procedures gxl-make-spreadsheet-permanent or gxl-make-spreadsheet-view-permanent.

You can clone a spreadsheet programmatically, using gxl-clone-spreadsheet. However, you cannot clone a spreadsheet view.

To create and display a spreadsheet with a simple format in a single step, use gxl-create-and-display-simple-spreadsheet. To display a view of a given spreadsheet on a new workspace, use gxl-create-and-display-spreadsheet-view.

To delete a spreadsheet or view programmatically, use the procedures gxl-delete-spreadsheet and gxl-delete-view.

The following procedures work with specifications as organized sets of lists to create spreadsheets and views:

- gxl-collect-specification parses an existing specification layout on a workspace and returns the layout as an organized set of lists.
- gxl-create-spreadsheet-from-collected-specification creates a spreadsheet from a specification in the form of an organized set of lists.
- gxl-create-spreadsheet-view-from-collected-specification creates a spreadsheet view from an organized set of lists.

You can also construct a specification on a workspace from an organized set of lists, using gxl-layout-specification.

See The Application Programmer's Interface for related information.

gxl-clone-spreadsheet

Produces a spreadsheet by cloning another spreadsheet.

Synopsis

gxl-clone-spreadsheet
(*sheet*: class gxl-spreadsheet, *window*: class g2-window)
-> *spreadsheet*: class gxl-spreadsheet

Argument	Description
sheet	The spreadsheet that is to be cloned.
window	The g2-window originating this call.

Return Value	Description
spreadsheet	The spreadsheet created by this call.

Description

This procedure creates a new spreadsheet that has properties derived from the given *sheet*:

- The transient dimensions, data values and cell formats in the cloned spreadsheet are the same as the source spreadsheet.
- The permanent dimensions, data values, and cell formats in the cloned spreadsheet are the same as the source spreadsheet.
- The cloned spreadsheet does not have any views, even if the source spreadsheet had one or more views.
- The cloned spreadsheet does not have a name.

Example

The following call creates a spreadsheet named spreadsheet-2, based on spreadsheet-1:

```
spreadsheet-2 = call gxl-clone-spreadsheet(spreadsheet-1, win);
```

gxl-collect-specification

Parses a specification layout on a workspace and returns the layout as an organized set of lists.

Synopsis

gxl-collect-specification
(*root-spec*: class gxl-root-specification, *window*: class g2-window)
-> *row-controllers*: class: item-list, *column-controllers*: class: item-list, *cell-groups*: class: item-list

Argument	Description
root-spec	The root specification whose workspace layout is returned as a set of lists.
window	The g2-window originating this call.

Return Value	Description
row-controllers	An item list of the row-controllers of the specification
column-controllers	An item list of the column-controllers
cell-groups	An item-list of the cell-groups

Description

This procedure creates three item-lists: one each for the cell-groups, row-controllers, and column-controllers of the particular specification. Starting with the given *root-spec*, it checks for connected row-controllers and inserts them in consecutive order in the *row-controllers* item-list. Next, it checks for connected column-controllers, and inserts them in consecutive order in the *column-controllers* item-list. Finally, based on the number of row-controllers, it checks for connected cell-groups in each row and inserts them in order in the *gxl* item-list.

If the total number of elements in the *cell-groups* item-list is zero (0) or does not equal the total number of elements in *column-controllers* times the total number of elements in *row-controllers*, this procedure signals an error (gxl-bad-specification-structure).

Example

The following call parses the layout of root-spec-1 and returns item-lists of its row controllers, column controllers and cell groups.

```
RowCont, ColCont, CellGroup = call gxl-collect-specification (root-spec-1, win);
```

gxl-create-and-display-simple-spreadsheet

Creates a spreadsheet with one editable area with given dimensions, font size, and cell type, and displays a view on a window.

Synopsis

gxl-create-and-display-simple-spreadsheet
(*cell-type:* symbol, *total-rows*: integer, *total-columns*: integer,
visible-rows: integer, *visible-columns*: integer, *cell-width*: integer,
cell-height: integer, *font-size*: symbol, *window*: class g2-window)
-> *spreadsheet*: class gxl-spreadsheet, *view*: class gxl-spreadsheet-view

Argument	Description
cell-type	The type of data cell in the editable area, which can be one of the following symbols: value-cell, quantity-cell, float-cell, integer-cell, truth-value-cell, symbol-cell or text-cell
total-rows	The number of rows in the spreadsheet.
total-columns	The number of columns in the spreadsheet.
visible-rows	The number of data rows to show in the view.
visible-columns	The number of data columns to show in the view.
cell-width	The width of the cells, in pixels.
cell-height	The height of the cells, in pixels.
font-size	The font size for the data cells, either the symbol small, large or extra-large.
window	The g2-window where the spreadsheet view is to be shown.

Return Value	Description
spreadsheet	The spreadsheet created by this call.
view	The view created by this call.

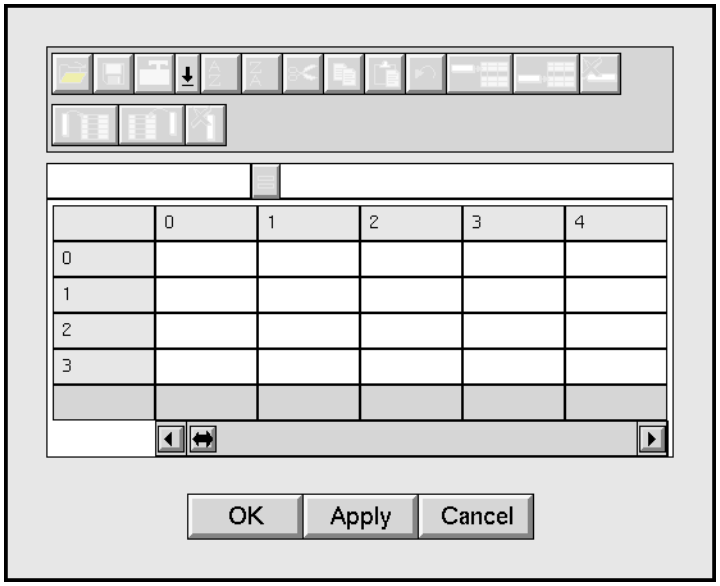
Description

This procedure creates a spreadsheet that has row selectors in the first column and column selector cells in the first row, and a single rectangular data area. It displays a view of the spreadsheet on the given *window*.

Example

The following example creates the view shown in the figure:

```
spreadsheet, view = call gxl-create-and-display-simple-spreadsheet
(the symbol float-cell, 4, 10, 5, 5, 80, 28, the symbol small, win);
```



gxl-create-and-display-spreadsheet-view

Creates a spreadsheet view on a new workspace, and shows the workspace on a window.

Synopsis

gxl-create-and-display-spreadsheet-view
(*sheet*:class gxl-spreadsheet, *root-spec*: class gxl-root-specification,
window: class g2-window)
-> *view*: class gxl-spreadsheet-view

Argument	Description
sheet	The spreadsheet that contains the data to be viewed.
root-spec	The root of the graphical specification which describes the spreadsheet view to be created.
window	The g2-window where the spreadsheet view is to be shown.

Return Value	Description
view	The view created by this call.

Description

This procedure creates a workspace and creates a view of the given *sheet* on the workspace, and then displays the view on the given *window*. The arguments to this function have the same meaning as in gxl-create-spreadsheet. The view is displayed at the center of the screen.

Example

The following example displays a view of the spreadsheet named spreadsheet-1, based on the root specification named root-spec-1 on the g2-window *win*:

```
view = call gxl-create-spreadsheet-view(spreadsheet-1, root-spec-1, win);
```

gxl-create-spreadsheet

Creates a spreadsheet from a specification.

Synopsis

gxl-create-spreadsheet
(*root-spec*: class gxl-root-specification, *window*: class g2-window)
-> *spreadsheet*: class gxl-spreadsheet

Argument	Description
root-spec	The root of the graphical specification which describes the spreadsheet to be created.
window	The g2-window originating this call.

Return Value	Description
spreadsheet	The spreadsheet created by this call.

Description

This procedure creates a spreadsheet based on the information contained in a graphical specification, which consists of a root specification connected to row and column controllers, which in turn are connected to cell group specifications. The specification must be well-posed: there must be at least one row controller and at least one column controller connected to the root specification, and the cell groups must fall into the rectangular grid established by the row and column controllers.

This procedure does not create a view of the spreadsheet.

Example

The following example returns a spreadsheet based on the root specification named root-spec-1:

```
spreadsheet = call gxl-create-spreadsheet(root-spec-1, win);
```

gxl-create-spreadsheet-from-collected-specification

Creates a spreadsheet from a specification in the form of an organized set of lists.

Synopsis

gxl-create-spreadsheet-from-collected-specification
(*root-spec*: class gxl-root-specification, *row-controllers*: class: item-list,
column-controllers: class: item-list, *cell-groups*: class: item-list,
window: class g2-window)
-> *spreadsheet*: class gxl-spreadsheet

Argument	Description
root-spec	The root specification containing the information from which the spreadsheet is created.
row-controllers	The ordered item-list of the row-controllers associated with the root specification.
column-controllers	The ordered item list of the column controllers associated with the root specification.
cell-groups	The ordered item list of the cell-groups associated with the root specification.
window	The g2-window originating this call.

Return Value	Description
spreadsheet	The resulting spreadsheet.

Description

This procedure returns an instance of the spreadsheet class based on the information provided by the given *root-spec*, and the item lists of *row-controllers*, *column-controllers*, and *cell-groups*.

Example

Given the item-lists of row controllers, column controllers, and cell groups for root-spec-1, the following call creates the spreadsheet.

```
sheet = call gxl-create-spreadsheet-from-collected-specification
(root-spec-1, row-item-list, col-item-list, cgroup-item-list, win);
```

gxl-create-spreadsheet-view

Creates a spreadsheet view for an existing spreadsheet based on a specification.

Synopsis

gxl-create-spreadsheet-view
(*sheet*:class gxl-spreadsheet, *root-spec*: class gxl-root-specification,
workspace: class kb-workspace, *x*: integer, *y*: integer,
window: class g2-window)
-> *view*: class gxl-spreadsheet-view

Argument	Description
sheet	The spreadsheet that contains the data to be viewed.
root-spec	The root of the graphical specification which describes the spreadsheet view to be created.
workspace	The workspace on which the view is to be placed.
x	The horizontal position, in workspace units, where the first cell of the spreadsheet view is to be drawn.
y	The vertical position, in workspace units, where the first cell of the spreadsheet view is to be drawn.
window	The g2-window originating this call.

Return Value	Description
view	The view created by this call.

Description

This procedure creates a view of the given spreadsheet using a graphical specification. The *root-spec* argument does not have to be the same specification that was used to create the spreadsheet, but the specification must have the same number of row and column controllers as the original specification, otherwise an error is signalled.

This procedure builds the view on the given *workspace*, placing the upper left-hand corner of the first cell at the given (*x*, *y*) coordinate. This procedure does not display the workspace containing the view.

Note This procedure requires that its execution be interrupted to allow other processing to occur. For more information, see "Allow Other Processing" in Chapter 21, "Procedures" in the *G2 Reference Manual*.

Example

The following example builds a view of the spreadsheet named spreadsheet-1 on workspace-1 starting at the origin, based on the root specification named root-spec-1:

```
view = call gxl-create-spreadsheet-view (spreadsheet-1, root-spec-1,
    workspace-1, 0, 0, win);
```

gxl-create-spreadsheet-view-from-collected-specification

Creates a spreadsheet view from an organized set of lists.

Synopsis

gxl-create-spreadsheet-view-from-collected-specification
(*sheet*: class gxl-spreadsheet, *root-spec*: class gxl-root-specification, *row-controllers*: class: item-list, *column-controllers*: class: item-list, *cell-groups*: class: item-list, *workspace*: class kb-workspace, *x-origin*: integer, *y-origin*: integer, *window*: class g2-window)
-> *view*: class gxl-spreadsheet-view

Argument	Description
sheet	The spreadsheet for which the view is created.
root-spec	The root-specification of the *Sheet*
row-controllers	The ordered item-list of the row-controllers associated with the root specification.
column-controllers	The ordered item list of the column controllers associated with the root specification
cell-groups	The ordered item list of the cell-groups associated with the root specification
workspace	The workspace on which the instance of the spreadsheet view is placed.
x-origin	The x origin coordinate on the workspace that is referenced when constructing the spreadsheet view.
y-origin	The y origin coordinate on the workspace that is referenced when constructing the spreadsheet view.
window	The g2-window originating this call.

Return Value	Description
view	The resulting spreadsheet view

Description

After creating an instance of the class of view named by the gxl-spreadsheet-view-class attribute of the given *root-spec*, this procedure defines and returns the view based on the information provided.

If the total number of elements in the *row-controllers*, *column-controllers*, and *cell-groups* do not check out, this procedure signals an error (gxl-bad-specification-structure).

Example

The following call creates and returns the view of spreadsheet-1 from the layout specification of root-spec-1:

```
spreadsheet-view = call gxl-create-spreadsheet-view-from-collected-specification
(spreadsheet-1, root-spec-1, row-item-list, col-item-list,
cgroup-item-list, wkspace,0, 0, win);
```

gxl-delete-spreadsheet

Deletes a spreadsheet and all its views.

Synopsis

gxl-delete-spreadsheet
(*sheet*: class gxl-spreadsheet, *window*: class g2-window)

Argument	Description
sheet	The spreadsheet that is to be deleted.
window	The g2-window originating this call.

Description

This procedure deletes a given *sheet* and all of its views. The deletion of a view has the side effect of deleting the workspace of the view if there are no remaining items on the workspace after the view is deleted. If the spreadsheet or any of its views are permanent, this call will first make them transient, and then perform the deletion.

Example

The following call deletes the spreadsheet named spreadsheet-1 and all the views of spreadsheet-1:

```
call gxl-delete-spreadsheet(spreadsheet-1, win);
```

gxl-delete-view

Deletes a view.

Synopsis

gxl-delete-view
(*view*: class gxl-spreadsheet-view, *window*: class g2-window)

Argument	Description
view	The view that is to be deleted.
window	The g2-window originating this call.

Description

This procedure deletes a given spreadsheet *view*. The deletion of a view has the side effect of deleting the workspace of the view if there are no remaining items on the workspace after the view is deleted.

If the view is permanent, this procedure will make it transient and then perform the deletion. This procedure also deletes the toolbar, pushbuttons, indicator and formula bar associated with the view.

Example

The following call will delete a spreadsheet *view-1*:

```
call gxl-delete-view(view-1, win);
```

gxl-layout-specification

Constructs the specification on a workspace from an organized set of lists.

Synopsis

gxl-layout-specification
(*root-spec*: class gxl-root-specification, *row-controllers*: class: item-list, *column-controllers*: class: item-list, *cell-groups*: class: item-list, *workspace*: class kb-workspace, *x-origin*: integer, *y-origin*: integer)

Argument	Description
root-spec	The root specification for which the layout of specification objects will be constructed on the *Workspace*.
row-controllers	The ordered item-list of the row-controllers
column-controllers	The ordered item list of the column controllers
cell-groups	The ordered item list of the cell-groups
workspace	The workspace upon which the specification layout is constructed
x-origin	The x origin coordinate on the workspace that is referenced when constructing the specification.
y-origin	The y origin coordinate on the workspace that is referenced when constructing the specification.

Description

Given the ordered item lists of column-controllers, row-controllers, and cell-groups for a particular specification, this procedure creates the specification on a workspace. Beginning with the *root-spec*, it places the root-specification object on the *Workspace* at the location specified by *XOrigin*, *YOrigin*. Then it places and connects the *column-controllers*, *row-controllers*, and *cell-groups* on the workspace.

Example

Given the item-lists of root-spec-1, the following call constructs the specification on a workspace.

```
new-spec = gxl-layout-specification (root-spec-1, row-item-list,
col-item-list, cellgroup-item-list, workspace-1, 0,0);
```

gxl-make-spreadsheet-permanent

Makes the spreadsheet and its current state persistent.

Synopsis

gxl-make-spreadsheet-permanent
(*sheet*: class gxl-spreadsheet, *window*: class g2-window)

Argument	Description
sheet	The spreadsheet that is to be made permanent.
window	The g2-window originating this call.

Description

Objects that are created programmatically in G2 are transient (non-persistent) unless they are specifically made permanent. A permanent object will survive a G2 reset, and can be saved in a KB file. For more information on permanence in G2, see the *G2 Reference Manual*.

Calling this procedure makes an instance of a gxl-spreadsheet permanent. In addition, calling this procedure preserves the current state of the spreadsheet as the initial state of the spreadsheet. The state of a spreadsheet includes the data, dimensions, and format information. The initial state of the spreadsheet becomes the state of the spreadsheet the next time G2 is started. You may call this procedure with a spreadsheet that is already permanent, as a way to set the initial state.

If the spreadsheet has views, they are not made permanent by this call.

Example

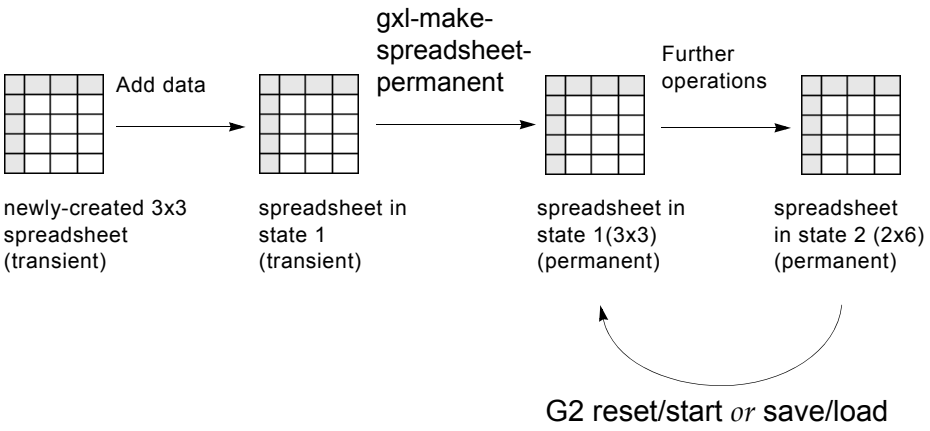
Suppose we create a spreadsheet named spreadsheet-1, that initially contains 3 rows and 3 columns, and load it with data. Call this state 1.

Now make the following call:

```
call gxl-make-spreadsheet-permanent(spreadsheet-1, win);
```

Suppose that we now perform various operations, such as editing data, adding and deleting rows and columns, sorting; so the spreadsheet is now 2 rows by 6 columns (call it state 2).

We now reset and start (or restart) G2. When G2 starts, the spreadsheet is back in state 1. This is illustrated next:



gxl-make-spreadsheet-view-permanent

Makes a view permanent, and additionally, calls gxl-make-spreadsheet-permanent on the spreadsheet associated with the view.

Synopsis

gxl-make-spreadsheet-view-permanent
(*view*: class gxl-spreadsheet, *window*: class g2-window)

Argument	Description
view	The view that is to be made permanent.
window	The g2-window originating this call.

Description

A view is a collection of G2 items anchored by an instance of a gxl-spreadsheet-view. This procedure makes a gxl-spreadsheet-view and all of its associated items permanent. In addition, calling this procedure preserves the current state of the view as the initial state of the view. The state of a view includes the editability and selectability of the cell groups and the scroll position. If you restart G2, the state of the view at the time you made the call to gxl-make-spreadsheet-view-permanent returns. You may call this procedure with a view that is already permanent, to set the initial state of the view.

The spreadsheet associated with the view is always made permanent at the same time as the view is made permanent, to assure consistency between the view and the spreadsheet.

Example

The following call makes *view-1* permanent and preserves its current state as the initial state:

```
call gxl-make-spreadsheet-view-permanent(view-1, win);
```

9

Loading and Unloading Data

Describes the API procedures for loading and unloading spreadsheet data.

Introduction

This chapter describes the procedures of the Application Programmer's Interface to GXL that you can use to programmatically move multiple data items into and out of the spreadsheet. You can programmatically add data to areas of the spreadsheet, or copy data from an area of the spreadsheet into a data structure you provide.

The following procedures load data from a list, array, or matrix into a spreadsheet:

- gxl-load-data-into-cell-group loads data into a particular cell group.
- gxl-load-data-into-defined-area loads data into any rectangular area of the spreadsheet.

To load data from a file into a spreadsheet:

1. Use the GFR procedure, gfr-load-file-into-list, which is documented in the *G2 Foundation Resources User's Guide*.
2. Load the resulting list into a spreadsheet, using either gxl-load-data-into-cell-group or gxl-load-data-into-defined-area.

The following procedures copy data from a spreadsheet to a list, array, or matrix:

- gxl-unload-data-from-cell-group copies data from a particular cell group.
- gxl-unload-data-from-defined-area copies data from any rectangular area of the spreadsheet.

You can save spreadsheet data to a file, using gxl-save-spreadsheet-area-to-stream.

See The Application Programmer's Interface for related information.

gxl-load-data-into-cell-group

Loads data from a value-array, value-list, or matrix into a specific cell group of a spreadsheet.

Synopsis

gxl-load-data-into-cell-group
(*sheet*:class gxl-spreadsheet, *group-number*: integer,
values: class object, *protocol*: symbol, *wrapping-data*: item-or-value,
update-views: truth-value, *window*: class g2-window)

Argument	Description
sheet	The spreadsheet which is the target of this call.
group-number	The reference number of the cell group that will be loaded with data.
values	A matrix, array or list containing the values that will be loaded into the cell group. Must be one of the following types (or a subclass thereof): item-array, item-list, value-array or value-list.
protocol	A flag to indicate how the data should be loaded, which can be one of the following symbols: rowwise, columnwise, rowwise-wrapping, or columnwise-wrapping.
wrapping-data	Additional data indicating how the data should be wrapped if *Protocol* is rowwise-wrapping or columnwise-wrapping. This argument can be an integer-array or integer-list (corresponding to whether *Values* is an array or list), or the symbol default.
update-views	A flag indicating if the views should be redrawn to reflect the data in the spreadsheet after the new data is loaded.
window	The g2-window originating this call.

Description

This procedure allows you to load data from a matrix, array, or list into a spreadsheet, with various options controlling how the data is placed into the cell group. The protocols used for loading data in this procedure are the same as in gxl-load-data-into-defined-area on . When you call this procedure, the effect is the same as if you called gxl-load-data-into-defined-area with the starting coordinates of the cell group and the dimensions of the cell group as the *first-row*, *first-col*, *n-rows*, and *n-cols* arguments to gxl-load-data-into-defined-area, with the following difference.

Calling gxl-load-data-into-cell-group establishes an internal relationship between the data source object *values* and the cell group. This relationship is used by gxl-apply-tabular-edit to automatically conclude data values from the spreadsheet into the data source object when gxl-apply-tabular-edit is called. However, calling gxl-load-data-into-defined-area directly does not set up a relationship between the source data object and a cell group.

For a description of the arguments *protocol*, *wrapping-data*, and *update-views*, see gxl-load-data-into-defined-area on .

This procedure does not cause cell callbacks, if any, to be called.

Example

This example shows how gxl-load-data-into-cell-group can be used in conjunction with gxl-apply-tabular-edit to load from a source object and then automatically conclude values back into the source object. For more examples on the use of loading protocols, see gxl-load-data-into-defined-area on .

Suppose that integer-array-1 = (0, 1, 2, 3, 4, 5, 6, 7, 8). The following calls create a spreadsheet in which the cell group 3 is an empty 3x3 area of integer cells, and then loads the data area with the contents of integer-array-1:

```
spreadsheet-1, view = call gxl-create-and-display-simple-spreadsheet
(the symbol integer-cell, 3, 3, 3, 3, 80, 28, the symbol small, win);
call gxl-load-data-into-cell-group(spreadsheet-1, 3, integer-array-1,
the symbol rowwise-wrapping, the symbol default, true, win);
```

The rowwise-wrapping flag causes the data to be loaded so that the first row of cell group 3 is (0, 1, 2), the second row is (3, 4, 5), and the third row is (6, 7, 8).

Now suppose that we manually edit the data in the spreadsheet, changing the values (0, 1, 2) to (10, 11, 12). The following call will update integer-array-1 so that integer-array-1 = (10, 11, 12, 3, 4, 5, 6, 7, 8):

```
call gxl-apply-tabular-edit(spreadsheet-1, win);
```

gxl-load-data-into-defined-area

Loads data from a value-array, value-list, or matrix into a specified area of a spreadsheet.

Synopsis

gxl-load-data-into-defined-area
(*sheet*:class gxl-spreadsheet, *first-row*: integer, *first-col*: integer, *n-rows*: integer, *n-cols*: integer, *values*: class object, *protocol*: symbol, *wrappingdata*: item-or-value, *update-views*: truth-value, *window*: class g2-window)

Argument	Description
sheet	The spreadsheet which is the target of this call.
first-row	The first row into which data is to be loaded.
first-col	The first column into which data is to be loaded.
n-rows	The number of rows in the target area.
n-cols	The number of columns in the target area.
values	A matrix, array or list containing the values that will be loaded into the cell group. Must be one of the following types (or a subclass thereof): item-array, item-list, value-array or value-list.
protocol	A flag to indicate how the data should be loaded, which can be one of the following symbols: rowwise, columnwise, rowwise-wrapping, or columnwise-wrapping.
wrapping-data	Additional data indicating how the data should be wrapped if *Protocol* is rowwise-wrapping or columnwise-wrapping. This argument can be an integer-array or integer-list (corresponding to whether *Values* is an array or list), or the symbol default.

Argument	Description
update-views	A flag indicating if the views should be redrawn to reflect the data in the spreadsheet after the new data is loaded.
window	The g2-window originating this call.

Description

This procedure allows you to load data into a rectangular area of a spreadsheet beginning at cell (*first-row*, *first-col*), and containing up to *n-row*s and *n-cols*. The data object *values* can be a vector of values (a value-array, value-list, or any subclass of value-array or value-list), or a matrix of values (an item-array or item-list containing value-arrays or value-lists as its elements). Gxl-load-data-into-defined-area treats arrays and lists identically, but treats vectors (one-dimensional data) differently from matrices (two-dimensional data).

If the target area defined in the call overruns either the row or column dimension of the spreadsheet, then the target area is automatically reduced to fit the spreadsheet. Data values outside of the target area are never changed, even if the dimensions of the source data object are larger than the target area. This procedure will, if necessary, ignore parts of the source data to avoid overrunning the target area.

If the data provided to this procedure does not fill the target area, the cells in the target area that were not assigned new values are cleared by this procedure.

Update-views indicates whether or not the views of the spreadsheet (if any) should be re-drawn after this procedure is complete. Generally, this argument should be true, unless you are doing several consecutive calls to this procedure (or another procedure which assigns values or formats to cells), in which case the last call to this procedure should be followed by a call to gxl-refresh-all-views.

Loading One-Dimensional (Vector) Data

When loading from a value-array or value-list, the *protocol* argument has four options: rowwise, columnwise, rowwise-wrapping and columnwise-wrapping:

- If *protocol* is rowwise, then the values in the vector are placed into the first row of the target area.
- If the *protocol* is columnwise, then the values in the vector are placed into the first column of the target area.
- If the *protocol* is rowwise-wrapping, the result depends on the *wrapping-data* argument as follows:

  - If *wrapping-data* is the symbol default, then values in the vector are loaded into the first row of the target area, with any leftover values placed in the second row of the target area, and further leftover values placed in the third row of the target area, continuing until the target area is full. If there are unused values in the data vector after the target area is full, the data are ignored.
  - If *wrapping-data* is an integer-list or an integer-array, the integers are used to indicate the number of values from the vector to be placed in each row of the target area. If there are more elements in the *wrapping-data* than rows in the target area, or if the data vector is not long enough to fill a row as specified by the *WrappingData*, the remaining elements of *wrapping-data* are ignored.
- If the protocol is columnwise-wrapping, the result depends on the *wrapping-data* argument as follows:
  - If *wrapping-data* is the symbol default, then values in the vector are loaded into the first column of the target area, with any leftover values placed in the second column of the target area, and further leftover values placed in the third column of the target area, continuing until the target area is full. If there are unused values in the data vector after the target area is full, further data are ignored.
  - If *wrapping-data* is an integer-list or an integer-array, the integers are used to indicate the number of values from the vector to be placed in each column of the target area. If there are more elements in the *wrapping-data* than columns in the target area, or if the data vector is not long enough to fill a column as specified by the *wrapping-data*, the remaining elements of *wrapping-data* are ignored.

Loading Two-Dimensional (Matrix) Data

When loading from a matrix, the *Protocol* argument has two options: rowwise and columnwise:

- If *protocol* is rowwise, the values from the first vector (value-array or value-list) in the matrix are placed into the first row of the target area, the values from the second vector in the matrix are placed into the second row of the target area, etc. If there are more items in the matrix than rows in the target area, the data is ignored.
- If *protocol* is columnwise, the values from the first vector (value-array or value-list) in the matrix are placed into the first column of the target area, the values from the second vector in the matrix are placed into the second column of the target area, etc. If there are more items in the matrix than columns in the target area, the data is ignored.

Examples

Suppose *vector-1* = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), *wrapping-data* = (4, 2, 4) and *matrix-1* = (0, 1, 2, 3; 4, 5; 6, 7, 8, 9), where the semi-colon indicates the end of each vector contained in the matrix (e.g., matrix-1 is an item-array or item-list containing 3 integer-arrays or integer-lists). Consider an area of a spreadsheet whose dimensions are 2 rows by 4 columns, starting at 0, 0. Here are the results of several calls to gxl-load-data-into-defined-area:

Example 1

call gxl-load-data-into-defined-area (spreadsheet-1, 0, 0, 2, 4, vector-1, the symbol rowwise, the symbol default,true, win);

0	1	2	3

Example 2

call gxl-load-data-into-defined-area (spreadsheet-1, 0, 0, 2, 4, vector-1, the symbol columnwise, the symbol default, true, win);

0			
1			

Example 3

call gxl-load-data-into-defined-area (spreadsheet-1, 0, 0, 2, 4, vector-1, the symbol rowwise-wrapping, the symbol default, true, win);

0	1	2	3
4	5	6	7

Example 4

call gxl-load-data-into-defined-area (spreadsheet-1, 0, 0, 2, 4, vector-1, the symbol columnwise-wrapping, the symbol default, true, win);

0	2	4	6
1	3	5	7

Example 5

call gxl-load-data-into-defined-area (spreadsheet-1, 0, 0, 2, 4, matrix-1, the symbol rowwise, the symbol default, true, win);

0	1	2	3
4	5		

Example 6

call gxl-load-data-into-defined-area (spreadsheet-1, 0, 0, 2, 4, matrix-1, the symbol columnwise, the symbol default, true, win);

0	4	6	
1	5	7	

Example 7

call gxl-load-data-into-defined-area (spreadsheet-1, 0, 0, 2, 4, vector-1, the symbol rowwise-wrapping, wrapping-data, true, win);

0	1	2	3
4	5		

Example 8

call gxl-load-data-into-defined-area (spreadsheet-1, 0, 0, 2, 4, vector-1, the symbol columnwise-wrapping, wrapping-data, true, win);

0	4	6	
1	5	7	

Note that in examples 7 and 8, the result of loading *vector-1* using the wrapping data is equivalent to the result of using *matrix-1*. In other words, the vector (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) with the wrapping data (4, 2, 4) is essentially the same as the matrix (0, 1, 2, 3; 4, 5; 6, 7, 8, 9).

gxl-unload-data-from-cell-group

Copies data in a spreadsheet into a value-array, value-list, or matrix provided by the user.

Synopsis

gxl-unload-data-from-cell-group
(*sheet:* class gxl-spreadsheet, *group-number*: integer,
values: class object, *protocol*: symbol, *wrapping-data*: item-or-value,
window: class g2-window)

Argument	Description
sheet	The spreadsheet which is the data source.
group-number	The reference number of the cell group whose data will be copied.
values	A matrix, array or list that will receive the data. Must be one of the following types (or a subclass thereof): item-array, item-list, value-array or value-list.
protocol	A flag to indicate how the data should be unloaded, which is one of the following symbols: rowwise, columnwise, rowwise-wrapping, or columnwise-wrapping.
wrapping-data	An optional integer-array or integer-list, (corresponding to whether *values* is an array or list), which on output will contain the number of values per row or column, if *values* is not a matrix. This argument may only be used when the *protocol* is rowwise-wrapping or columnwise-wrapping. Otherwise, this argument should be the symbol default.
window	The g2-window originating this call.

Description

This procedure allows you to unload data from a cell group into another data structure, which may be a matrix (an item-array or item-list whose elements are value-arrays or value-lists), or a vector (a value-array or value-list). The argument *protocol* controls how the data is placed into the target data structure. The

protocols used for unloading data in this procedure are the same as in gxl-unload-data-from-defined-area.

When you call this procedure, the effect is the same as if you called gxl-unload-data-from-defined-area with the starting coordinates of the cell group and the dimensions of the cell group as the *first-row*, *first-col*, *n-rows*, and *n-cols* arguments to gxl-unload-data-from-defined-area.

Example

See gxl-unload-data-from-defined-area.

gxl-unload-data-from-defined-area

Copies data from a spreadsheet area into a value-array, value-list, or matrix provided by the user.

Synopsis

gxl-unload-data-from-defined-area
(*sheet:* class gxl-spreadsheet, *first-row*: integer, *first-col*: integer, *n-rows*: integer, *n-cols*: integer, *values*: class object, *protocol*: symbol, *wrapping-data*: item-or-value, *window*: class g2-window)

Argument	Description
sheet	The spreadsheet which is the data source.
first-row	The first row from which data is to be copied.
first-col	The first column from which data is to be copied.
n-rows	The number of rows in the target area of the spreadsheet.
n-cols	The number of columns in the target area of the spreadsheet.
values	A matrix, array or list that will receive the data. Must be one of the following types (or a subclass thereof): item-array, item-list, value-array or value-list.
protocol	A flag to indicate how the data should be unloaded, which is one of the following symbols: rowwise, columnwise, rowwise-wrapping, or columnwise-wrapping.
wrapping-data	An optional integer-array or integer-list (corresponding to whether *Values* is an array or list), which on output will contain the number of values per row or column, if *Values* is not a matrix. This argument may only be used when the *Protocol* is rowwise-wrapping or columnwise-wrapping. Otherwise, this argument should be the symbol default.
window	The g2-window originating this call.

Description

This procedure is closely related to gxl-load-data-into-defined-area. If you load data into a spreadsheet using gxl-load-data-into-defined-area (or into a cell group using gxl-load-data-into-cell-group) using a given *protocol* and *wrapping-data*, and then use this procedure to unload data using the same *protocol* and *wrapping-data*, the format of your data will be unchanged, although containing updates to the data made in the spreadsheet.

This procedure copies the data contained in a rectangular area of a spreadsheet beginning at cell (*first-row*, *first-col*), and containing up to *n-row*s and *n-cols*. If the area of the spreadsheet defined in the call overruns the dimension of the spreadsheet, then *n-row*s and *n-cols* are automatically reduced to fit the spreadsheet.

The data object *values* can be a vector of values (a value-array, value-list, or any subclass of value-array or value-list), or a matrix of values (an item-array or item-list containing value-arrays or value-lists or a subclass of these types). Gxl-unload-data-from-defined-area treats arrays and lists identically, but treats vectors (one-dimensional data) differently from matrices (two-dimensional data).

Cells that contain no values (empty cells) are handled according to the following rule:

- Empty cells that would be at the end of the data structure are ignored.
- Embedded empty cells are filled with a default value appropriate to the data structure that is receiving the data.

The G2 defaults for the seven value types are:

Type	Default value
integer	0
float	0.0
quantity	0.0
text	"" (the empty string)
truth-value	false
symbol	G2
value	0.0

For example, if you are unloading a row of cells that contain the values (0, 1, <empty>, 3, 4, 5, <empty>, <empty>) into an integer array, on output your integer array will be the following 6-element vector: (0, 1, 0, 3, 4, 5), where G2's default value for integer arrays, 0, replaces the embedded empty cell.

Unloading One-Dimensional (Vector) Data

When unloading data into a vector (a value-array or value-list), the *Protocol* argument has four options: rowwise, columnwise, rowwise-wrapping and columnwise-wrapping:

- If *Protocol* is rowwise, then the values from the first row of the target area are copied into the vector.
- If the *protocol* is columnwise, then the values from the first column of the target area are copied into the vector.
- If the *protocol* is rowwise-wrapping, the result depends on the *wrapping-data* argument, as follows:
    - If *wrapping-data* is the symbol default, then values from the target area of the spreadsheet are loaded row by row into the vector, with the values in the first row followed directly by the values in the second row, etc. Empty cells in all rows up to the last data value in the last row containing data are replaced by default values. Empty cells beyond the last data value in the last row containing data within the target area are ignored.
    - If *wrapping-data* is an integer-list or an integer-array, then on output *wrapping-data* represents the number of values in each row of data in the target area. The values from the target area are copied row by row from the target area. Trailing empty cells in each row are ignored rather than replaced by default values. Empty rows beyond the last row containing data are ignored; therefore the number of elements in *wrapping-data* may be less than the number of rows in the target area.
- If the protocol is columnwise-wrapping, the result depends on the *wrapping-data* argument as follows:
    - If *wrapping-data* is the symbol default, then values from the target area of the spreadsheet are loaded column by column into the vector, with the values in the first column followed directly by the values in the second column, etc. Empty cells in all columns up to the last data value in the last column containing data are replaced by default values. Empty cells beyond the last data value in the last column containing data within the target area are ignored.
    - If *wrapping-data* is an integer-list or an integer-array, then on output *wrapping-data* represents the number of values in each column of data in the target area. The values from the target area are copied column by column from the target area. Trailing empty cells in each column are ignored rather than replaced by default values. Empty columns beyond the last column containing data are ignored; therefore the number of elements in *wrapping-data* may be less than the number of columns in the target area.

Unloading Two-Dimensional (Matrix) Data

A matrix consists of an item-array or item-list, or "spine," containing vectors which constitute the "ribs" of the matrix. When you unload a spreadsheet area into a matrix, the matrix may have to be resized. Special considerations arise when the dimension of the spine of the matrix needs to be changed.

When the length of the spine is reduced, garbage (items in memory that cannot be accessed by the KB, except by global searches over all items of a given class) can be created by orphaning the vectors that were contained in spine before it was shortened. If you want GXL to delete orphaned vectors, the attribute gxl-cleanup-when-resizing-matrices of the spreadsheet should be set to true (the default). If you do not want these vectors to be deleted by GXL, set this attribute to false.

When the length of the spine needs to be increased, new "ribs" need to be created. However, GXL does not have the information necessary to decide what class of vector should be created. In general, you need to provide a procedure that can be called by GXL that adds new "ribs" to the matrix. If this procedure is not provided, GXL will use the most specific type of array (not list) that accommodates the data contained in the corresponding row or column of the spreadsheet.

To provide your own matrix extension procedure, you set the attribute gxl-matrix-extension-procedure of the spreadsheet to the name of your procedure. To use the built-in default method for extending matrices, set this attribute to the symbol unspecified.

The signature of the matrix extension procedure you provide is as follows:

> your-procedure-name (*matrix*: class object, *position*: integer, *value-type*: symbol)

where:

Argument	Description
matrix	Either an item-list or item-array, and is the same object as the *Values* argument passed to gxl-unload-data-from-defined-area or gxl-unload-data-from-cell-group.

Argument	Description
position	An integer which indicates the element number where a new vector is to be added to *Matrix*.
value-type	A symbol which represents the type of values that the new vector must hold, based on the contents of the spreadsheet. Valid symbols are: integer, float, quantity, symbol, text, truth-value, or value

Note You can provide a more general type of vector than required by *value-type*, for example, a quantity list if *value-type* is float. However, providing a more specific type than required generates an error.

Your matrix extension procedure must add the vector to the spine of the matrix.

When loading a matrix, the *Protocol* argument has two options: rowwise and columnwise:

- If *protocol* is rowwise, the values from the first row of the target area are copied into the first vector (value-array or value-list) in the matrix, the values from the second row of the target area are placed into the second vector of the matrix, etc. If there are empty cells at the end of any row, they are not included in the values copied into the corresponding vector.

  Thus, the length of a vector in the matrix might be less than the number of columns in the target area. Embedded empty cells are represented by default values. If there are empty rows after the last row containing data, these rows are not represented in the matrix. Therefore, on output, the matrix spine might be shorter than the number of rows in the target area.

- If *protocol* is columnwise, the values from the first column of the target area are copied into the first vector (value-array or value-list) in the matrix, the values from the second column of the target area are placed into the second vector of the matrix, etc. If there are empty cells at the end of any column, they are not included in the values copied into the corresponding vector.

  Thus, the length of a vector in the matrix might be less than the number of rows in the target area. Embedded empty cells are represented by default values. If there are empty columns after the last column containing data, these columns are not represented in the matrix. Therefore, on output, the matrix spine might be shorter than the number of columns in the target area.

Examples

Suppose *vector-1* is an integer array, *wrapping-data* is an integer list, and *matrix-1* is an item array. Here are the results of several calls to gxl-unload-data-from-defined-area, where the target area is three rows by four columns, starting at row 0, column 0, if the data in the area is as follows:

0	1	2	3
4	5		

Example 1

call gxl-unload-data-from-defined-area (spreadsheet-1, 0, 0, 3, 4, vector-1, the symbol rowwise, the symbol default, win);

vector-1 = (0, 1, 2, 3)

Example 2

call gxl-unload-data-from-defined-area (spreadsheet-1, 0, 0, 3, 4, vector-1, the symbol columnwise, the symbol default, win);

vector-1 = (0, 4)

Example 3

call gxl-unload-data-from-defined-area(spreadsheet-1, 0, 0, 3, 4, vector-1, the symbol rowwise-wrapping, the symbol default, win);

vector-1 = (0, 1, 2, 3, 4, 5)

Example 4

call gxl-unload-data-from-defined-area(spreadsheet-1, 0, 0, 3, 4, vector-1, the symbol columnwise-wrapping, the symbol default, win);

vector-1 = (0, 4, 0, 1, 5, 0, 2, 0, 0, 3)

Example 5

call gxl-unload-data-from-defined-area(spreadsheet-1, 0, 0, 3, 4, matrix-1, the symbol rowwise, the symbol default, win);

matrix-1 = (0, 1, 2, 3; 4 5)

(The semicolon represents the end of each vector contained in the matrix. In this case, the length of the spine of *matrix-1* is 2, its first vector is length 4, and its second vector is length 2.)

Example 6

call gxl-unload-data-from-defined-area(spreadsheet-1, 0, 0, 3, 4, matrix-1, the symbol columnwise, the symbol default, win);

matrix-1 = (0, 4; 1, 5; 2; 3)

Example 7

call gxl-unload-data-from-defined-area(spreadsheet-1, 0, 0, 3, 4, vector-1, the symbol rowwise-wrapping, wrapping-data, win);

vector-1 = (0, 1, 2, 3, 4, 5)

wrapping-data = (4, 2)

Example 8

call gxl-unload-data-from-defined-area(spreadsheet-1, 0, 0, 3, 4, vector-1, the symbol columnwise-wrapping, wrapping-data, win);

vector-1 = (0, 4, 1, 5, 2, 3)

wrapping-data = (2, 2, 1, 1)

Note the similarities between Examples 5 and 6 and Examples 7 and 8.

gxl-save-spreadsheet-area-to-stream

Writes data in a specific rectangular area of a spreadsheet to a file.

Synopsis

gxl-save-spreadsheet-area-to-stream
(*sheet:* class gxl-spreadsheet, *stream*: class g2-stream, *separator*: text, *first-row*: integer, *first-col*: integer, *n-rows*: integer, *n-cols*: integer, *progress*: class integer-parameter, *window*: class g2-window)

Argument	Description
sheet	The spreadsheet which is the data source.
stream	A g2-stream that was generated by calling g2-open-file-for-write or g2-open-file-for-read-and-write.
separator	The text used to separate values on a line in the file
first-row	The first row from which data is to be copied.
first-col	The first column from which data is to be copied.
n-rows	The number of rows in the target area of the spreadsheet.
n-cols	The number of columns in the target area of the spreadsheet.
progress	An integer parameter that is incremented as the save proceeds, which can also be used to interrupt this procedure.
window	The g2-window originating this call.

Description

This procedure allows you write data from a spreadsheet to a file. The *first-row*, *first-col*, *n-rows*, and *n-cols* arguments define the rectangular area of the spreadsheet that is saved by this call. The file that is written must be opened prior to this call, and is provided as a g2-stream object. For more information on streams, see the *G2 System Procedures Reference Manual*. This procedure always

saves the data rowwise, and puts a line feed/carriage return at the end of each row of the spreadsheet's data that is saved.

The *separator* is a text character, usually a comma, that is written between each value that is written on a line in the file. The separator is not written at the beginning or end of a line. To represent empty cells, consecutive separators are written with nothing between them.

The printing of text strings follows these rules:

- Texts are enclosed in beginning and ending quotation marks.
- Embedded quotes within text strings are doubled, for example, the string abc "def" ghi is printed as "abc ""def"" ghi".

This handling of text values and embedded quotes is compatible with Microsoft Excel's comma-separated value (csv) file format.

Because saving a large spreadsheet may take some time, this procedure periodically allows other processing. Just before allowing other processing, the *progress* parameter is updated to indicate the percentage complete of the save. *Progress* has a final value of 100 when the save is finished. If you start (rather than call) this procedure, you can set up a monitor that receives control when *progress* is updated, by using a wait until *progress* receives a value statement.

The other use of the *progress* parameter is to abort a save in progress. If you delete *progress* before the completion of the save, the save is aborted

When you save a spreadsheet area containing a column header, the column header is written to the file once for each column it spans. For example, if the column header spans three columns, then the output file will have the text of the column header written three times consecutively on the line corresponding to the row of the spreadsheet containing the header.

The Gensym Foundation Resources (GFR) module provides the utilities needed to read comma-separated text files. For example, the gfr-load-file-into-list procedure returns file data in a form loadable into a spreadsheet using gxl-load-data-into-cell-group or gxl-load-data-into-defined-area. For information on gfr-load-file-into-list, see the *G2 Foundation Resources User's Guide*.

Example

Suppose the following data is stored in a spreadsheet *sheet-1,* where upper-case represents symbols:

true	false		1.0e32
abc	ABC	Bob says “hi”	
1	2		
		1.234	4

If *progress* is an integer-parameter and *stream-1* is a g2-stream that has been opened for writing, then we can make the following call:

```
call gxl-save-spreadsheet-area-to-stream (sheet-1, stream-1, “,”,
0, 0, 5, 4, Progress, win);
```

Here is the resulting file:

```
true, false, , 9.9999999999999987e31

“abc”, ABC, “Bob says ““hi””””,

1, 2, ,

, , 1.234, 4

, , ,
```

This file can be imported into Microsoft Excel as a comma-separated value file.

10

Accessing Spreadsheet and View Properties

Describes the API procedures for accessing GXL spreadsheet and view properties.

Introduction

This chapter describes the procedures of the Application Programmer's Interface to GXL that you can use to programmatically access the properties of spreadsheets and views. See The Application Programmer's Interface for related information.

The following table summarizes the information about spreadsheets that you can obtain programmatically:

To get this spreadsheet property...	Use this procedure...
Color pattern of a cell	gxl-get-cell-color
Current value stored in a cell	gxl-get-cell-contents
Group number of a cell	gxl-get-group-number-at-coordinates
Size of a cell group	gxl-get-cell-group-dimensions
Location of a cell group	gxl-get-cell-group-coordinates
Initialization data of a cell group	gxl-get-cell-group-initialization-data
Layout of cell groups	gxl-get-cell-group-layout
Initialization, validation or callback procedures associated with a cell group	gxl-get-cell-group-procedure-attribute

To get this spreadsheet property...	Use this procedure...
Type of cell associated with a cell group	gxl-get-cell-type-of-group
Dimensions of the spreadsheet	gxl-get-cell-type-of-group
Specification used to create the spreadsheet	gxl-get-specification-of-spreadsheet

The following table summarizes the information about views that you can obtain programmatically:

To get this view property...	Use this procedure...
Visible row-column dimensions of a cell group	gxl-get-cell-group-visible-dimensions
Float format specification of a cell group	gxl-get-float-format-of-group-on-view
Selectability and editability of a cell group	gxl-get-protection-of-group-on-view
Font, height, and width size attributes of a cell group	gxl-get-size-attributes-of-cells-in-view
Selected columns	gxl-get-selected-column-range
Selected rows	gxl-get-selected-row-range
Selected rectangular area	gxl-get-selection-limits
Workspace location of a particular cell for a given view	gxl-get-workspace-location-of-cell
Workspace location of the cell group for a given view	gxl-get-workspace-location-of-cell-group

You can navigate between spreadsheets and their views, using gxl-get-views-of-spreadsheet or gxl-get-spreadsheet-of-view.

When you need to set an attribute of a specification object, you can use gxl-get-specification-object, which retrieves the specification object, based on its position in the specification layout.

GXL software version information is available through gxl-get-version.

See The Application Programmer's Interface for related information.

gxl-get-cell-color

Returns the color pattern of a cell.

Synopsis

gxl-get-cell-color
(*sheet*: class gxl-spreadsheet, *row*: integer, *column*: integer,
window: class g2-window)
-> *background-color*: symbol, *text-color*: symbol, *border-color*: symbol

Argument	Description
sheet	The spreadsheet that is the subject of this call.
row	The row coordinate of the cell whose color pattern is to be returned.
column	The column coordinate of the cell whose color pattern is to be returned.
window	The g2-window originating this call.

Return Value	Description
background-color	The background color of the cell.
text-color	The text color of the cell.
border-color	The border color of the cell.

Description

Each cell in the spreadsheet has a color pattern described by its background, text and border colors. Given a *row*, *column* coordinate and a *sheet*, this procedure returns these colors. If *row* or *column* are out of bounds, this procedure signals an error (gxl-row-or-column-out-of-bounds).

Example

The following call retrieves the color pattern of the cell located at row 0, column 0 in spreadsheet-1:

```
background-color, text-color, border-color = call gxl-get-color-pattern-of-cell
    (spreadsheet-1, 0, 0, win);
```

gxl-get-cell-contents

Returns the value stored in a cell.

Synopsis

gxl-get-cell-contents
(*sheet*: class gxl-spreadsheet, *row*: integer, *column*: integer,
window: class g2-window)
-> *cell-contents*: value

Argument	Description
sheet	The spreadsheet that is the subject of this call.
row	The row coordinate of the cell whose value is to be returned.
column	The column coordinate of the cell whose value is to be returned.
window	The g2-window originating this call.

Return Value	Description
cell-contents	The value stored at the given row/column location of the spreadsheet.

Description

Given a *row*, *column* coordinate and a *sheet*, this procedure returns the value stored in the cell. If the cell is empty, the text string "!nv" is returned. If *row* or *column* are out of bounds, this procedure signals an error (gxl-row-or-column-out-of-bounds).

Example

The following code segment calls gxl-get-cell-contents to retrieve the value of the cell located at row 0, column 0 in spreadsheet-1, and informs the operator if it is empty:

```
cell-value = call gxl-get-cell-contents(spreadsheet-1, 0, 0, win);
if cell-value = "!nv" then inform the operator that "Cell (0,0) is empty";
```

gxl-get-cell-group-coordinates

Returns the starting and ending row/column coordinates of a cell group.

Synopsis

gxl-get-cell-group-coordinates
(*sheet*: class gxl-spreadsheet, *group-number*: integer,
window: class g2-window)
-> *first-row*: integer, *first-col*: integer, *last-row*: integer,
last-col: integer

Argument	Description
sheet	The spreadsheet that is the subject of this call.
group-number	The reference number of the cell group whose coordinates are to be determined by this call.
window	The g2-window originating this call.

Return Value	Description
first-row	The first row of the cell group.
first-col	The first column of the cell group.
last-row	The last row of the cell group.
last-col	The last column of the cell group.

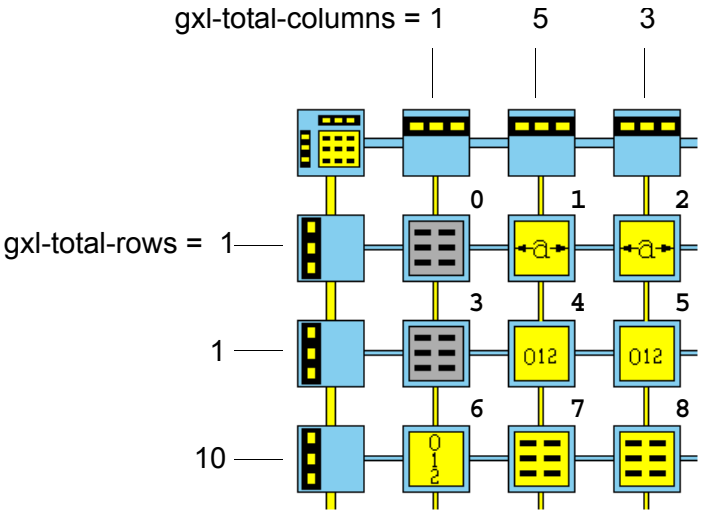
Description

Each spreadsheet is made of a number of rectangular cell groups. Each of these groups is referred to by a number that identifies its position in the spreadsheet specification.

This procedure returns the coordinates of the first cell (the cell in the upper left-hand corner) in a given cell group, and the coordinates of the last cell in the group (the cell in the lower right-hand corner). If an invalid group number is given, this procedure signals an error (gxl-group-number-out-of-bounds).

Example

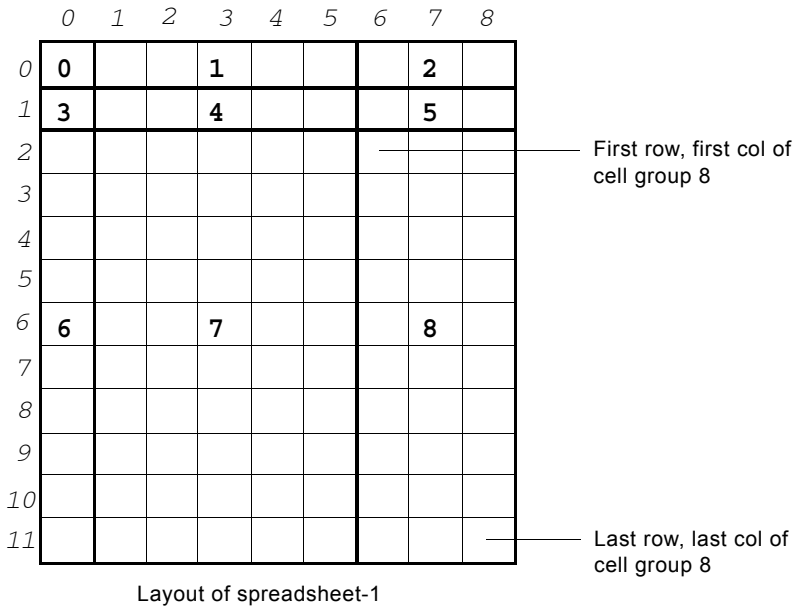
The following spreadsheet specification is used to create spreadsheet-1:



Specification for spreadsheet-1

The cell groups in the specification are numbered starting from 0, left-to-right in consecutive rows of the specification.

The following figure illustrates the corresponding spreadsheet layout of spreadsheet-1:



Layout of spreadsheet-1

Note that the row and column numbers are zero-based; therefore, even though this spreadsheet has 12 rows, the last row number is 11.

The following call determines the coordinates of cell group 8:

```
first-row, first-col, last-row, last-column = call gxl-get-cell-group-coordinates
    (spreadsheet-1, 8, win);
```

The return values are first-row = 2, first-col = 6, last-row = 11, last-column = 8.

gxl-get-cell-group-dimensions

Returns the number of rows and columns in a cell group.

Synopsis

gxl-get-cell-group-dimensions
(*sheet*: class gxl-spreadsheet, *group-number*: integer,
window: class g2-window)
-> *row-count*: integer, *col-count*: integer

Argument	Description
sheet	The spreadsheet that is the subject of this call.
group-number	The reference number of the cell group whose dimensions are to be determined by this call.
window	The g2-window originating this call.

Return Value	Description
row-count	The current number of rows in the cell group.
col-count	The current number of columns in the cell group.

Description

This procedure returns the dimensions of a cell group, which is specified by its *group-number*. If an invalid group number is given, this procedure signals an error (gxl-group-number-out-of-bounds).

Example

Using the same example as shown in gxl-get-cell-group-coordinates, the following call yields the dimensions of cell group 8:

```
nrows, ncols = call gxl-get-cell-group-dimensions(spreadsheet-1, 8, win);
```

The result is nrows = 10, ncols = 3.

gxl-get-cell-group-initialization-data

Returns the initialization data of a cell group.

Synopsis

gxl-get-cell-group-initialization-data
(*sheet*: class gxl-spreadsheet, *group-number*: Integer,
window: class g2-window)
-> *initialization-data*: value

Argument	Description
sheet	The spreadsheet that is the target of this call.
group-number	The reference number of the cell group whose initialization data is returned.
window	The g2-window originating this call.

Return Value	Description
initialization-data	The initialization data associated with the given cell group of the *Sheet*.

Description

For a given cell group of a spreadsheet, this procedure returns the *initialization-data* associated with that cell group. If the given *GroupNumber* is out of bounds, this procedure signals an error (gxl-group-number-out-of-bounds).

Example

The following call returns the initialization data of column-header (1) for spreadsheet-1.

```
InitData = call gxl-get-cell-group-initialization-data (spreadsheet-1, 1, win);
```

gxl-get-cell-group-layout

Returns the number of cell groups in the vertical and horizontal directions

Synopsis

gxl-get-cell-group-layout
(*sheet*: class gxl-spreadsheet, *window*: class g2-window)
-> *vertical-cell-group*: integer, *horizontal-cell-group*: integer

Argument	Description
sheet	The spreadsheet that is the target of this call.
window	The g2-window originating this call.

Return Value	Description
vertical-cell-group	The number of items vertically arranged on the layout associated with the *Sheet*.
horizontal-cell-group	The number of items horizontally arranged on the layout associated with the *Sheet*.

Description

This procedure returns the number of cell groups, as they are arranged vertically and horizontally, for a given *sheet*.

Example

The following call returns the layout of cell groups for spreadsheet-1:

```
NVerticalCellGroups, NHorizontalCellGroups =call gxl-get-cell-group-layout
(spreadsheet-1, win);
```

The returned values 3, 3, define a cell group layout that has 3 vertical cell groups and 3 horizontal cell groups.

gxl-get-cell-group-procedure-attribute

Returns the name of any procedures associated with given cell group.

Synopsis

gxl-get-cell-group-procedure-attribute
(*sheet*: class gxl-spreadsheet, *group-number*: integer,
procedure-attribute: symbol, *window*: class g2-window)
-> *Procedure*: symbol

Argument	Description
sheet	The spreadsheet that is the target of this call.
group-number	The reference number of the cell group whose associated procedure is returned
procedure-attribute	The symbolic name of the attribute whose procedure, if any, is returned. Valid attributes are: gxl-initialization-procedure gxl-reinitialization-procedure gxl-additional-validation-procedure gxl-callback-procedure gxl-selection-callback-procedure
window	The g2-window originating this call.

Return Value	Description
procedure	A symbol naming a suitable initialization, validation or callback procedure associated with a given cell group of the *Sheet*, or the symbol unspecified.

Description

This procedure returns the symbolic name of the procedure, if any, for the specified *procedure-attribute* of a given cell group. Depending on the *procedure-attribute*, the procedure can be:

- A special validator for the given cell group.
- An initialization or reinitialization procedure for the given cell group.

- A callback procedure of the spreadsheet for the given cell group.
- A selection callback procedure for the given cell group.

If the given *group-number* is out of bounds, this procedure signals an error (gxl-group-number-out-of-bounds). If the given *procedure-attribute* is not a valid attribute, this procedure signals an error (gxl-bad-procedure-attribute).

Example

The following call returns the symbolic name of the procedure for the gxl-initialization-procedure attribute of the column-header cell group (5) associated with spreadsheet-1.

```
ProcedureName = call gxl-get-cell-group-procedure-attribute
(spreadsheet-1, 5, gxl-initialization-procedure, win);
```

The return value is gxl-serial-integer-initialization, which is the default setting for this category of cell group.

gxl-get-cell-group-visible-dimensions

Returns the visible row/column dimensions of a group on a particular view.

Synopsis

gxl-get-cell-group-visible-dimensions
(*view*: class gxl-spreadsheet-view, *group-number*: integer,
window: class g2-window)
-> *row*: integer, *column*: integer

Argument	Description
view	The spreadsheet view that is the target of this call.
group-number	The reference number of the cell group whose visible row/column dimensions are returned
window	The g2-window originating this call.

Return Value	Description
row	The row dimensions of a cell group on a particular *View*.
column	The column dimensions of a cell group on a particular *View*.

Description

This procedure gets the spreadsheet associated with the specified *view*, and finds the row- controller and column-controller associated with the specified cell group.

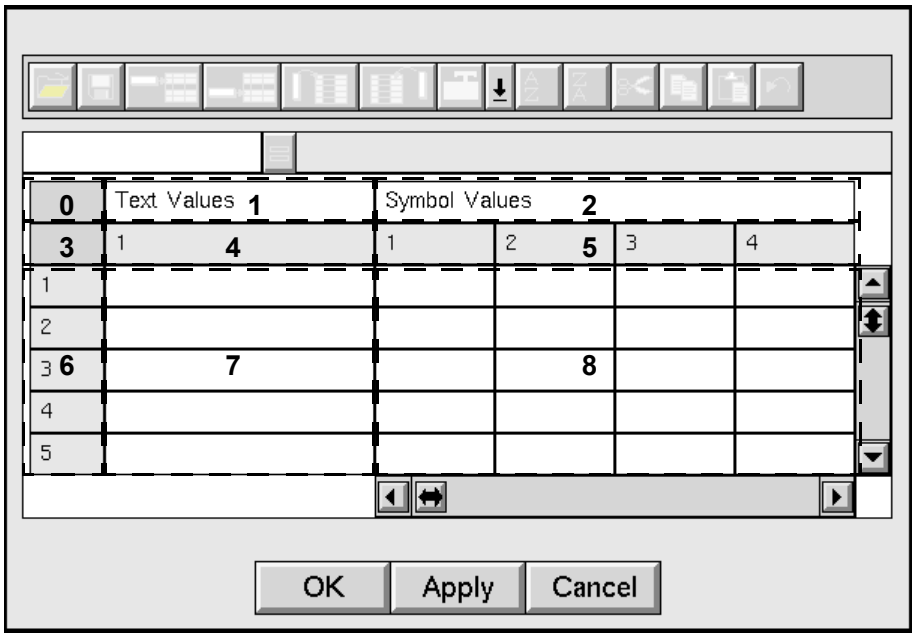
If both horizontal and vertical scrollable areas exist on the view for the given cell group, the number of visible rows and columns are returned. If horizontal and vertical scroll areas do not exist for the given cell group, the procedure returns 0 for the row, column dimensions.

If the given *group-number* is out of bounds, this procedure signals an error (gxl-group-number-out-of-bounds).

Example

Assuming that the figure displays the current view of spreadsheet-1, the following call for cell group 8 would return the row-column dimensions of its visible cells:

```
Row, Column =call gxl-get-cell-group-visible-dimensions
(spreadsheet-1-view, 8, win);
```



The returned values would be 5, 4; that is, the visible dimensions of cell group (8) are 5 rows by 4 columns.

gxl-get-cell-type-of-group

Returns the type of cell in a given cell group.

Synopsis

gxl-get-cell-type-of-group
(*sheet*: class gxl-spreadsheet, *group-number*: integer,
window: class g2-window)
-> *cell-type*: symbol

Argument	Description
sheet	The spreadsheet that is the subject of this call.
group-number	The reference number of the cell group whose cell type is to be returned.
window	The g2-window originating this call.

Return Value	Description
cell-type	The cell type of the cell group, which is one of the following symbols: integer-cell, float-cell, quantity-cell, text-cell, symbol-cell, truth-value-cell, value-cell, column-header, global-selector, disabled-cell, column-selector, or row-selector.

Description

Each cell group has a specific type of cell which dictates the type of value the cell can contain. This procedure returns the cell type. This procedure can be used to determine the value type accepted by the cell.

Example

The following call retrieves the cell type of group 3 in spreadsheet-1:

```
cell-type = call gxl-get-cell-type-of-group(spreadsheet-1, 3, win);
```

gxl-get-float-format-of-group-on-view

Returns the float format specification for a cell group on a spreadsheet view.

Synopsis

gxl-get-float-format-of-group-on-view
(*view*: class gxl-spreadsheet-view, *group-number*: integer,
window: class g2-window)
-> *use-default*: truth-value, *minimum-width*: integer,
precision: integer, *output-format*: symbol,
strip-zeros: truth-value

Argument	Description
view	The spreadsheet view that is the target of this call.
group-number	The reference number of the cell group whose float format specification is returned.
window	The g2-window originating this call.

Return Value	Description
use-default	The value true, if default formatting is used
minimum-width	Specifies the minimum number of characters in the formatted version of float. If the formatted float value has fewer characters, the text is padded on the left side with space characters to equal the given width. If the formatted value has more characters, the width expands to accommodate the text without adding any space characters.

Return Value	Description
precision	Specifies either the number of digits to the right of the decimal point or the significant digits, depending on the output-format value. When the formatted float uses the float or exponent output-format, precision indicates the digits to the right of the decimal point. When using the best output-format, precision determines the number of significant digits.
output-format	Determines the precision format. Valid formats are: • float, which displays the value in a float format. • exponent, which displays the value as an exponent. • best, which displays the value as a float or an exponent if the result is either too small or too large for the specified precision.
strip-zeros	Specifies whether zeros to the right of the last non-zero digit are stripped, where true removes trailing zeros.

Description

This procedure returns the float format of a given cell group on the specified *view*. The display format is based on the settings specified in the gxl-float-format attribute for the cell group.

The return values - *minimum-width*, *precision*, *output-format*, *strip-zeros* - are the same as in the G2 system procedure, g2-float-to-text.

Example

The following call returns the float format of cell group 4 of view-1:

```
use-default, minimum-width, precision, output-format,
strip-zeros = call gxl-get-float-formamt-of-group-on-view
(view-1, 4, win);
```

The returned values are true, 1, 4, best, true. The default float format is used for values in cell group 4.

gxl-get-group-number-at-coordinates

Returns the group number of a cell at given spreadsheet coordinates.

Synopsis

gxl-get-group-number-at-coordinates
(*sheet*: class gxl-spreadsheet, *row*: integer, *column*: integer,
window: class g2-window)
-> *group-number*: integer

Argument	Description
sheet	The spreadsheet that is the subject of this call.
row	The row coordinate of the cell whose group number is to be returned.
column	The column coordinate of the cell whose group number is to be returned.
window	The g2-window originating this call.

Return Value	Description
group-number	The number of the cell group that contains the cell at the given row/column coordinates.

Description

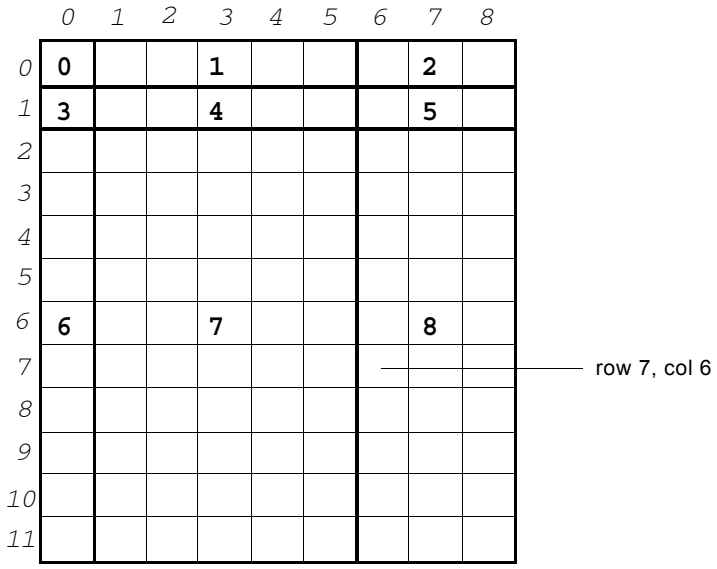
Each cell in a spreadsheet belongs to exactly one cell group. This procedure returns the number of the cell group to which a specified cell belongs. If *row* or *column* are out of bounds, this procedure signals an error (gxl-row-or-column-out-of-bounds).

Example

Using the same example as shown in gxl-get-cell-group-coordinates on , the following call yields the cell group number of the cell located at row 7, column 6:

```
group-number = call gxl-get-group-number-at-coordinates
(spreadsheet-1, 7, 6, win);
```

As the following figure illustrates, the result is group-number = 8, which contains cells in the rectangular area from (2, 6) to (11, 8).



Layout of spreadsheet-1

gxl-get-protection-of-group-on-view

Returns the selectability and editability permissions of a cell group on a spreadsheet view.

Synopsis

gxl-get-protection-of-group-on-view
(*view*: class gxl-spreadsheet-view, *group-number*: integer,
window: class g2-window)
-> *selectable*: truth-value, *editable*: truth-value

Argument	Description
view	The spreadsheet view that is the target of this call.
group-number	The reference number of the cell group whose selectability and editability status are returned.
window	The g2-window originating this call.

Return Value	Description
selectable	true if cell group can be selected.
editable	true if cell group can be edited.

Description

This procedure returns the selectability and editability status of given cell group for a given spreadsheet *view*, which are specified by the gxl-cells-are-selectable and gxl-cells-are-editable attributes of the cell group.

For details on editability and selectability, see .

If the given *group-number* is out of bounds, this procedure signals an error (gxl-group-number-out-of-bounds).

Example

If the cells of cell group 7 can be selected and edited, the following call returns true, true.

```
select, edit = call gxl-get-protection-of-group-on-view (view-1, 7, win);
```

gxl-get-selected-column-range

Returns the column numbers of the currently selected columns on a view.

Synopsis

gxl-get-selected-column-range
(*view*: class gxl-spreadsheet-view, *window*: class g2-window)
-> *first-col*: integer, *last-col*: integer

Argument	Description
view	The spreadsheet view that is the subject of this call.
window	The g2-window originating this call.

Return Value	Description
first-col	The column number of the first column selected on the view, or -1 if no column is selected.
last-col	The column number of the last column selected on the view, which may be the same as the first selected column.

Description

Cells, rows, or columns may be selected on spreadsheet views. Each view has a selection state which is independent of the selections on other views of the same spreadsheet. This procedure returns the first selected column and the last selected column in a given *view*. If no columns are selected, the procedure returns -1, -1.

Example

If columns 7 through 10 are selected on view-1, then the following call will return first-selected-column = 7 and last-selected-column = 10:

```
first-selected-column, last-selected-column = call gxl-get-selected-column-range
    (view-1, win);
```

gxl-get-selected-row-range

Returns the row numbers of the currently selected rows on a view.

Synopsis

gxl-get-selected-row-range
(*view*: class gxl-spreadsheet-view, *window*: class g2-window)
-> *first-row*: integer, *last-row*: integer

Argument	Description
view	The spreadsheet view that is the subject of this call.
window	The g2-window originating this call.

Return Value	Description
first-row	The row number of the first row selected on the view, or -1 if no row is selected.
last-row	The row number of the last row selected on the view, which may be the same as the first selected row.

Description

Cells, rows, or columns may be selected on spreadsheet views. Each view has a selection state which is independent of the selections on other views of the same spreadsheet. This procedure returns the first selected row and the last selected row in a given *view*. If no rows are selected, the procedure returns -1, -1.

Example

If rows 7 through 10 are selected on view-1, then the following call will return first-selected-row = 7 and last-selected-row = 10:

```
first-selected-row, last-selected-row = call gxl-get-selected-row-range
(view-1, win);
```

gxl-get-selection-limits

Returns the coordinates of the currently selected rectangular area on a view.

Synopsis

gxl-get-selection-limits
(*view*: class gxl-spreadsheet-view, *window*: class g2-window)
-> *first-cell-row*: integer, *first-cell-col*: integer,
last-cell-row: integer, *last-cell-col*: integer

Argument	Description
view	The spreadsheet view that is the subject of this call.
window	The g2-window originating this call.

Return Value	Description
first-cell-row	The row number of the first cell selected on the view, or -1 if no cell is selected.
first-cell-col	The column number of the first cell selected on the view or -1 if no cell is selected.
last-cell-row	The row number of the last cell selected on the view, which may be the same as the row of the first selected cell.
last-cell-col	The column number of the last cell selected on the view, which may be the same as the column of the first selected cell.

Description

Selections are always rectangular areas on a view. This procedure returns the coordinates of the first selected cell (the cell in the upper left-hand corner of the selection) and the coordinates of the last selected cell (the cell in the lower right-hand corner of the selection), which may be the same if a single cell is selected.

If no cells are selected, this procedure returns -1, -1, -1, -1. If a column is selected, the first row will be 0 and the last row will be the last row of the spreadsheet. If a row is selected, the first column will be 0 and the last column will be the last column of the spreadsheet.

Example

If we select cell (1, 1) and then shift-select cell (4, 5) on view-1, thus selecting the rectangular area delimited by these two cells, then the following call will return first-row = 1, first-column = 1, last-row = 4, and last-column = 5:

```
first-row, first-column, last-row, last-column = call gxl-get-selection-limits
(view-1, win);
```

gxl-get-size-attributes-of-cells-in-view

Returns the font size, width, and height of cells in a cell group on a spreadsheet view.

Synopsis

gxl-get-size-attributes-of-cells-in-view
(*view*: class gxl-spreadsheet-view, *group-number*: integer,
Window: class g2-window)
-> *width*: integer, *height*: integer, *font*: symbol

Argument	Description
view	The spreadsheet view that is the target of this call.
group-number	The reference number of the cell group whose cell attributes are returned.
window	The g2-window originating this call.

Return Value	Description
width	The width of the cell in the cell group of the *View*
height	The height of the cell in the cell group of the *View*.
font	The size of the font used in the cell group of the *View*.

Description

For a given spreadsheet *view*, this procedure returns the following cell appearance attributes of the cell group associated with a given *group-number*:

- Cell width
- Cell height
- Font size

If the cell group associated with the given *group-number* is out of bounds, this procedure signals an error (gxl-group-number-out-of-bounds). If no cells are

visible on the *View* for the given cell group, this procedure signals an error (gxl-no-visible-cells).

For a discussion on how the settings of these attributes affect the view display, see Customizing the Data Display in Cells.

Example

The following call returns the cell width and height, and font size used with cell group 7:

```
width, height, font = call gxl-get-size-attributes-of-cells-in-view
(view-1, 7, win);
```

The returned values are 80, 28, small, that is, the cells of cell group 7 are 80 pixels wide, 28 pixels high and use small font to display data.

gxl-get-specification-object

Provides a convenient way to retrieve a given specification object by its position in a graphical specification.

Synopsis

gxl-get-specification-object
(*root-spec*: class gxl-root-specification, *type*: symbol, *number*: integer,
window: class g2-window)
-> *specification*: class gxl-specification-object

Argument	Description
root-spec	The root of the graphical specification from which a connected object is to be retrieved.
type	The type of specification object to be returned, one of the symbols row-controller, column-controller, or cell-group.
number	An integer giving the coordinate of the specification object to be retrieved
window	The g2-window originating this call.

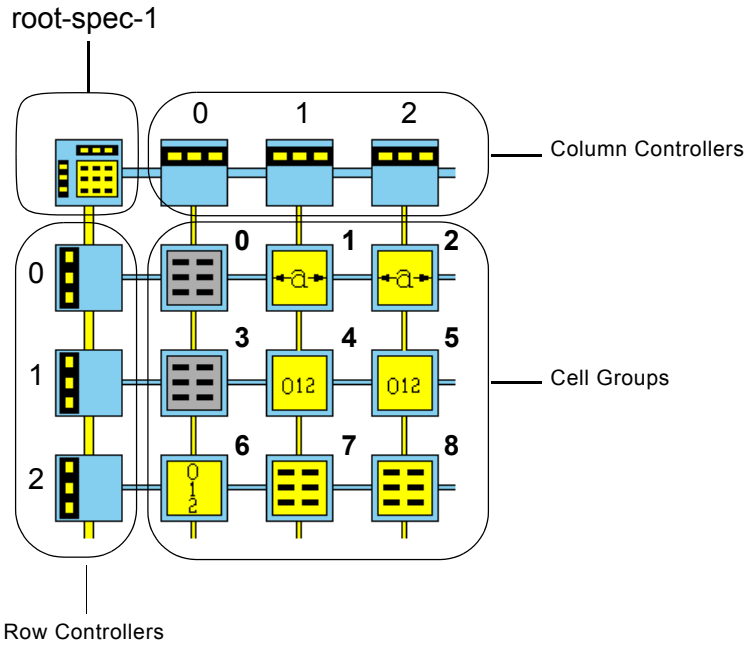
Value	Description
specification	The specification object of the given type and location.

Description

This procedure returns a specification object that is part of a graphical specification connected to a given *root-spec*. You may use this procedure to get a specification object when you need to set an attribute in an object that is part of a specification.

Example

Consider this specification:



To retrieve the column controller labelled "2", make the following call:

```
col-controller-2 = call gxl-get-specification-object (root-spec-1,
the symbol column-controller, 2, win);
```

To retrieve the cell group labelled "7", make the following call:

```
cell-group-7 = call gxl-get-specification-object (root-spec-1,
the symbol cell-group, 7, win);
```

gxl-get-specification-of-spreadsheet

Returns the specification that was used to create a spreadsheet, if it exists.

Synopsis

gxl-get-specification-of-spreadsheet
(*sheet*: class gxl-spreadsheet, *window*: class g2-window)
-> *specification*: class gxl-root-specification

Argument	Description
sheet	The spreadsheet whose specification is to be found.
window	The g2-window originating this call.

Return Value	Description
specification	The root specification of the specification that provided the template for creation of the spreadsheet.

Description

Spreadsheets are created from specifications, and a spreadsheet retains a pointer to the root specification object from which it was created. This procedure returns the specification associated with a spreadsheet. If the specification does not exist, this procedure signals the error gxl-no-specification. Typically, you would use this procedure to find an appropriate specification when you want to create a view of the spreadsheet, if the same specification is to be used to create the view as was used to create the spreadsheet.

Example

The following call returns the specification associated with spreadsheet-1:

```
root-spec-1 = call gxl-get-specification-of-spreadsheet(spreadsheet-1, win);
```

gxl-get-spreadsheet-dimensions

Returns the current size of a spreadsheet.

Synopsis

gxl-get-spreadsheet-dimensions
(*sheet*: class gxl-spreadsheet, *window*: class g2-window)
-> *rows*: integer, *cols*: integer

Argument	Description
sheet	The spreadsheet that is the subject of this call.
window	The g2-window originating this call.

Return Value	Description
rows	The number of rows in the spreadsheet.
cols	The number of columns in the spreadsheet.

Description

This procedure returns the number of rows and columns in a spreadsheet.

Example

The following call determines the dimensions of spreadsheet-1:

```
nrows, ncolumns = call gxl-get-spreadsheet-dimensions(spreadsheet-1, win);
```

gxl-get-spreadsheet-of-view

Returns the spreadsheet associated with a view.

Synopsis

gxl-get-spreadsheet-of-view
(*view*: class gxl-spreadsheet-view, *window*: class g2-window)
-> *spreadsheet*: class gxl-spreadsheet

Argument	Description
view	The view that is the subject of this call.
window	The g2-window originating this call.

Return Value	Description
spreadsheet	The spreadsheet that is the data source for the view.

Description

Each view has exactly one spreadsheet that serves as the data source for the view. This procedure returns the spreadsheet associated with a view. If the spreadsheet does not exist, this procedure signals an error (gxl-no-spreadsheet).

Example

The following call returns the spreadsheet associated with view-1:

```
spreadsheet-1 = call gxl-get-spreadsheet-of-view(view-1, win);
```

gxl-get-version

Returns the version of the GXL module.

Synopsis

```
gxl-get-version ()
    ->  version: text, sequence-number: integer
```

Return Value	Description
version	A text describing the current version of the module.
sequence-number	A quantity which increments on every revision of the module.

Description

This procedure allows you to find out the version of the gxl module that is currently loaded. The *version* is returned as text in the format:

Major.Minor Type Revision

where:

Major is an integer representing the major release number.

Minor is an integer representing the minor release number.

Type is a word describing the type of release, such as *Revision*, *Alpha* or *Beta*.

Revision is an integer representing the revision number.

For example, the first beta release might be:

"7.0 Beta 0a"

The format of this string is subject to change.

The *revision* can be used to establish the release order of different versions of the module. This number will increase with each revision of the software.

Example

The following call returns version information about the gxl module:

```
version, sequence-number = call gxl-get-version();
```

gxl-get-views-of-spreadsheet

Returns a list of views associated with a spreadsheet.

Synopsis

gxl-get-views-of-spreadsheet
(*sheet*: class gxl-spreadsheet, *window*: class g2-window)
-> *views*: class item-list

Argument	Description
sheet	The spreadsheet whose specification is to be found.
window	The g2-window originating this call.

Return Value	Description
views	The list of views.

Description

A spreadsheet can have an arbitrary number of views. This procedure returns all the views associated with a spreadsheet in an item list. If there are no views, the item list is returned empty.

To determine what windows, if any, a view is displayed upon, you must iterate over all g2-windows and use the G2 system procedure g2-item-is-showing-on-window to tell you for each window if the view is visible.

Example

The following call returns the views associated with spreadsheet-1, where views is type item-list:

```
views = call gxl-get-views-of-spreadsheet(spreadsheet-1, win);
```

gxl-get-workspace-location-of-cell

Returns the left, top, right, and bottom workspace coordinates of a cell displayed on a spreadsheet view.

Synopsis

gxl-get-workspace-location-of-cell
(*view*: class gxl-spreadsheet-view, *row*: integer, *column*: integer
window: class g2-window)
-> *left*: integer, *top*: integer, *right*: integer, *bottom*: integer

Argument	Description
view	The spreadsheet view that is the target of this call.
row	The row location of a particular cell
column	The column location of a particular cell
window	The g2-window originating this call.

Return Value	Description
left	The top left x coordinate workspace location of a particular cell on the *View*.
top	The top left y coordinate workspace location of a particular cell on the *View*.
right	The bottom right x coordinate workspace location of a particular cell on the *View*.
bottom	The bottom right y coordinate workspace location of a particular cell on the *View*.

Description

Based on the *row*, *column* coordinate location of a cell on a given spreadsheet view, this procedure returns the corresponding workspace location of the cell. The coordinates are measured in pixels, where *left*, *top* and *right*, *bottom* pairs

equal the top left x,y location and bottom right x,y location of the cell on the workspace.

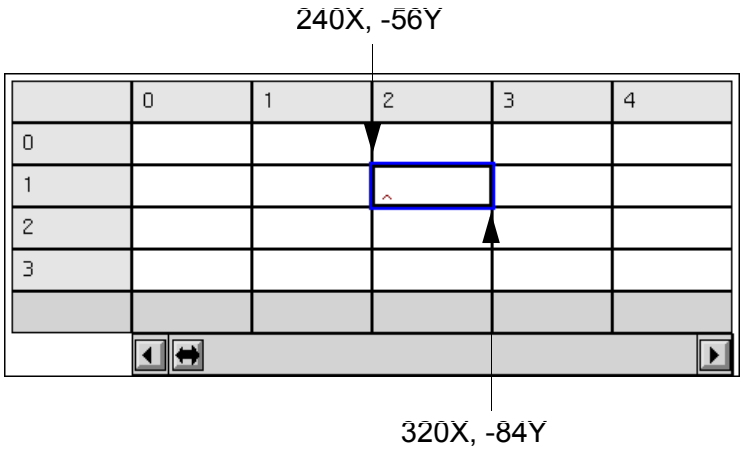
If the *row*, *column* coordinates are out of bounds, this procedure signals an error (gxl-group-number-out-of-bounds). If no cells are visible on the view for the given row/column coordinates, this procedure signals an error (gxl-no-visible-cells).

Example

The following call returns the workspace coordinates of the highlighted cell on view-1, based on its row, column location.

```
left, top, right, bottom = call gxl-get-workspace-location-of-cell
(view-1, 2, 3, win);
```

The workspace coordinates are 240, -56, 320, -84



gxl-get-workspace-location-of-cell-group

Returns the left, top, right, and bottom workspace coordinates of a cell group on a spreadsheet view.

Synopsis

gxl-get-workspace-location-of-cell-group
(*view*: class gxl-spreadsheet-view, *group-number*: integer,
window: class g2-window)
-> *left*: integer, *top*: integer, *right*: integer, *bottom*: integer

Argument	Description
view	The spreadsheet view that is the target of this call.
group-number	The reference number of the cell group whose workspace location coordinates are returned.
window	The g2-window originating this call.

Return Value	Description
left	The top left x coordinate workspace location of a particular cell group on the *View.*
top	The top left y coordinate workspace location of a particular cell group on the *View*.
right	The bottom right x coordinate workspace location of a particular cell group on the *View.*
bottom	The bottom right y coordinate workspace location of a particular cell group on the *View*.

Description

Use this procedure to get the workspace location of a particular cell group on a given *view*, where the *left*, *top* pair and *right*, *bottom* pair equal the top left x,y location and bottom right x,y location of the cell group on the workspace.

If the given cell group is out of bounds, this procedure signals an error (gxl-group-number-out-of-bounds). If the given cell group does not currently exist within a scrollable area, this procedure signals an error (gxl-no-visible-cell).

Example

The following call returns the coordinate location of cell group 7 on view-1:

```
left, top, right, bottom = call gxl-get-workspace-location-of-cell-group
(view-1, 7, win);
```

The returned coordinates are 50, -50, 190, -370

11

Setting Spreadsheet and View Properties

Describes the API procedures for setting GXL spreadsheet and view properties.

Introduction

This chapter describes the procedures of the Application Programmer's Interface to GXL that you can use to programmatically to set properties of spreadsheets and views.

The following table summarizes the properties of spreadsheets and views that you can set programmatically:

To set this property...	Use this procedure...
Contents of a cell	gxl-set-cell-contents
Color pattern of a cell	gxl-set-color-pattern-of-cell, gxl-set-color-pattern-of-cell-to-default
Color patterns of spreadsheet to their default settings	gxl-set-all-color-patterns-to-default
Contents of a column header cell	gxl-set-group-column-header
Display format of floating point numbers for a cell group on an existing view	gxl-set-float-format-of-group-on-view
User access for selecting/editing cells on a view	gxl-set-protection-of-group-on-view gxl-set-protection-on-entire-view

Using gxl-set-cell-group-procedure-attribute, you can specify or change the procedures associated with initialization, validation, or callbacks of a cell group for an existing spreadsheet.

You can also set the style of the editor used when editing formulas and cells with these procedures: gxl-set-editor-buttons and gxl-set-editor-scrolling.

See The Application Programmer's Interface for related information.

gxl-set-all-color-patterns-to-default

Sets all color patterns of a spreadsheet or a view to their default values.

Synopsis

gxl-set-all-color-patterns-to-default
(*sheet*: class gxl-spreadsheet, *window*: class g2-window)

Argument	Description
sheet	The spreadsheet that is the target of this call.
window	The g2-window originating this call.

Description

This procedure changes the background, border, and text color patterns of all the views associated with a given *sheet* to the default color patterns, and then updates the view displays to reflect the color pattern change.

Example

The following call sets the color patterns of any views associated with spreadsheet-1 to their default colors:

```
call gxl-set-all-color-patterns-to-default (spreadsheet-1, win);
```

gxl-set-cell-contents

Sets the value contained in a cell.

Synopsis

gxl-set-cell-contents
(*sheet*: class gxl-spreadsheet, *row*: integer, *column*: integer, *value:* value, *update-views:* truth-value, *window*: class g2-window)

argument	Description
sheet	The spreadsheet that is the subject of this call.
row	The row coordinate of the cell whose value is to be set.
column	The column coordinate of the cell whose value is to be set.
value	The value to be placed into the cell.
update-views	A flag indicating whether the views of the spreadsheet should be redrawn after the cell value is set.
window	The g2-window originating this call.

Description

This procedure sets the contents of a single cell of a given *sheet* to *value*, which can be an integer, float, text, symbol, or truth-value. No checking is performed to make sure that the type of value placed into the cell conforms to the type of cell in the cell group; it is the responsibility of the caller to assure the type of value is consistent with the cell type. If the cell has an active callback, this procedure produces a call to the callback.

Caution Do not use this procedure to set the value of column header cells: use gxl-set-group-column-header on for that purpose.

Normally, you call this procedure with *update-views* = true. However, if you are calling this procedure several times in a row, you will significantly improve the efficiency of your code by setting *update-views* = false, following the calls to gxl-set-cell-contents with a single call to gxl-refresh-all-views.

To clear a cell, you may call this function with *value* = "!nv".

If *row* or *column* are out of bounds, this procedure signals an error (gxl-row-or-column-out-of-bounds).

Example

The following call sets the value of the cell located at row 0, column 0 in spreadsheet-1 to the symbol ABC, and updates the views to reflect this new value:

```
call gxl-set-cell-contents(spreadsheet-1, 0, 0, the symbol abc, true, win);
```

gxl-set-cell-group-procedure-attribute

Sets any of the initialization, validation, or callback procedures associated with a cell group.

Synopsis

gxl-set-cell-group-procedure-attribute
(*sheet*: class gxl-spreadsheet, *group-number*: integer,
attribute-name: symbol, *new-value:* symbol, *window*: class g2-window)

Argument	Description
sheet	The spreadsheet that is the subject of this call.
group-number	The reference to the cell group whose procedure is to be set.
attribute-name	The name of the procedural attribute that is to be set, which is one of the following symbols: gxl-additional-validation-procedure gxl-initialization-procedure gxl-reinitialization-procedure gxl-callback-procedure gxl-selection-callback-procedure
new-value	A symbol naming a suitable initialization, validation or callback procedure, or the symbol unspecified.
window	The g2-window originating this call.

Description

This procedure sets any of the following procedural attributes of cell groups for a given *sheet*: gxl-additional-validation-procedure, gxl-initialization-procedure, gxl-reinitialization-procedure, gxl-callback-procedure, or gxl-selection-callback-procedure.

To remove a procedure from a cell group, set the relevant procedural attribute to unspecified.

Example

The following call sets the callback procedure of cell group (2) on spreadsheet-1 to my-new-callback:

```
call gxl-set-cell-group-procedure-attribute(spreadsheet-1, 2, the symbol
gxl-callback-procedure, the symbol my-new-callback, win);
```

gxl-set-color-pattern-of-cell

Sets the color pattern of a cell.

Synopsis

gxl-set-color-pattern-of-cell
(*sheet*: class gxl-spreadsheet, *row*: integer, *column*: integer,
background-color: symbol, *text-color:* symbol, *border-color:* symbol,
update-views: truth-value, *window*: class g2-window)

Argument	Description
sheet	The spreadsheet that is the subject of this call.
row	The row coordinate of the cell whose color pattern is to be changed.
column	The column coordinate of the cell whose color pattern is to be changed.
background-color	The desired background color for the cell.
text-color	The desired text color for the cell.
border-color	The desired border color for the cell.
update-views	A flag indicating whether the views of the spreadsheet should be redrawn after the cell color is set.
Window	The g2-window originating this call.

Description

Each cell in the spreadsheet has a color pattern described by its background, text and border colors. The symbols provided to this procedure are selected from the 64 color palette of G2, and do not include the meta-colors foreground and background.

If you give this procedure an invalid color, it will use white for the background and black for the text and border colors. If *row* or *column* are out of bounds, this procedure signals an error (gxl-row-or-column-out-of-bounds).

Normally, you call this procedure with *update-views* = true. However, if you are calling this procedure several times in a row, you will significantly improve the

efficiency of your code by setting *update-views* = false, and following the calls to gxl-set-color-pattern-of-cell with a single call to gxl-refresh-all-views.

Example

The following call sets the color pattern of the cell located at row 0, column 0 in spreadsheet-1 to background red, text black, and border yellow:

```
call gxl-set-color-pattern-of-cell(spreadsheet-1, 0, 0, the symbol red,
the symbol black, the symbol yellow, true, win);
```

gxl-set-color-pattern-of-cell-to-default

Sets the color pattern of a cell to the default colors defined for the cell group.

Synopsis

gxl-set-color-pattern-of-cell-to-default
(*sheet*: class gxl-spreadsheet, *row*: integer, *column*: integer,
window: class g2-window)

Argument	Description
sheet	The spreadsheet that is the subject of this call.
row	The row coordinate of the cell whose color pattern is to be changed.
column	The column coordinate of the cell whose color pattern is to be changed.
window	The g2-window originating this call.

Description

Use this procedure if you want to return the color of a cell to the color it had before any color change operations. This procedure automatically updates views.

Example

The following call returns the color pattern of the cell located at row 0, column 0 in spreadsheet-1 to its original color pattern:

```
call gxl-set-color-pattern-of-cell-to-default (spreadsheet-1, 0, 0, win);
```

gxl-set-editor-buttons

Determines whether editor buttons are shown on a particular G2 window when performing in-place cell editing and formula entry.

Synopsis

gxl-set-editor-buttons
(*show-buttons*: truth-value, *window:* class g2-window)

Argument	Description
show-buttons	Truth value determining whether to show editor buttons.
window	The g2-window where the editor behavior is to be set.

Description

This procedure controls whether spreadsheet views should include editor buttons. The buttons are the cancel, undo, paste, and language-specific buttons that normally appear at the left side of the editor when editing messages and procedures in G2. The call only affects the editor behavior on the window specified by the second argument.

Example

The following call turns on editor buttons on a window named telewindow-1:

```
call gxl-set-editor-buttons(true, telewindow-1);
```

To return to the normal appearance of the editor, use the following call:

```
call gxl-set-editor-buttons (false, telewindow-1);
```

gxl-set-editor-scrolling

Determines whether the editor displays a vertical scroll bar on a particular G2 window when performing in-place cell editing and formula entry.

Synopsis

gxl-set-editor-scrolling
(*use-scrolling*: truth-value, *window:* class g2-window)

Argument	Description
use-scrolling	Truth value determining whether or not to use scrolling.
window	The g2-window where the editor behavior is to be set.

Description

This procedure determines whether spreadsheet views should include editor scrolling. If *use-scrolling* is true, the editor will show a scroll bar on the right side of the edit area, making it convenient to edit multi-line cell values. This procedure only affects the behavior of the editor on the window specified by the second argument.

Example

The following call turns on editor scrolling on a window named telewindow-1:

```
call gxl-set-editor-scrolling(true, telewindow-1);
```

To return to the normal appearance of the editor, use the following call:

```
call gxl-set-editor-scrolling (false, telewindow-1);
```

gxl-set-float-format-of-group-on-view

Sets the formatting of floating point numbers in cell groups for an existing view.

Synopsis

gxl-set-float-format-of-group-on-view
(*View*: class gxl-spreadsheet-view, *group-number*: integer,
use-default: truth-value, *min-width:* integer, *precision*: integer,
output-format: symbol, *strip-zeros*: truth-value,
window: class g2-window)

Argument	Description
view	The view that is the subject of this call.
group-number	The reference number of the cell group on the view whose format information is to be changed.
use-default	A truth-value indicating if default formatting is to be used.
min-width	Specifies the minimum number of characters in the formatted version of float.
precision	Specifies either the number of digits to the right of the decimal point, or the significant digits, depending on the *output-format*.
output-format	Determines the representation of the float.
strip-zeros	A truth-value indicating whether zeros to the right of the last non-zero digit are to be stripped.
window	The g2-window originating this call.

Description

This procedure sets the visual representation of floats in a given *view*. The underlying value is not affected by the formatting. The *group-number* specifies the cell group whose formats are to be changed.

If *use-default* = true, the default format described in The Spreadsheet Toolbar will be applied. The remaining arguments of the float format are set, but have no effect on the appearance unless this procedure is called again with *use-default* = false.

The remaining arguments of this procedure are the same as in the G2 system procedure, g2-float-to-text:

- *min-width* specifies the minimum number of characters in the formatted value. If the formatted float value has fewer characters than the number you specify, the text in the cell is padded on the left side with space characters.
- *output-format* is a symbol indicating the representation of the value. If this argument is float, the values are displayed as floats; if exponent, exponential notation is used; if best, values are displayed as an exponent if the result is too small or large for the specified precision.
- *precision* indicates the digits to the right of the decimal, if the *output-format* is float or exponent. Otherwise, *precision* is the number of significant digits.
- *strip-zeros* controls whether zeros to the right of the last non-zero digit are removed.

After calling this procedure, you must refresh the view for the new format to take effect.

Example

The following call sets the format of floats in view-1, group 2, to a minimum width of 1, 6 digits precision, in exponential format, without stripping of trailing zeros:

```
call gxl-set-float-format-of-group-on-view (view-1, 2, false, 1, 6,
the symbol exponent, false, win);
```

gxl-set-group-column-header

Sets the value displayed in a column header.

Synopsis

gxl-set-group-column-header
(*sheet*: class gxl-spreadsheet, *group-number*: integer, *text:* text,
update-views: truth-value, *window*: class g2-window)

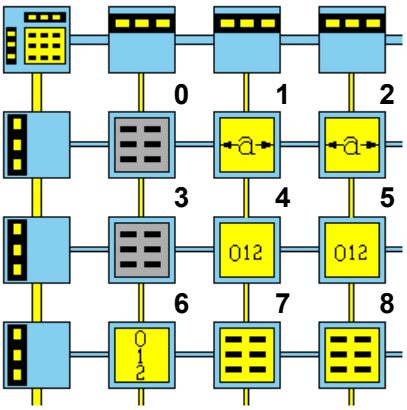
Argument	Description
sheet	The spreadsheet that is the subject of this call.
group-number	The reference number of the column header that is to be updated.
text	The new text to be displayed in the column header.
update-views	A flag indicating whether the views of the spreadsheet should be redrawn after the update.
window	The g2-window originating this call.

Description

You set the contents of a column header through this procedure, rather than gxl-set-cell-contents. This procedure uses the *group-number* to refer to the header, instead of row/column coordinates. The *update-views* argument is used in the same way as in gxl-set-cell-contents. If the *group-number* is out of bounds, this procedure signals an error (gxl-group-number-out-of-bounds).

Example

Consider the following spreadsheet specification used to create spreadsheet-1:



Groups 1 and 2 are column headers, so this procedure should only be used with *group-number* = 1 or 2.

The following call sets the column header text of group 2 in spreadsheet-1:

```
call gxl-set-group-column-header (spreadsheet-1, 2, “New Header”,
true, win);
```

gxl-set-protection-on-entire-view

Sets the editability and selectability properties of a view.

Synopsis

gxl-set-protection-on-entire-view
(*view*: class gxl-spreadsheet-view, *selectable:* truth-value,
editable: truth-value, *window*: class g2-window)

Argument	Description
view	The view that is the subject of this call.
selectable	A truth-value indicating whether the cells in the cell group on the view can be selected with the mouse.
editable	A truth-value indicating whether the cells in the cell group on the view can be edited by the user.
window	The g2-window originating this call.

Description

This procedure allows you to restrict or unrestrict selection and editing of every cell on a view. The effect of this procedure is equivalent to calling gxl-set-protection-of-group-on-view for every cell group. For details on editability and selectability, see .

Example

The following call makes a view read-only:

```
call gxl-set-protection-on-entire-view(view-1, false, false, win);
```

gxl-set-protection-of-group-on-view

Sets the editability and selectability properties of a cell group on a view.

Synopsis

gxl-set-protection-of-group-on-view
(*view*: class gxl-spreadsheet-view, *group-number*: integer,
selectable: truth-value, *editable*: truth-value, *window*: class g2-window)

Argument	Description
view	The view that is the subject of this call.
group-number	The reference number of the cell group whose protection properties are to be changed.
selectable	A truth-value indicating whether the cells in the cell group on the view can be selected with the mouse.
editable	A truth-value indicating whether the cells in the cell group on the view can be edited by the user.
window	The g2-window originating this call.

Description

This procedure allows you to restrict or unrestrict selection and editing of cells on a given *view*. Each cell group may have different protection. If cells in a cell group are selectable, you can click or drag the mouse over them to select them, and then perform operations from the tool bar such as changing the cell colors, cutting and copying their values, etc. If a cell group is not editable, you cannot edit the cell contents. If a cell is neither selectable nor editable, clicking or dragging the mouse on the cell does nothing.

Cells that are not selectable might still be indirectly selected using row and column selector cells. If you want to assure that a cell cannot be selected by row and column selectors, make the associated selector cells are unselectable.

To be editable, a cell must be selectable. Although you are not prevented from choosing this combination, setting editability to true, while the selectability is false, is inconsistent. Note that setting the editability and selectability properties to false does not prevent access to the cell contents programmatically.

Example

The following call makes cell group 1 uneditable and unselectable, thus making the cells in group 1 read-only:

```
call gxl-set-protection-of-group-on-view(view-1, 1, false, false, win);
```

12

Additional View Procedures

Describes miscellaneous API procedures related to GXL views.

Introduction

This chapter describes the procedures of the Application Programmer's Interface to GXL that you can use to programmatically manipulate views:

- gxl-move-spreadsheet-view moves a view on a workspace.
- gxl-refresh-all-views allows you to force re-draws of views, in cases where re-draws might have been suppressed, especially while loading data into the spreadsheet.
- gxl-set-selection-limits allows you to select and unselect one or more cells in a rectangular area on a view.
- gxl-scroll-to-column-in-view and gxl-scroll-to-row-in-view are the programmatic equivalents to manual scrolling.

See The Application Programmer's Interface for related information.

gxl-move-spreadsheet-view

Programmatically moves a view to a new location on a workspace.

Synopsis

gxl-move-spreadsheet-view
(*view*: class gxl-spreadsheet-view, *delta-x*: integer, *delta-y*: integer, *window*: class g2-window)

Argument	Description
view	The spreadsheet view that is the subject of this call.
delta-x	The horizontal distance in pixels that the view should be moved.
delta-y	The vertical distance in pixels that the view should be moved.
Window	The g2-window originating this call.

Description

This procedure moves a given *view* by an amount (*delta-x*, *delta-y*) on its current workspace. All items associated with the view (tool bar, pushbuttons, etc.) are also moved.

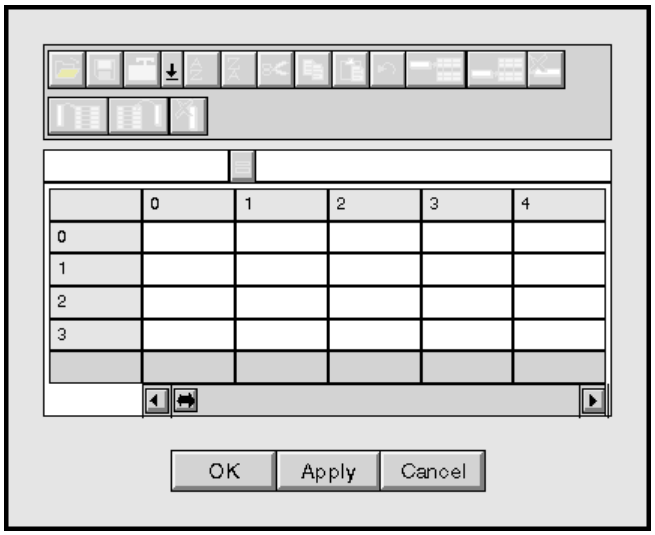
Note You cannot transfer a view between workspaces programmatically.

Example

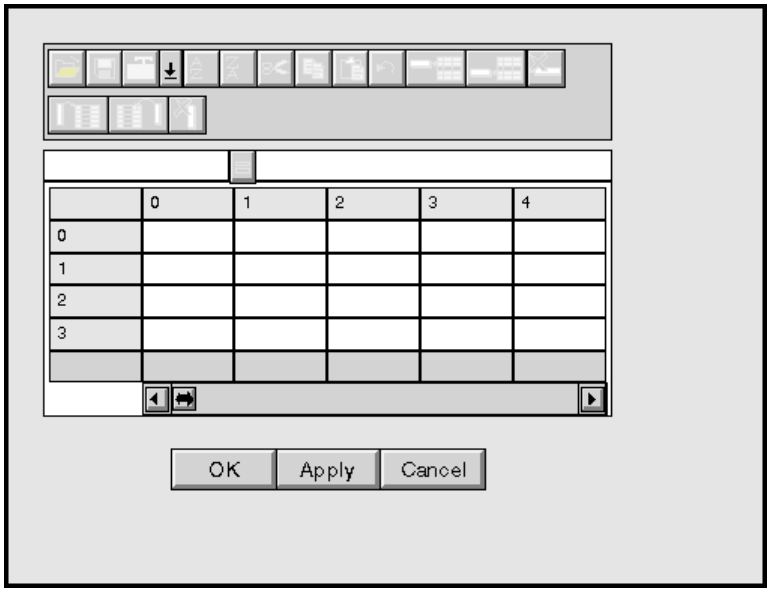
The following call shifts view-1 up 50 pixels and to the left by 100 pixels on the workspace:

```
call gxl-move-spreadsheet-view (view-1, -100, 50, win);
```

The following figure illustrates how view-1 appears before and after moving the view on the workspace:



View-1 before



View-1 after

gxl-refresh-all-views

Forces a redraw of all views associated with a spreadsheet.

Synopsis

gxl-refresh-all-views
(*sheet*: class gxl-spreadsheet, *window:* class g2-window)

Argument	Description
sheet	The spreadsheet that is the subject of this call.
window	The g2-window originating this call.

Description

This procedure forces each view associated with the given *Sheet* to redraw, using the current data and cell contents. This procedure needs to be called only if updating of the views has been explicitly suppressed (*update-views* = false) when calling one of the following procedures:

- gxl-set-cell-contents
- gxl-set-color-pattern-of-cell
- gxl-load-data-into-cell-group
- gxl-load-data-into-defined-area
- gxl-set-group-column-header with

The correct technique when you are making calls to these procedures with *update-views* = false is to call gxl-refresh-all-views once directly after the calls to these procedures, so that the work of drawing is only performed once.

Example

The following code fragment provides an example of setting cell values in an area of the spreadsheet named spreadsheet-1, with *UpdateViews* = false, and then updating views:

```
for i= first-row to last-row do
    for j= first-col to last-col do
        call gxl-set-cell-contents(spreadsheet-1, i, j, 10*i + j, false, win)
    end;
end;
call gxl-refresh-all-views(spreadsheet-1, win);
```

gxl-scroll-to-column-in-view

Programmatically scrolls a view to expose a given column.

Synopsis

gxl-scroll-to-column-in-view
(*view*: class gxl-spreadsheet-view, *target-column*: integer,
window: class g2-window)

Argument	Description
view	The view that is to be scrolled.
target-column	The column that is to be displayed.
window	The g2-window originating this call.

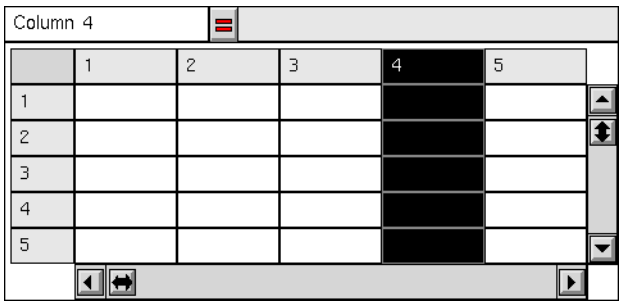
Description

This procedure scrolls so that the *target-column* is visible on a view. The *target-column* will be displayed as the first visible column, if it is possible to scroll into that position. Otherwise, the view will be scrolled as far as possible, advancing *target-column* as far left in the view as possible.

For example, if *target-column* is the last column of the spreadsheet, it cannot be advanced beyond the last column of the view, and that is where it will be displayed as a result of calling this procedure.

Example

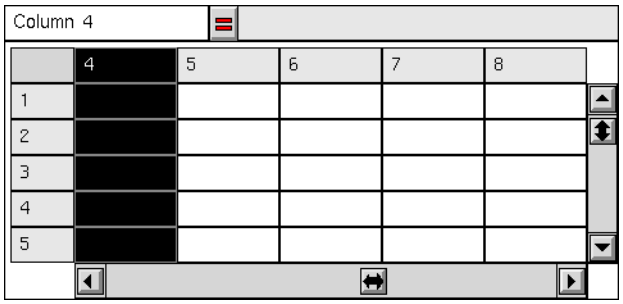
Consider the existing spreadsheet view, view-1, which currently displays the first six columns:



To scroll to the fourth column programmatically, you would invoke the following call:

```
call gxl-scroll-to-column-in-view(view-1, 5, win);
```

This is the result:



The columns scroll horizontally, displaying column 4 as the first column.

gxl-scroll-to-row-in-view

Programmatically scrolls a view to expose a given row.

Synopsis

gxl-scroll-to-row-in-view
(*view*: class gxl-spreadsheet-view, *target-row*: integer,
window: class g2-window)

Argument	Description
view	The spreadsheet view that is to be scrolled.
target-row	The row that is to be displayed.
window	The g2-window originating this call.

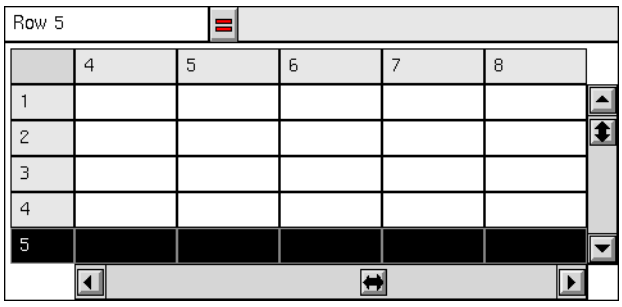
Description

This procedure scrolls so that *target-row* is visible on a view. The *target-row* will be displayed as the first visible row, if it is possible to scroll into that position. Otherwise, the view will be scrolled as far as possible, advancing *target-row* as far upward in the view as possible.

For example, if *target-row* is the last row of the spreadsheet, it cannot be advanced beyond the last row of the view, and that is where it will be displayed as a result of calling this procedure.

Example

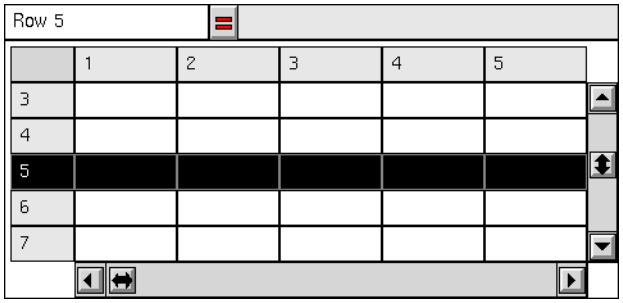
Consider the existing spreadsheet view, view-1, which currently displays the first six rows:



To scroll to the seventh row programmatically, you would invoke the following call:

```
call gxl-scroll-to-row-in-view(view-1, 8, win);
```

This is the result:



gxl-set-selection-limits

Sets the selected area of a view.

Synopsis

gxl-set-selection-limits
(*view*: class gxl-spreadsheet-view, *first-row*: integer, *first-col*: integer, *last-row*: integer, *last-col*: integer, *window*: class g2-window)

Argument	Description
view	The spreadsheet view for which the selection is to be made.
first-row	The first row of the selected area.
first-col	The first column of the selected area.
last-row	The last row of the selected area.
last-col	The last column of the selected area.
window	The g2-window originating this call.

Description

This procedure selects one or more cells in a rectangular area on a view. The area to select is defined by the *first-row*, *first-col*, *last-row*, and *last-col* arguments. When the selection is made, any existing selection is deselected.

To select a single cell, call this procedure with:

first-row = *last-row*
first-col = *last-col*

To deselect all cells, call this procedure with

first-row = -1

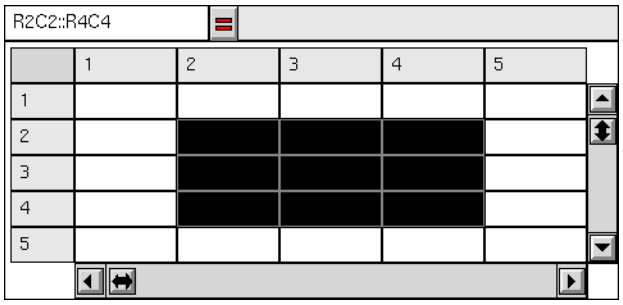
If the new selection extends to any cell groups that have selection callback procedures, a call will be made to the selection callback of each cell group affected by the selection.For details on selection callbacks, see .

Example

The following call selects the area bounded by row 2, column 2 and row 4, column 4 of view-1:

```
call gxl-set-selection-limits (view-1, 2, 2, 4, 4, win);
```

Here is the result:



13

Row and Column Operations

Describes the API methods for adding and deleting rows and columns, and API procedures for sorting.

Introduction

This chapter describes the procedures and methods of the Application Programmer's Interface to GXL that you can use to programmatically change the row and column dimensions of a spreadsheet, and perform sorting operations.

Using gxl-add-rows and gxl-add-columns methods, you can add one or more rows and columns to a spreadsheet at a specific location. Using gxl-remove-rows and gxl-remove-columns methods, you can delete one of more rows and columns of a spreadsheet. By default, GXL updates the views and associated scrolling behavior when adding or deleting rows and columns.

For sorting you can use the gxl-sort, gxl-permute-rows, and gxl-sort-and-return-permutations procedures.

See The Application Programmer's Interface for related information.

gxl-add-columns

Expands the dimension of a spreadsheet by adding one or more columns.

Synopsis

gxl-add-columns
(*sheet*: class gxl-spreadsheet, *Target*: integer, *columns-to-add*: integer, *mode:* symbol, *update-views*: truth-value, *scroll-to*: truth-value, *window*: class g2-window)

Argument	Description
sheet	The spreadsheet that is the subject of this call.
target	The column adjacent to where new columns are to be inserted, or a group number.
columns-to-add	The number of new columns to add to the spreadsheet.
mode	The symbol before, after, or group, which indicates whether the new columns are to be added before or after the *target*, or in a currently empty group.
update-views	A truth-value indicating whether to update the views after adding the columns.
scroll-to	A truth-value indicating whether to update the scroll area after adding the columns. The default value for this argument is true.
window	The g2-window originating this call.

Description

This method adds columns to a spreadsheet. Using the *mode* argument, you control the cell group assignment of the new columns. In the case that *mode* is before or after, the *target* argument is a column adjacent to where the new columns are added. The columns that are added are in the same cell groups as the cells in the target column, and are either inserted before or after the target column, as directed by the *mode* argument.

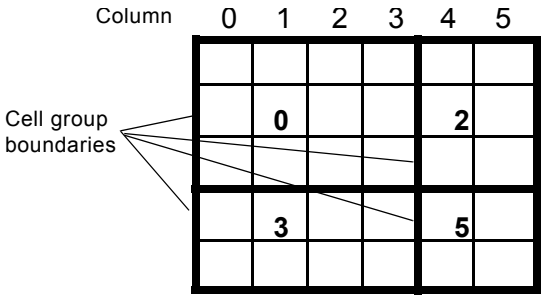
To add columns to a cell group that currently contains no columns, set the *mode* argument to the symbol group. In this case, *target* is the group number of the cell group where you want to add columns.

The cells in the columns that are added are initially empty. However, if the group has a reinitialization procedure, it will be called after the columns are added. The reinitialization procedure can be used to initialize the values of the new columns. For information on reinitialization procedures, see .

When *update-views* and *scroll-to* are set to true, this method refreshes existing views of the *sheet* to accommodate the additional columns in the view display, where *update-views* redraws the view and *scroll-to* adjusts the scroll areas of the view to include the new columns.

Example

Consider a spreadsheet consisting of six cell groups, as shown below. Cell groups 1 and 4 currently have no columns.



The following call adds 1 new column between existing columns 3 and 4, with the new column belonging to cell groups 0 and 3:

```
call gxl-add-columns(spreadsheet-1, 3, 1, the symbol after, true, true, win);
```

If you want to add a column between existing columns 3 and 4 that belongs to cell groups 2 and 5, you would use the following call:

```
call gxl-add-columns(spreadsheet-1, 4, 1, the symbol before, true, true, win);
```

To add seven columns to the currently-empty cell groups 1 and 4, you use the mode group as follows:

```
call gxl-add-columns(spreadsheet-1, 1, 7, the symbol group ,true, true, win);
```

Note that in this call, the second argument could be either group 1 or 4, and the same result would be achieved, since the number of columns in cell groups 1 and 4 are always the same.

gxl-add-rows

Expands the dimension of a spreadsheet by adding one or more rows.

Synopsis

gxl-add-rows
(*sheet*: class gxl-spreadsheet, *target*: integer, *rows-to-add*: integer, *mode:* symbol, *update-views*: truth-value, *scroll-to*: truth-value, *window*: class g2-window)

Argument	Description
sheet	The spreadsheet that is the subject of this call.
target	The row adjacent to where new rows are to be inserted, or a group number.
rows-to-add	The number of new rows to add to the spreadsheet.
mode	The symbol before, after, or group, which indicates whether the new rows are to be added before or after the *target*, or in a currently empty group.
update-views	A truth-value indicating whether to update the views after adding the rows.
scroll-to	A truth-value indicating whether to update the scroll area after adding the rows. The default value for this argument is true.
window	The g2-window originating this call.

Description

This method adds rows to a spreadsheet. Using the *mode* argument, you control the cell group assignment of the new rows. If *mode* is before or after, the *target* argument is a row adjacent to where the new rows are added. The rows that are added are in the same cell groups as the cells in the target row, and are either inserted before or after the target row, as directed by the *mode* argument.

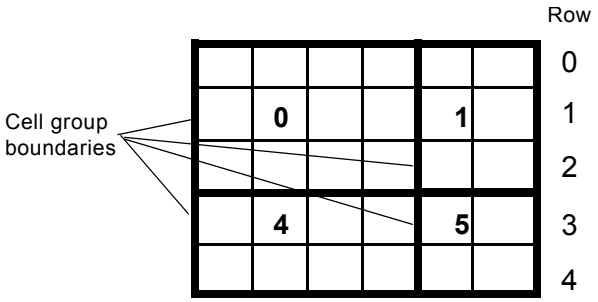
To add rows to a cell group that currently contains no rows, set the *mode* argument to the symbol group. In this case, *target* is the group number of the cell group where you want to add rows.

The rows that are added are initially empty. However, if the group has a reinitialization procedure, it will be called after the rows are added. The reinitialization procedure can be used to initialize the values of the new rows. For information on reinitialization procedures, see .

When *update-views* and *scroll-to* are set to true, this method refreshes existing views of the *sheet* to accommodate the additional rows in the view display, where *update-views* redraws the view and *scroll-to* adjusts the scroll areas of the view to include the new rows.

Example

Consider a spreadsheet consisting of six cell groups, as shown below. The cell groups 2 and 3 currently have no rows.



The following call adds 1 new row between existing rows 2 and 3, with the new row belonging to cell groups 0 and 1:

```
call gxl-add-rows(spreadsheet-1, 2, 1, the symbol after, true, true, win);
```

If you want to add a row between existing rows 2 and 3 that belongs to cell groups 4 and 5, you would use the following call:

```
call gxl-add-rows(spreadsheet-1, 3, 1, the symbol before, true, true, win);
```

To add seven rows to the currently-empty groups 2 and 3, you use the mode group, as follows:

```
call gxl-add-rows(spreadsheet-1, 2, 7, the symbol group, true, true, win);
```

Note that in this call, the second argument could be either group 2 or 3, and the effect is the same, since the number of rows in cell groups 2 and 3 are always the same.

gxl-permute-rows

Sorts rows of the target columns according to a given pattern of permutations.

Synopsis

gxl-permute-rows
(*sheet*: class gxl-spreadsheet, *first-row*: integer, *last-row*: integer,
permutations: class integer-list, *target-columns:* class integer-list,
window: class g2-window)

Argument	Description
sheet	The spreadsheet that is the subject of this call.
first-row	The first row of the target area.
last-row	The last row of the target area.
permutations	An integer list that determines the sorting of the rows of the target area.
target-columns	A list of columns that are to be included in the sorting operation.
window	The g2-window originating this call.

Description

This is a low-level sorting routine that can be used to implement sorting based on your own sorting criterion. To use this procedure, you first determine the new row ordering that you want, and represent this order in the *permutations* list.

The length of the permutations list must equal the number of rows to be sorted, and it must contain all the integers from 0 to *nrows* - 1, where:

nrows = *last-row* -*first-row* + 1

The order of the integers in this list determines the new order of the rows. If *permutations*[n] = m, then data from the m-th row of the target area is placed in the n-th row of the target area.

Only columns that are part of the *target-columns* list are included in the sort. These columns do not have to be consecutive. You must have one or more columns in *target-columns*.

Example

Consider a spreadsheet consisting of the entries shown below.

Columns

Rows	0	1	2	3	4	5
0	a	b	c	d	e	f
1	g	h	i	j	k	l
2	m	n	o	p	q	r
3	s	t	u	v	w	x

Suppose *permutations* = (2, 0, 1) and *target-columns* = (1, 2, 4). The following call will re-arrange the data in the spreadsheet as shown below:

```
call gxl-permute-rows(spreadsheet-1, 1, 3, permutations, target-columns, win);
```

Columns

Rows	0	1	2	3	4	5
0	a	b	c	d	e	f
1	g	t	u	j	w	l
2	m	h	i	p	k	r
3	s	n	o	v	q	x

gxl-remove-columns

Deletes one or more columns from the spreadsheet.

Synopsis

gxl-remove-columns
(*sheet*: class gxl-spreadsheet, *first-col*: integer,
columns-to-remove: integer, *update-views*: truth-value,
window: class g2-window)

Argument	Description
sheet	The spreadsheet that is the subject of this call.
first-col	The first column that is to be deleted.
columns-to-remove	The number of columns to remove from the spreadsheet.
update-views	A truth-value indicating whether to update the views after deleting the columns.
window	The g2-window originating this call.

Description

This method deletes *columns-to-remove* columns from a given *sheet*, starting at (and including) *first-col*.

If there are reinitialization procedures for any of the cell groups affected by this procedure, they are called after the columns are removed. For information on reinitialization procedures, see .

When *update-views* = true, this method redraws existing views of the *sheet* to remove the deleted columns.

Example

The following call deletes columns 3 and 4 from *spreadsheet-1*:

```
call gxl-remove-columns(spreadsheet-1, 3, 2, true, win);
```

gxl-remove-rows

Deletes one or more rows from the spreadsheet.

Synopsis

gxl-remove-rows
(*sheet*: class gxl-spreadsheet, *first-row*: integer, *rows-to-remove*: integer, *update-views*: truth-value, *window*: class g2-window)

Argument	Description
sheet	The spreadsheet that is the subject of this call.
first-row	The first row that is to be deleted.
rows-to-remove	The number of rows to remove from the spreadsheet.
update-views	A truth-value indicating whether to update the views after deleting the rows.
window	The g2-window originating this call.

Description

This method deletes *rows-to-remove* rows from a given *sheet*, starting at (and including) *first-row*.

If there are reinitialization procedures for any of the cell groups affected by this procedure, they are called after the columns are removed. For information on reinitialization procedures, see .

When *update-views* = true, this method redraws existing views of the *sheet* to remove the deleted rows.

Example

The following call deletes rows 3 and 4 from the spreadsheet named spreadsheet-1:

```
call gxl-remove-rows(spreadsheet-1, 3, 2, true, win);
```

gxl-sort

Sorts a range of rows of given target columns by sorting on a key column.

Synopsis

gxl-sort
(*sheet*: class gxl-spreadsheet, *key-column*: integer,
auxiliary-columns: class integer-list, *first-row:* integer, *last-row:* integer,
direction: symbol, *window*: class g2-window)

Argument	Description
sheet	The spreadsheet that is the subject of this call.
key-column	The column that provides the sorting order.
auxiliary-columns	Columns that are sorted along with the key column.
first-row	The first row of the target area.
last-row	The last row of the target area.
direction	Indicates the direction of the sort, either the symbol ascending or descending.
window	The g2-window originating this call.

Description

This procedure sorts rows in a target area by sorting the *key-column* into ascending or descending order. The columns in the *auxiliary-columns* list are sorted according to the order of sorting on the *key-column*. The *auxiliary-columns* list may be empty if only the key column is to be sorted. The *first-row* and *last-row* arguments define the row range to be included in the sort.

If *direction* is the symbol ascending, then the rows of the *key-column* are sorted in the following order:

1. Empty cells.
2. Quantities, sorted in ascending order.
3. Truth-values, with false preceding true.
4. Symbols, in alphabetical order.
5. Texts, sorted according to the order given by the G2 greater than (>) operator.

If the sort is descending, the opposite ordering applies.

No second-order sorting is carried out by this procedure; therefore, duplicate entries appear in arbitrary order.

If you want greater control over the order of sorting, you can use gxl-permute-rows to carry out arbitrary sorts using criteria you control. See for details.

Example

Consider a spreadsheet consisting of the entries shown in the following figure.

		Column 0	1	2	3
Row	0	4	1	2	3
	1	XX	4	5	6
	2	“xx”	7	8	9
	3		10	11	12

Using column 0 as the key column and using *auxiliary-columns* = (1, 3), the following call sorts the columns in ascending order:

```
call gxl-sort(spreadsheet-1, 0, auxiliary-columns, 0, 3, the symbol ascending, win);
```

Here are the results of the sort:

		Column 0	1	2	3
Row	0		10	2	12
	1	4	1	5	3
	2	XX	4	8	6
	3	“xx”	7	11	9

gxl-sort-and-return-permutations

Sorts and returns an integer list of permutations, mapping the sorted results to the original list.

Synopsis

gxl-sort-and-return-permutations
(*sheet*: class gxl-spreadsheet, *key-column*: integer,
auxiliary-columns: class integer-list, *first-row*: integer, *last-row*, integer,
direction: symbol, *window*: class g2-window)
-> *permutations*: class: integer-list

Argument	Description
sheet	The spreadsheet that is the target of this call.
key-column	The column that provides the sorting order.
auxiliary-columns	Columns that are sorted along with the key column.
first-row	The first row of the target area.
last-row	The last row of the target area.
direction	Indicates the direction of the sort, either the symbol ascending or descending.
window	The g2-window originating this call.

Return Value	Description
permutations	An integer-list of the sorted orders of the *KeyColumn* and any auxiliary columns.

Description

This procedure, like gxl-sort, sorts rows in a target area by sorting the *key-column* into ascending or descending order. The columns in the *auxiliary-columns* list are sorted according to the order of sorting on the *key-column*. See gxl-sort on , which describes the order of the sort based on the data values in the key and auxiliary columns.

The *auxiliary-columns* list may be empty if only the key column is to be sorted. *first-row* and *last-row* define the row range to be included in the sort.

This procedure returns the results of the sort in the form of an integer-list.

Example

The following call sorts the specified columns of data in spreadsheet-1 and returns an integer list of the permutations.

```
permutations = call gxl-sort-and-return-permutations (spreadsheet-1, 2,
    aux-columns, 2, 6, descending, win);
```

14

Toolbar Procedures

Describes the API procedures for managing the appearance of toolbars on GXL spreadsheet views.

Introduction

This chapter describes the procedures of the Application Programmer's Interface to GXL that allow you to programmatically manage the appearance of toolbars on spreadsheet views.

To...	Use this procedure...
Create an empty toolbar on the workspace of a view	gxl-add-toolbar-to-view
Add the built-in buttons to the toolbar	gxl-add-built-in-tools-to-toolbar
Add your own buttons, or any individual button to the toolbar	gxl-add-tool-to-toolbar
Add a button or toolbar to an existing view	gxl-add-accoutrement-to-view
Get a list of buttons of a toolbar on a given view	gxl-get-toolbar-of view

Procedures Controlling the Undo Function

The procedures gxl-get-undo-buffer, gxl-backup-area-into-undo-buffer, and gxl-restore-area-from-undo-buffer provide the interface to the undo function, allowing programmatic control over undo functions. You can use these procedures when implementing your own toolbar buttons.

See The Application Programmer's Interface for related information.

gxl-add-accoutrement-to-view

Associates an existing button or toolbar with an existing spreadsheet view.

Synopsis

gxl-add-accoutrement-to-view
(*view*: class item, *accoutrement*: class item, *window*: class g2-window)

Argument	Description
view	The existing spreadsheet view on which you want to add a button or toolbar.
accoutrement	The tool, button, or toolbar you wish to add to the *View*.
window	The g2-window originating this call.

Description

This procedure associates a given *accoutrement* to an existing *view*. The toolbar or button should already be on the workspace of the *view*, at the desired position. Note that if you programmatically create a button, it will not be recognized as associated with a view until this procedure is called.

Example

The following call associates the custom toolbar, tools-2 to the existing view, view-2.

```
call gxl-add-accoutrement-to-view (view-2, tools-2, win)
```

gxl-add-built-in-tools-to-toolbar

Programmatically adds buttons to an existing toolbar.

Synopsis

gxl-add-built-in-tools-to-toolbar
(*toolbar*: class gxl-toolbar, *tools*: class symbol-array,
window: class g2-window)

Argument	Description
toolbar	The toolbar to which the buttons should be added.
tools	An array containing one or more of the following symbols: file-save, file-load, insert-row-after, insert-row-before, insert-column-after, insert-column-before, delete-column, delete-row, select-color, cut, copy, paste, undo, sort-ascending, sort-descending, rc-indicator, formula-tool
window	The g2-window originating this call.

Description

This procedure adds the buttons specified in the *tools* array to the specified *toolbar*. The buttons are added in the order they appear in *tools* from left to right on the toolbar. If necessary, the buttons will be laid out in multiple rows.

Example

The following code fragment adds the cut, copy and paste buttons to toolbar-1:

```
create a symbol-array tools;
change the array-length of tools to 3;
change tools[0] = the symbol cut;
change tools[1] = the symbol copy;
change tools[2] = the symbol paste;
call gxl-add-built-in-tools-to-toolbar(toolbar-1, tools, win);
delete tools;
```

gxl-add-toolbar-to-view

Adds a toolbar to a view.

Synopsis

gxl-add-toolbar-to-view
(*view*: class gxl-spreadsheet-view, *width*: integer, *window*: class g2-window)
-> *toolbar*: class gxl-toolbar

Argument	Description
view	The spreadsheet view to which the toolbar is to be added.
width	The desired width of the toolbar, or 0, which defaults the toolbar width to the width of the view.
window	The g2-window originating this call.

Return Value	Description
toolbar	The toolbar created by this call.

Description

This procedure creates an empty toolbar of the given *width* and places it upon the workspace of the existing *view* as follows:

- Places the toolbar above the *view* and above the associated row-column indicator and formula bar, if any.
- Aligns the left side of the toolbar with the left side of the *view*.
- Determines the width of the toolbar from the *width* argument.

The height of the toolbar is 43 pixels.

To make the width of the toolbar equal to the width of the view, call this procedure with *width* = 0.

Once created and populated with buttons, you can move or resize the toolbar manually or programmatically using G2's move command or the system procedure g2-change-size-of-item-per-area.

Although you can have more than one toolbar for a given view, this procedure will not provide an adequate layout for multiple toolbars.

Example

The following call creates an empty toolbar of the default width and places it on the workspace containing the spreadsheet view named view-1:

```
call gxl-add-toolbar-to-view (view-1, 0, win);
```

gxl-add-tool-to-toolbar

Adds a single item to the toolbar.

Synopsis

gxl-add-tool-to-toolbar
(*toolbar*: class gxl-toolbar, *tool*: class item, *window*: class g2-window)

Argument	Description
toolbar	The toolbar to which the item should be added.
tool	Any item to be added to the toolbar.
window	The g2-window originating this call.

Description

This procedure adds an item to the *toolbar*, placing it just to the right of the last tool in the bottom row of tools. If there is insufficient room in the bottom row of tools for the new tool, the existing tools will be pushed up to start another row of tools below the others.

To create an attractive layout with a custom tool, your icon should be 32 pixels in height.

You can adjust the location of a tool once it has been added to the toolbar, using any G2 method of moving an item.

Example

The following code adds a custom button named my-button to toolbar-1:

```
call gxl-add-tool-to-toolbar(toolbar-1, my-button, win);
```

gxl-backup-area-into-undo-buffer

Fills an undo buffer with data from a given area of the spreadsheet.

Synopsis

gxl-backup-area-into-undo-buffer
(*buffer*: class gxl-undo-buffer, *sheet*: class gxl-spreadsheet,
first-row: integer, *first-col:* integer,*last-row:* integer, *last-col:* integer,
window: class g2-window)

Argument	Description
buffer	The undo buffer that is to be loaded with data.
sheet	The spreadsheet that is providing the data.
first-row	The first row of the backup area.
first-col	The first column of the backup area.
last-row	The last row of the backup area.
last-col	The last column of the backup area.
window	The g2-window originating this call.

Description

This procedure can be used by custom toolbar buttons to provide undo functionality. You first retrieve the undo buffer for view using gxl-get-undo-buffer. Calling gxl-backup-area-into-undo-buffer copies the data and formats from a spreadsheet area to the buffer. The procedure gxl-restore-area-from-undo-buffer can then be used to implement the undo functionality, if you do not use the built-in undo button.

Example

The following code fragment backs up the first five rows and columns of data associated with the spreadsheet view named view-1:

```
buffer-1 = call gxl-get-undo-buffer(view-1, win);
spreadsheet-1 = call gxl-get-spreadsheet-of-view(view-1, win);
call gxl-backup-area-into-undo-buffer(buffer-1, spreadsheet-1, 0, 0, 4, 4, win);
```

gxl-get-toolbar-of view

Returns the toolbar associated with a spreadsheet view.

Synopsis

gxl-get-toolbar-of-view
(*view*: class gxl-spreadsheet-view, *window*: class g2-window)
-> *toolbars*: class item-list

Argument	Description
view	The spreadsheet view that is the subject of this call.
window	The g2-window originating this call.

Return Value	Description
toolbars	An item list containing the toolbars on the given view.

Description

This procedure returns an item list containing the toolbar or toolbars associated with a given *view*. If the view does not have a toolbar, the returned item list will be empty. The elements of the item list are of class gxl-toolbar.

You can use this procedure in preparation for adding a custom button to the toolbar, or programmatically repositioning or resizing it. You may use the G2 system procedures g2-move-from-area-of-workspace and g2-change-size-of-item-per-area for positioning or resizing the toolbar.

Example

The following call returns toolbars associated with view1, a spreadsheet view:

```
IL = call gxl-get-toolbar-of-view(view1, win);
for TB = each gxl-toolbar in IL do
    inform the operator that “Found a toolbar with width [the item-width of TB],
    height [the item-height of TB], and upper-left corner at ([gfr-left(TB)],
    [gfr-top(TB)])”;
end;
delete IL;
```

gxl-get-undo-buffer

Returns the undo buffer associated with a view.

Synopsis

gxl-get-undo-buffer
(*view*: class gxl-spreadsheet-view, *window*: class g2-window)
-> *buffer*: class gxl-undo-buffer

Argument	Description
view	The spreadsheet view whose undo buffer is to be retrieved.
window	The g2-window originating this call.

Return Value	Description
buffer	The undo buffer associated with the view.

Description

This procedure retrieves the undo buffer associated with a given *view*.

Example

The following call retrieves the undo buffer associated with the spreadsheet view named view-1:

```
buffer-1 = call gxl-get-undo-buffer(view-1, win);
```

gxl-restore-area-from-undo-buffer

Copies the data from the undo buffer back into the spreadsheet location it came from.

Synopsis

gxl-restore-area-from-undo-buffer
(*buffer*: class gxl-undo-buffer, *sheet*: class gxl-spreadsheet,
window: class g2-window)

Argument	Description
buffer	The undo buffer that provides the data for the undo.
sheet	The spreadsheet that is the target of the undo action.
window	The g2-window originating this call.

Description

This procedure can be used by custom toolbar buttons to provide undo functionality. Calling gxl-restore-area-from-undo-buffer restores an area of the spreadsheet that was backed up using gxl-backup-area-into-undo-buffer. The data and formats are returned to the same area of the spreadsheet.

Example

The following code fragment restores backup data associated with *view-1*:

```
buffer-1 = call gxl-get-undo-buffer(view-1, win);
spreadsheet-1 = call gxl-get-spreadsheet-of-view(view-1, win);
call gxl-restore-area-from-undo-buffer(buffer-1, spreadsheet-1, win);
```

15

Tabular Edit Operations

Describes the API procedures for launching and managing GXL edit sessions.

Introduction

This chapter describes the procedures of the Application Programmer's Interface to GXL that you can use to programmatically set up edit sessions on objects of various types, including G2 lists and arrays, and spreadsheets. You can use:

- gxl-edit-simple-tabular-object to start an edit session on lists or arrays.
- gxl-edit-spreadsheet to start an edit session on an existing spreadsheet.

If you programmatically launch an edit session, you may use gxl-wait-for-pushbuttons-on-view to suspend your procedure until the user presses the OK, Apply or Cancel buttons on the view.

To set up the callbacks you want to receive when the user applies the changes made during the edit session or ends the session, use gxl-set-pushbutton-callback. If you provide a callback, use gxl-get-view-of-pushbutton to find the view associated with the button receiving the callback. You may use gxl-apply-tabular-edit to unload data from cell groups into the data sources that were used when loading the spreadsheet with data.

See The Application Programmer's Interface for related information.

gxl-apply-tabular-edit

Causes data to be unloaded from the spreadsheet back to the source objects.

Synopsis

gxl-apply-tabular-edit
(*sheet*: class gxl-spreadsheet, *window*: class g2-window)

Argument	Description
sheet	The spreadsheet that is the subject of this call.
window	The g2-window originating this call.

Description

This procedure provides a shortcut to unloading data from a given *sheet*. It causes data from the spreadsheet to be copied back into objects which originally provided the data to the spreadsheet. It is used in conjunction with gxl-load-data-into-cell-group.

When you call gxl-load-data-into-cell-group, the spreadsheet retains a record of what object provided the data, and the protocol and wrapping data used in loading the data. If you later call gxl-apply-tabular-edit, and if the data source object still exists, a call is made to gxl-unload-data-from-cell-group, using the same protocol and wrapping data. This procedure will unload all cell groups that were loaded using gxl-load-data-into-cell-group.

This procedure is generally used to end edit sessions and automatically conclude values back to their source.

Example

The following code illustrates how this procedure is used to unload data that was loaded using gxl-load-data-into-cell-group ( ). Assume the existence of a spreadsheet with two cell groups, where vals-1 is a value-array, wrapping-data-1 is an integer-list, and vals-2 is a symbol-list.

```
call gxl-load-data-into-cell-group(spreadsheet-1, 0, vals-1,
the symbol columnwise-wrapping, wrapping-data-1, true, win);
call gxl-load-data-into-cell-group(spreadsheet-1, 1, vals-2,
the symbol rowwise, the symbol default, true, win);
```

Vals-1 is now loaded into cell group 0 of spreadsheet-1, and vals-2 is loaded into cell group 2. Later, perhaps in a different procedure, we can make the following call:

```
call gxl-apply-tabular-edit(spreadsheet-1, win);
```

This call causes the current values in cell group 0 to be unloaded with the protocol columnwise-wrapping into vals-1 and wrapping-data-1, and the contents of cell group 1 to be unloaded into vals-2 with the protocol rowwise and wrapping data default.

gxl-edit-simple-tabular-object

Creates a spreadsheet to edit an array or list and displays a view on a given window.

Synopsis

gxl-edit-simple-tabular-object
(*values*: class object, *visible-rows:* integer, *visible-columns:* integer, *Window:* class g2-window)

Argument	Description
values	A value-list, value-array, item-list or item-array that is to be edited.
visible-rows	The number of rows on the view that is created by this call.
visible-columns	The number of columns on the view that is created by this call. This argument is ignored when editing a value array or value list.
window	The g2-window where the view is to be displayed.

Description

This procedure provides a convenient way to edit G2 arrays and lists using a default specification that is built into GXL. Using this call, you can only control the number of rows and columns displayed on the view. The total row/column dimensions are set by the dimensions of *values*. When editing vector data, the *visible-columns* argument is ignored and the data is always presented in a single column. If the defaults provided by gxl-edit-simple-tabular-object are not acceptable, you must create your own specification.

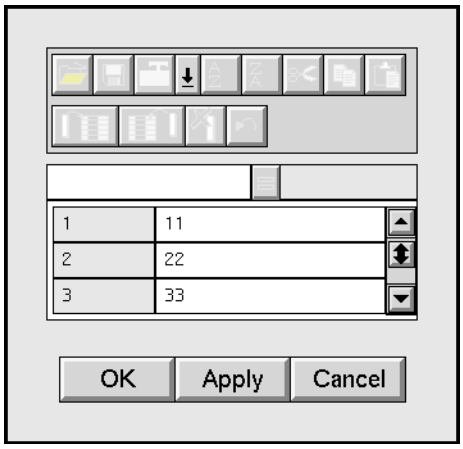
The *values* argument can be any value list or value array, or an item list or item array, provided that the items in the item list or array are themselves value lists or value arrays. If the list or array that is edited is empty, one active cell is provided on the view.

When an edit is in progress on an object, it cannot be edited by another user.

Example

If integer-array-1 = (11, 22, 33, 44, 55, 66, 77, 88), then the following call produces the view depicted below:

```
call gxl-edit-simple-tabular-object(integer-array-1, 3, 1, win);
```



gxl-edit-spreadsheet

Begins an edit session with the given spreadsheet as the data source.

Synopsis

gxl-edit-spreadsheet
(*sheet*: class gxl-spreadsheet, *specification:* class gxl-root-specification,
window: class g2-window)

Argument	Description
sheet	The spreadsheet that is to be edited.
specification	A root specification that defines properties of the view created by this call.
window	The g2-window where the view is to be displayed.

Description

This procedure provides a convenient way to edit spreadsheets. When you call this procedure, a clone of *sheet* is created; and a view of the clone is created using *specification* and displayed on *Window*.

Modifications of the data via the view are stored in the cloned spreadsheet until you select the OK or Apply buttons on the view, which causes the data in the cloned spreadsheet to be copied to *sheet*. If you select the Cancel button on the view, the cloned spreadsheet is deleted, and the original spreadsheet is unchanged.

Only one user can edit a given spreadsheet at a time.

Example

The following call starts an edit session on the spreadsheet named spreadsheet-1, using the specification used in the creation of spreadsheet-1:

```
specification = call gxl-get-specification-of-spreadsheet(spreadsheet-1, win);
call gxl-edit-spreadsheet(spreadsheet-1, specification, win);
```

gxl-get-view-of-pushbutton

Returns the view associated with a pushbutton.

Synopsis

gxl-get-view-of-pushbutton
(*button:* class gxl-pushbutton, *window*: class g2-window)
-> *view*: item-or-value

Argument	Description
button	The button that is the subject of this call.
window	The g2-window originating this call.

Return Value	Description
view	The view associated with the button, or false.

Description

This procedure tells you which spreadsheet view, if any, to which a specified *button* is associated. It is typically used when you provide your own callback for a pushbutton. This procedure returns false if there is no view associated with the *button*.

Example

The following returns the view associated with the OK button:

```
view-1 = call gxl-get-view-of-pushbutton(ok-button, win);
```

gxl-set-pushbutton-callback

Sets the callback procedure for a pushbutton.

Synopsis

gxl-set-pushbutton-callback
(*view*: class gxl-spreadsheet-view, *button-id*: text, *Callback:* symbol,
window: class g2-window)

Argument	Description
view	The view that is associated with the target button.
button-id	The gxl-id attribute of the target button.
callback	The name of the procedure that should be called when the button is selected.
window	The g2-window originating this call.

Description

Use this procedure when you want to replace the standard callback on the OK, Apply, and Cancel buttons. This procedure locates the button based on the given *button-id* and the associated *view*. To set the callbacks of the standard OK, Apply and Cancel buttons on a view, refer to the IDs "OK", "Apply" and "Cancel".

You can also use this procedure to set the callback of an existing toolbar button or custom button. To use this procedure with a custom button, you should set the ID of the button in the class definition, or on the instance when it is first added to the toolbar.

The callback you provide must have the following signature:

```
my-callback (Button: class gxl-pushbutton, Win: class g2-window)
```

You may set the callback to the symbol unspecified, if you do not want control to be passed to a callback when the button is selected.

To determine the view associated with the button when your callback is called, use gxl-get-view-of-pushbutton.

Example

The following sets the callback of the OK button associated with view-1 to the symbol my-callback:

```
call gxl-set-pushbutton-callback(view-1, “OK”, the symbol my-callback, win);
```

gxl-set-pushbutton-label

Sets the label of a pushbutton.

Synopsis

gxl-set-pushbutton-label
(*button*: class gxl-pushbutton, *label*: value, *fontsize*: symbol, *resize*:truth-value, *window:* class g2-window)

Argument	Description
button	The pushbutton for which you want to set a label.
label	The text of the label or the symbol default.
fontsize	The font size used for the label: small, large, extra-large.
resize	Determines how the *label* is positioned in the *button*.
window	The g2-window originating this call.

Description

This procedure sets the text of the *label* for a given *button*. If *resize* is true, it adjusts the button size, if necessary, to fit the text. If *resize* is false, it centers the label without changing the button size.

Example

The following example sets the label "Respond" to the pushbutton my-button so that the text fits within the button.

```
call gxl-set-pushbutton-label(my-button, "Respond", the symbol small, true, win);
```

gxl-wait-for-pushbuttons-on-view

Keeps a procedure in a wait state until the OK, Apply, or Cancel button is selected by the user on a view.

Synopsis

gxl-wait-for-pushbuttons-on-view
(*view*: class gxl-spreadsheet-view, *parameter*: class text-parameter,
window: class g2-window)

Argument	Description
view	The view that is the target of this call.
parameter	On exit, the value of this parameter is the ID of the button that was selected on the view.
window	The g2-window originating this call.

Description

This procedure can be used as a convenient way to wait inside a procedure for a user to finish working on a view. Normally, if you programmatically run an edit session, you might want to hold onto local variables in your procedure to use them to manage the end of the edit session.

Calling gxl-wait-for-pushbuttons-on-view puts your procedure into a wait state that ends when the user selects the OK, Cancel or Apply button on the view. The text parameter you provide contains the ID of the button (either "OK", "Apply" or "Cancel") when this procedure returns. If the view is deleted while waiting for gxl-wait-for-pushbuttons-on-view, it will return with the *parameter* "deleted".

Example

The following code fragment creates a view for the spreadsheet named spreadsheet-1, and then waits until a button is selected on the view:

```
create a text-parameter param;
view-1 = call gxl-create-and-display-spreadsheet-view (spreadsheet-1,
  specification-1, win);
call gxl-wait-for-pushbuttons-on-view (view-1, param, win);
inform the operator that "The Id of the button selected was [param]";
```

Part III

Appendix and Glossary

Appendix A: GXL Memory Requirements

Provides formulas for estimating the amount of memory required for spreadsheets and views of different dimensions.

Glossary

A

GXL Memory Requirements

Describes the memory requirements for GXL.

This Appendix provides formulas that you can use to make estimates of the amount of memory required for spreadsheet and view objects of different dimensions. The estimates given here are for GXL 5.0 Rev. 0, and are subject to change in future versions of GXL.

The main variable for spreadsheets in terms of memory requirements is the total number of cells in the spreadsheet and the type of value stored in the cells. If NCells is the number of rows times the number of columns in a spreadsheet storing floats, then the memory required on 32-bit machines is:

```
Bytes = 6000 + 24*NCells
```

Thus, the memory requirement for a spreadsheet with 1000 cells containing floats is approximately 30,000 bytes; for 10,000 cells, 246,000 bytes. More memory is required for storing symbols and text.

The memory requirement for views is primarily a function of the number of visible cells, which may be only a small fraction of the total cells in the spreadsheet. If VCells is the number of visible rows times the number of visible columns in a view, then the memory required on 32-bit machines is:

```
Bytes = 16000 + 480*VCells
```

This estimate includes the memory required for the accoutrements associated with a view, such as the toolbar, pushbuttons, etc. Thus, the memory requirement for a view with 100 visible cells is about 64,000 bytes.

Glossary

A

Application Programmers Interface: The set of procedures, methods, classes, and attributes designated as "public interface" items of GXL that enable programmatic creation and manipulation of GXL spreadsheet views within any G2 application.

C

cell: A cell is a location in the spreadsheet that contains data of a specific G2 value type. A cell can also be empty, and contain no value. A GXL spreadsheet can have any number of cells.

cell group specification: The cell group specification object defines the properties of a cell group. There are six types of cell group specifications: data cell group, row selector, column selector, column header, disabled group, and global selector.

cell group: A cell group is a rectangular section of the spreadsheet. Within a cell group, all cells have the same type. Within a spreadsheet, each cell group can contain a different value type and can potentially respond differently to user input.

column controller: The column controller is a specification object that specifies properties common to each cell group aligned vertically with it.

column header: The column header is a cell group specification object that is used to create a single cell that spans the entire width of the cell group. Usually, it is used to label one or more columns of a view.

column selector: The column selector is a cell group specification object that creates a cell which when selected, selects the entire column containing the cell. These cells are often used to number the columns of a spreadsheet.

D

data cell group: The data cell group is a specification object that specifies the type of data that its cells contain. All cells within a data cell group contain the same type of data: floats, integers, quantities, symbols, texts, truth-values, or values.

data type: Each data cell contains a specific G2 value type. The types of data cells are: value cells, which contain any G2 value type; quantity cells, which contain floats or integers; float cells, which contain floating point numbers; integer cells, which contain integers; symbol cells, which contain G2 symbols; truth value cells, which contain true or false; and text cells, which contain any text string.

disabled group: The disabled group is a specification object that is used to create cells that cannot contain data and are neither editable nor selectable. They are used to fill areas of the spreadsheet that do not contain data.

E

edit session: An edit session is a mode of operation in which changes to data are held in a buffer, enabling the user to cancel the changes, without affecting the original data.

F

float cell: A type of data cell that accepts and displays float values in decimal or exponential format.

focal cell: When entering a formula in GXL, the focal cell is the target cell whose value is to be calculated.

formula bar: The GXL formula bar, is a tool that allows you to calculate cell values, using mathematical operations, built-in G2 functions, or functions you define.

G

global selector: The global selector is a cell group specification object that is used to create cells which when selected, select the entire spreadsheet.

I

integer cell: A type of data cell that accepts and displays integer values.

Q

quantity cell: A type of data cell that accepts and displays float or integer values.

R

root specification: The GXL root specification object defines certain properties of the spreadsheet and spreadsheet view.

row-column indicator: The row-column indicator is a navigation aid which displays locations of the spreadsheet. The row-column indicator is a tool on the spreadsheet view that serves as a navigation aid by displaying corresponding locations in the spreadsheet.

row controller: The row controller is specification object that specifies properties common to the cell groups aligned horizontally with it.

row selector: The row selector is a cell group specification object that is used to create cells which when selected, selects the entire row containing the cell. Often, cells of this type are used to serially number the rows of a spreadsheet.

S

spreadsheet: The central object of GXL, the spreadsheet stores data as a two-dimensional grid. The data can be any combination of these G2 value types: floats, integers, quantities, symbols, text, and truth-values.

specification: A specification is a graphical layout of objects that serves as a template for the creation of spreadsheets and views. The layout of the specification objects, the classes of objects involved, and the attribute values of the objects define the specification.

spreadsheet view: The spreadsheet view displays the data contained in a GXL spreadsheet. Typically, a view displays only a subset of the cells in a spreadsheet.

symbol cell: A type of data cell that accepts and displays symbolic values.

T

text cell: A type of data cell tat accepts and displays text strings.

truth-value cell: A type of data cell that accepts and displays the values true or false.

V

value cell: A type of data cell that accepts and displays any valid G2 value.

view areas: View areas are horizontal and vertical areas superimposed on the data of a spreadsheet. If a cell is inside both a horizontal and a vertical view area, it is visible in the view.

Index

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V