

# G2 Diagnostic Assistant

User's Guide

Version 5.1 Rev. 0



G2 Diagnostic Assistant User's Guide, Version 5.1 Rev. 0  
November 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.

Copyright (c) 1985-2015 Gensym Corporation

All rights reserved. No part of this document may be reproduced, stored in a retrieval system, translated, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Gensym Corporation.

Gensym®, G2®, Optegrity®, and ReThink® are registered trademarks of Gensym Corporation.

NeurOn-Line™, Dynamic Scheduling™, G2 Real-Time Expert System™, G2 ActiveXLink™, G2 BeanBuilder™, G2 CORBALink™, G2 Diagnostic Assistant™, G2 Gateway™, G2 GUIDE™, G2GL™, G2 JavaLink™, G2 ProTools™, GDA™, GFI™, GSI™, ICP™, Integrity™, and SymCure™ are trademarks of Gensym Corporation.

Telewindows is a trademark or registered trademark of Microsoft Corporation in the United States and/or other countries. Telewindows is used by Gensym Corporation under license from owner.

This software is based in part on the work of the Independent JPEG Group.

Copyright (c) 1998-2002 Daniel Veillard. All Rights Reserved.

SCOR® is a registered trademark of PRTM.

License for Scintilla and SciTE, Copyright 1998-2003 by Neil Hodgson, All Rights Reserved.

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.openssl.org/>).

All other products or services mentioned in this document are identified by the trademarks or service marks of their respective companies or organizations, and Gensym Corporation disclaims any responsibility for specifying which marks are owned by which companies or organizations.

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

Part Number: DOC114-510

# Contents

---

<b>Preface</b>	<b>xiii</b>	
About this Guide	xiii	
Audience	xiv	
Conventions	xiv	
Related Documentation	xvi	
Customer Support Services	xviii	
<b>Chapter 1</b>	<b>Basics of Using GDA</b>	<b>1</b>
Introduction	2	
Modes	2	
Planning the Application	3	
Using Modules	3	
The Phases of an Application	4	
Overview of Palettes and Blocks	5	
Starting to Use GDA	5	
Loading GDA the First Time	5	
Renaming the Top-Level Module	7	
Saving an Application	7	
Starting GDA with Your Application	8	
Using System Tables	9	
Parameters GDA Does Not Check at Startup	9	
Timing Parameters	9	
Color Parameters	9	
Font Parameters	9	
Menu Parameters	10	
Miscellaneous Parameters	10	
Parameters GDA Checks at Startup	10	
Timing Parameters	10	
Drawing Parameters	11	
Building a Diagram	11	
Some Important Practices	11	
Don't Edit a Diagram While GDA Is Reset	11	
Make Sure To Enable Data Input	11	

Save the Application Periodically	11
Creating Application Workspaces	12
Cloning Blocks	13
Cloning a Block From the Palette	14
Searching for a Particular Block	14
Cloning an Existing Block	15
Configuring Block Attributes	16
Configuring Attributes in the Configuration Dialog Box	16
Deleting Blocks	18
Connecting Blocks	18
Connecting to Peer Input Blocks	20
Deleting Paths and Stubs	21
Connecting One Output to Two Blocks	21
Rules for Connecting Blocks	22
Improving the Appearance of a Diagram	24
Aligning Blocks Using a Snap Grid	24
Creating Vertices in Path Connections	25
Labeling Diagrams	26
Editing Attribute Displays	27
Displaying a Block's Table	28
Displaying the Value on a Path	28
Displaying the Path's Table	29
Adding a Path Display	30
Running an Application	30
Running Your Own Procedure When You Start G2	31
Controlling the Flow of Data in an Application	32
Enabling and Disabling Data Input	32
Inhibiting Data Flow Through a Block	33
Enabling and Disabling Evaluation of a Block	33
Toggling Animation	34
Allowing Other Processing	35
Setting the Maximum Timeout for Data Seeking	35
Connecting to Remote Processes	35
Customizing Menus	36
Using Menu Preferences	36
Creating New Menu Preference Objects	38
How GDA Manages the Default Menu Preferences	39
Extending Main Menu Choices	39
Translating Menu Choices	40
Miscellaneous Features	40
Setting Configuration Panel Colors	40
Setting the Color for Titles, Type-in Boxes, Text, and Scroll Messages	42
Setting the Color for Display Items Within Configuration Panels	42
Setting Network Colors	44

## **Chapter 2 Using Blocks and Paths 47**

Introduction 48

Basic Block Behavior 48

Reading Notes and Errors 48

Adding Comments to a Block 49

Resetting Blocks 50

Evaluating Blocks 51

Overriding Block Values 51

Overriding a Data Block 54

Overriding a Control Block 54

Overriding a Discrete Inference Block 54

Overriding a Fuzzy Inference Block 55

Locking and Unlocking Blocks 55

Enabling and Disabling Evaluation 56

Clearing Block Errors 57

Setting Block Colors 57

Setting Alarm Colors 59

Using Paths 60

Using Data Paths 61

Using Inference Paths 62

Filtering Data Passed to Inference Paths 62

Using Control Paths 63

Using Item Paths 63

Creating Item Paths 63

Placing an Item Onto an Item Path Interactively 63

Placing an Item Onto an Item Path Programmatically 64

Displaying the Item on the Path 65

Resetting Paths 65

Using Links 65

Using Connection Posts 66

Highlighting Paths and Connection Posts 68

Setting Path Colors 68

Using Path Attributes 70

The Quality Attribute 70

The Timestamp Attribute 71

The Collection-Time Attribute 71

The Expiration-Time Attribute 72

Specifying Validity Interval for a Variable 72

Determining How Blocks Use no-value Inputs 74

Determining Output Path Attributes for Peer Input Blocks 75

Determining Whether a Block Uses Expired Inputs 75

Example of Determining Path Attributes Using a Peer Input Block 76

Creating Customized Path Connections 78

Creating a New Connection Subclass	78
Customizing the Connection Path Regions	81
Adding Custom Connections to a Custom Subclass	81
Specifying Initial Values	82
Specifying an Initial Data Value	82
Specifying an Initial Control Value	82
Specifying an Initial Status Value	83
Maintaining a History of Values	83
How the History Feature Works	84
Specifying the Size of the History	84
How GDA Handles Point-Based Histories	85
How GDA Handles Time-Based Histories	85
How GDA Handles Histories When the Sample Time is Fixed	86
Performance Issues	86
Deciding Which Sample Time Option to Choose	86
Specifying When to Propagate Data	87
How GDA Handles Point-Based Updates	87
How GDA Handles Time-Based Updates	88
Performance Issues	88
Specifying What Happens to History Upon Reset	89
Specifying What to Do With Partial History	90
How GDA Handles Nonmonotonic Values	91
Specifying How to Handle Multiple Values	91
Specifying and Generating Explanations	92
Specifying an Explanation	93
How to Describe Why a Block Passes True, False, or Unknown	93
Describing the Block's Input Value	94
Generating Explanations	95
Generating Explanations for Most Blocks	97
Creating a Description for Logic Gates	98
Specifying Fuzzy Logic Attributes	100
Specifying the Type of Logic to Use	100
Specifying Uncertainty	101
Specifying Hysteresis	102
Using Variables and Parameters	104
Choosing the Type of Variable or Parameter	104
Creating a Variable or Parameter	105
Using Variables to Connect to External Data	105
Creating a Sensor	106
Connecting a Variable or Parameter to a Block	107
Overriding Values of Variables and Parameters	108
Coercing Data Using Variables and Parameters as Input	109
Examples of Data Coercion Using Variables and Parameters as Input	111

	Coercing Data Using Variables and Parameters as Output	113
	Examples of Data Coercion Using Variables and Parameters as Output	115
	Evaluating Expressions in Attributes	117
	Using a G2 Expression	119
	Using a G2 Function or Procedure	119
	Examples	120
	Using the GXL Spreadsheet to Edit Data	123
	Understanding the GDA Block Evaluation Engine	125
	Invoking an Individual Block	125
	Executing Diagrams	126
	Executing Blocks that Have Been Manually Evaluated	126
	Executing Blocks that Have Been Launched by the System	126
	Sweeping Diagrams with Entry Points	127
	Setting the Sweep Interval	127
	Evaluating Blocks on Individual Workspaces	128
<b>Chapter 3</b>	<b>Using GDA Queues</b>	<b>131</b>
	Introduction	131
	Summary of the Queue Features	132
	Using the Alarm Queue	134
	Filtering Queue Entries	134
	Permanent and Temporary Filters	135
	Creating a Permanent Filter	135
	Entry Attributes Used with Filters	137
	Creating a Temporary Filter	139
	Making a Temporary Filter Permanent	144
	Applying a Filter	145
	Deleting a Filter	146
	Sorting Entries	147
	Sending Entries to Another Queue	147
	Viewing the Details of an Entry	148
	Displaying the Entry Message Text	150
	Adding and Viewing Comments	150
	Providing and Viewing Advice for Alarms	151
	Viewing an Alarm Explanation	153
	Viewing the Alarm History	153
	Saving Details to a File	154
	Showing the Source of an Entry	155
	Acknowledging Alarms	156
	Saving a Queue Entry	159
	Removing Entries from the Queue View	160
	Locking the Alarm Queue View	160

Using the Error Queue 162  
Using the Explanation Queue 163  
Using the Message Queue 164  
Sending Messages to Queues Using the Queue Message Block 165

## **Chapter 4 Custom Block Wizard 167**

Introduction 167  
Using the Custom Class Wizard 168  
Creating a New Custom Subclass 169  
    Specifying the Class Name 171  
    Customizing the Icon 172  
    Customizing the Connection Stubs 173  
    Customizing the Attributes 175  
    Specifying the Palette and Module 177  
    Applying the New Class Definition 178  
    Cloning and Configuring the Custom Block 179  
Customizing the Block Evaluator 180  
    Displaying the Block Evaluator 180  
    Types of Custom Block Evaluators 182  
    Editing the Block Evaluator Procedure 183  
    Example Using the power-block Custom Block 185  
    Declaring the Procedure Name and Arguments 185  
    Declaring Input and Output Path Local Names 186  
        Local Names for the General and Multiple Invocations Custom  
        Blocks 186  
        Local Names for the Peer Input Custom Blocks 187  
    Obtaining Input Path Values 187  
    Determining Output Path Attributes for Custom Blocks 188  
    Editing the Custom Portion of the Block Evaluator 189  
    Setting Output Path Values 189  
Editing an Existing Custom Subclass 190  
Deleting an Existing Custom Subclass 191  
Custom Class Reference 193  
    General 194  
    Peer Input 195  
        Creating Connections for Custom Peer Input Blocks 195  
        Customizing the Peer Input Custom Block Evaluator 195  
    Configuring 196  
    Multiple Invocations 200  
        Determining How the Block Handles Multiple Control Signals 200  
    Single Source Encapsulation 206



Creating a New Subclass of SSE Block	206
The Master Diagram and the Local Diagram	208
Editing the Master Diagram	210
Displaying the Master Diagram	210
Editing the Blocks on the Master Diagram	211
Using Rule Terminals on Single Source Encapsulation Blocks	211
Saving the Master Diagram	212
Saving the Master Diagram	212
Cancelling Editing the Master Diagram	212
Saving the Master Diagram Locally	213
Creating Instances of a Single Source Encapsulation	213
Viewing the Local Diagram	213
Changing the Value of an Attribute	214
Configuring Attributes in the Master Diagram	215

## **Chapter 5 Creating and Configuring Queues 217**

Introduction	217
Attributes of Queues You Can Modify	218
Creating a New Queue	219
Configuring a Queue	221
Configuring Attributes that Handle New Entries	223
Beep for New Entry	224
Default Priority	224
Display Messages	224
Entry Class	224
Item Addition Callback	224
Configuring Attributes that Handle Changes and Deletions to Entries	225
Attribute Update Callback	225
Confirm Deletions	225
Item Removal Callback	226
Configuring Attributes that Handle Queue Capacity	226
Entry Lifetime	226
Entry Limit	227
Configuring Attributes that Are Specific to Alarm Queues	227
Alarm Log Formatter	228
Autogenerate Explanations	228
History Limit	228
Recurring Entry Class	228
Reuse Entry	229
Using Tracebacks for Alarms and Errors	229
Using a New Queue	229

Logging Queue Entries	229
Enabling and Disabling Alarm Logging	230
Providing a Name and Location for the Log File	230
When GDA Creates a New Log File	231
Determining the Log File Header and Message Contents	231
The Log File Name	231
The Log File Header	231
Alarm Log Entries	232
Error Log Entries	233
Explanation Log Entries	233
Message Log Entries	233
Customizing the Entry	233
Incremental Logging of Alarm Entries	233
The Time Format for Alarm Entries	234

## **Chapter 6    Creating Queue Views    235**

Introduction	235
Characteristics of the Built-in Queue Views	236
Creating a New Queue View Template	240
Configuring a Queue View Template	242
Configuring a Built-in Queue View	243
Configuring a New Queue View	243
Modifying Queue View Attributes on the Configure Dialog	244
Configuring the Lines per Row Attribute	244
Configuring the Font Size Attribute	245
Configuring Sorting Attributes	245
Configuring the Layout of a Queue View	245
Modifying the Queue View Label	246
Modifying Queue View Colors	247
Manipulating Toolbar Buttons	248
Deleting Toolbar Buttons	248
Moving Toolbar Buttons	248
Adding Toolbar Buttons	249
Modifying Button Attributes	251
Creating and Customizing Buttons	252
Queue View Button Classes	252
Detail View Button Classes	253
Modifying the Queue Entry Counters	253
Modifying Columns	254
Specifying the Columns that Appear in the View	255
Moving Columns in the View	255
Changing the Number of Rows Visible in the Queue View	255
Changing the Font Size of the Text Displayed in a Column	256
Changing the Column Color	256

Changing the Height and Width of Columns	256
Specifying Whether Clicking on a Column Header Sorts Entries	256
Controlling the Type of Data Displayed in Columns	257
Controlling the Format of Floating Point Numbers in a Column	257
Determining Whether a User Can Edit Data in a Column	257
Controlling Whether a Column Cell Can Be Selected	258
Formatting the Contents of a Column	258
Configuring Date and Time Formatting	259
Configuring Ordination	261
Default Alarm Queue Color Formatter Procedures	263
Modifying Column Headers	264
Specifying the Attribute to Display in the Column	264
Modifying the Header Label Text	264
Modifying the Font Size of the Label	265
Modifying Colors of the Label	265
Monitoring a Column Value	265
Configuring the Detail View	266
Configuring the Detail View Template	266
Displaying the Default Detail View Templates	268
Configuring the Detail View Colors	269
Configuring the Detail View Size	270
Configuring the Detail View Label	270
Creating Your Own Detail View Template	270
Associating the Detail View with the View Details Button	271
Configuring the Detail View Position and Scale	271
Creating and Configuring the Access Manager	273
How the Access Manager Works	274
Specifying the Queue View Template or Access Manager	275
Creating an Access Manager	276
Configuring an Access Manager	277
The Access Manager Toolbar Buttons	278

**Glossary 281**

**Index 289**



# Preface

---

*Describes this manual and the conventions that it uses.*

About this Guide	xiii
Audience	xiv
Conventions	xiv
Related Documentation	xvi
Customer Support Services	xviii



## About this Guide

G2 Diagnostic Assistant (GDA), is an environment for developing and running intelligent operator applications. Its principal component is a graphical language that lets you express complex diagnostic procedures as a diagram of blocks, also called an Information Flow Diagram (IFD). These blocks are connected by paths that show how data flows through the diagram.

This guide provides general information about how to use GDA. It assumes you are familiar with G2. For information on the behavior of each specific GDA block, see the *GDA Reference Manual*.

This user's guide has these chapters:

<b>This chapter...</b>	<b>Describes...</b>
<a href="#">Basics of Using GDA</a>	How to perform basic operations, such as using the menus, setting up an application, creating a diagram, and running an application.
<a href="#">Using Blocks and Paths</a>	Basic features of GDA blocks and paths.

<b>This chapter...</b>	<b>Describes...</b>
<a href="#">Using GDA Queues</a>	How to use the queues, on which GDA displays its alarms, messages, explanations, and error messages.
<a href="#">Custom Block Wizard</a>	How to use the GDA wizard to create custom subclasses of GDA blocks.
<a href="#">Creating and Configuring Queues</a>	How to create and configure GDA queues.
<a href="#">Creating Queue Views</a>	How to create queue views, which provide access to queue entries.

## Audience

This book is intended to be used primarily by programmers using GDA to develop end-user applications. You should be familiar with G2.

## Conventions

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

### Typographic

<b>Convention Examples</b>	<b>Description</b>
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

Convention Examples	Description
conclude that the x of y ...	Text of G2 procedures, methods, functions, formulas, and expressions
<i>new-argument</i>	User-specified values in syntax descriptions
<u><i>text-string</i></u>	Return values of G2 procedures and methods in syntax descriptions
File Name, OK, Apply, Cancel, General, Edit Scroll Area	GUIDE and native dialog fields, button labels, tabs, and titles
File > Save	GMS and native menu choices
Properties	
<b>workspace</b>	Glossary terms
<i>c:\Program Files\Gensym\</i>	Windows pathnames
<i>/usr/gensym/g2/kbs</i>	UNIX pathnames
<i>spreadsh.kb</i>	File names
<i>g2 -kb top.kb</i>	Operating system commands
<i>public void main() gsi_start</i>	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? Guide*
- *G2 System Procedures Reference Manual*
- *G2 System Procedures Reference Card*
- *G2 Class Reference Manual*
- *Telewindows User? Guide*
- *G2 Gateway Bridge Developer? Guide*

## **G2 Utilities**

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

## **G2 Developers' Utilities**

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



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

### **Bridges and External Systems**

- *G2 ActiveXLink User? Guide*
- *G2 CORBALink User? Guide*
- *G2 Database Bridge User? Guide*
- *G2-ODBC Bridge Release Notes*
- *G2-Oracle Bridge Release Notes*
- *G2-Sybase Bridge Release Notes*
- *G2 JMail Bridge User? Guide*
- *G2 Java Socket Manager User? Guide*
- *G2 JMSLink User? Guide*
- *G2-OPC Client Bridge User? Guide*
- *G2 PI Bridge User? Guide*
- *G2-SNMP Bridge User? Guide*
- *G2 CORBALink User? Guide*
- *G2 WebLink User? Guide*

### **G2 JavaLink**

- *G2 JavaLink User? Guide*
- *G2 DownloadInterfaces User? Guide*
- *G2 Bean Builder User? Guide*

## G2 Diagnostic Assistant

- *GDA User? 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](http://www.gensym-support.com).

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

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

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

➔ Use the following numbers and addresses:

	<b>Americas</b>	<b>Europe, Middle-East, Africa (EMEA)</b>
<b>Phone</b>	(781) 265-7301	+31-71-5682622
<b>Fax</b>	(781) 265-7255	+31-71-5682621
<b>Email</b>	<a href="mailto:service@gensym.com">service@gensym.com</a>	<a href="mailto:service-ema@gensym.com">service-ema@gensym.com</a>

# Basics of Using GDA

---

*Describes how to perform the basic operations in GDA.*

Introduction	2
Modes	2
Planning the Application	3
Starting to Use GDA	5
Using System Tables	9
Building a Diagram	11
Running an Application	30
Customizing Menus	36
Miscellaneous Features	40



## Introduction

GDA, a layered product built on top of G2, is a visual programming environment for developing applications that monitor and control real-time processes.

A GDA application contains schematic diagrams that:

- **Acquire data** from real-time processes
- **Make inferences** based on the data
- **Take actions** based on the inference values, such as raising alarms, sending messages to operators, or concluding new setpoints

You create a GDA diagram by cloning blocks from palettes and by connecting them together. You can then configure the attributes of the blocks to control their behavior.

Once you have created diagrams of your process, you run them to observe their behavior and test your application.

## Modes

GDA provides three operational modes: Administrator mode, Developer mode, and User mode. While developing your application, you use both Developer mode and Administrator mode. The end user runs the application in User mode or in a mode you create.

---

**Note** You can change modes by selecting Mode from the Gensym menu and choosing a mode. You can also change between Developer mode and Administrator mode by entering Ctrl-Y.

---

- **Developer mode** enables you to create and define objects and other elements of the application. You can also run the application in this mode, clicking on buttons to open workspaces or run procedures associated with the buttons.
- **Administrator mode** provides more complete access to all objects in the diagram, giving you options not available in Developer mode. Generally, you use these options infrequently, although they are important in being able to fully describe your application. For example, in Developer mode, if you click on the label of a subworkspace button, GDA opens the subworkspace. In Administrator mode, clicking on the button enables you to edit its label.
- **User mode** provides restricted access to menus, limiting the user's ability to change the diagram. This mode enables the user to acknowledge alarms and lock and override block values.

You can define user modes for your application. See the "Configurations" chapter in the *G2 Reference Manual*, for more information on configuring user modes.

## Planning the Application

It is very important that you carefully plan the application, which involves:

- Studying your process and identifying its components
- Categorizing the components to create object classes
- Describing the behavior of each component so you can identify attributes and develop methods for each object class
- Listing the rules that govern the behavior of your system

- Identifying the components whose behavior is critical for the smooth functioning of your process and describing the conditions that should cause alarms
- Creating modules

## Using Modules

All but the simplest KBs should consist of multiple modules. Developing an application using multiple, small modules offers many advantages, including:

- It enables developers to divide and merge work.
- It results in potentially reusable modules.

GDA itself is modularized, consisting of a number of modules.

When you develop your application, you begin in the top-level module, called `gdaapps`. When you save an application, you save it to another file name.

Other advantages, as well as a more complete discussion of using modules, are contained in the *G2 Developer? Guide*, an essential manual for the GDA developer interested in using modules.

For more information about creating, populating, and saving a module, see the chapter on “Modules and Modularized KBs” in the *G2 Reference Manual*.

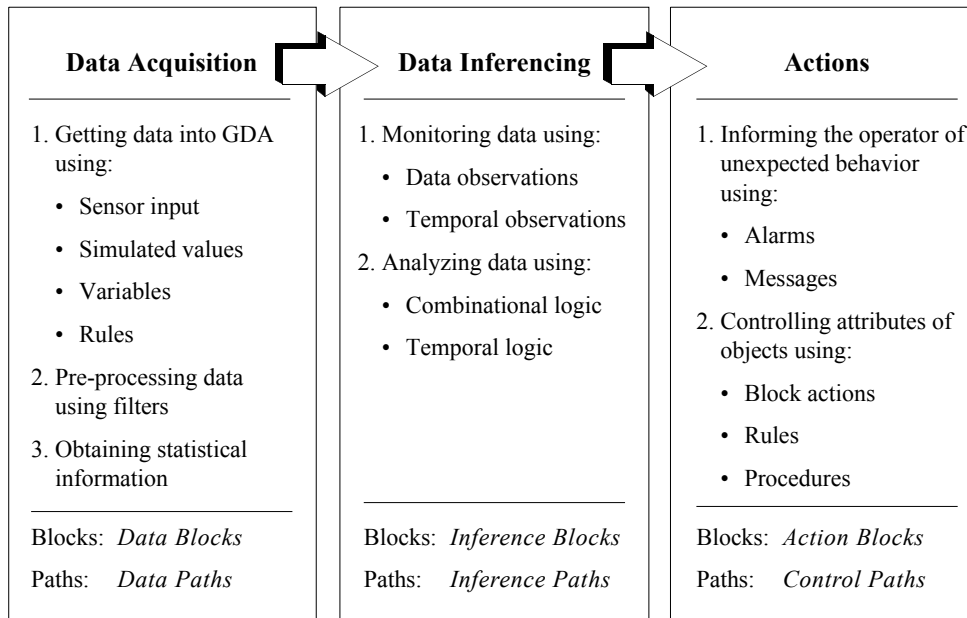
## The Phases of an Application

A GDA diagram consists of blocks, which process data, and paths, which pass different types of data through the diagram for monitoring and analysis. A typical GDA diagram does the following:

- **Acquires data**, filters the data, and performs operations on the data, such as arithmetic or SPC operations. This part of the diagram uses data blocks and data paths.
- **Makes inferences** about the data and performs logic operations on the inference values. This part of the diagram uses inference blocks and inference paths.
- **Performs actions** based on the inference values. This part of the diagram uses action blocks and control paths.

In addition, a GDA diagram can perform a number of special purpose functions, such as creating alarms when certain conditions are met, charting your data, incorporating G2 rules into your diagram, and networking your application.

The following figure outlines the general features of each segment of the diagram:



## Overview of Palettes and Blocks

The GDA palettes, accessible from the **Palettes** menu, are divided into three categories, which correspond to the three basic parts of the diagram:

- **Data blocks**, which operate on numeric, textual, or symbolic values
- **Inference blocks**, which derive truth values from data values, or which operate on truth values (including fuzzy values)
- **Action blocks**, which perform actions on other blocks or on the environment

Each palette is described in detail in its own chapter in the *GDA Reference Manual*. The common features of all the blocks on a palette are described at the beginning of the chapter, and the blocks are described within each chapter.

## Starting to Use GDA

To start developing a GDA application, first, you start G2, then you load *gdaapps.kb*. This top-level KB requires a number of G2 utility KBs, which are located in the *g2\kbs\utils* directory on Windows and in the *g2/kbs/utils* directory on UNIX in your G2 installation directory.

To run GDA, you must use a module search path to point to the required utility KBs. One way to specify the module search path is by using the *-module-search-path* command-line option to start G2. When you start G2 from the shortcut on the Start menu on Windows platforms, the *-module-search-path* argument

appears in the shortcut, which you can modify, as needed. You can also configure the `module-search-path` attribute in the Server Parameters system table to make the module search path a permanent part of your application.

For details on specifying the module search path, see the *G2 Reference Manual*.

## Loading GDA the First Time

### To load GDA the first time:

- 1 Start G2, using the `-module-search-path` command-line option, specified as a string to find components required by NOL application.

The required and optional components for NOL application are:

- G2 utility
- NOL modules
- GDA basic modules
- G2i modules
- Optional G2 bridges, such as G2 OPCLink, G2 JavaLink, and G2-PI bridge.

For example, from a command window or shell:

```
Windows    g2 -module-search-path "'.\| '..\nol'
              '..\g2\kbs\utils' '..\gda' '..\g2i\kbs'
              '..\opclink\kbs' '..\pi' '..\jmail\kbs'
              '..\jms\kbs' '..\| '..\javalink\KBs'
              '..\protocols\kbs' '..\activexlink' '..\gsi'
              '..\gw\kbs' '..\tw2\kbs' '..\odbc'"""
```

---

```
UNIX       g2 -module-search-path " './| '..\nol'
              './g2/kbs/utils' './gda' './g2i/kbs'
              './opclink/kbs' './pi' './jmail/kbs'
              './jms/kbs' './| './javalink/KBs'
              './protocols/kbs' './activexlink'
              './gsi' './gw/kbs' './tw2/kbs'
              './odbc'"
```

For example, from a command window:

```
g2 -module-search-path "'.\| '..\nol'
  '..\g2\kbs\utils' '..\gda' '..\g2i\kbs'
  '..\opclink\kbs' '..\pi' '..\jmail\kbs'
  '..\jms\kbs' '..\| '..\javalink\KBs'
  '..\protocols\kbs' '..\activexlink' '..\gsi'
  '..\gw\kbs' '..\tw2\kbs' '..\odbc'"""
```

---

**Note** You must surround the module search path name with double quotes. If the directory contains spaces, you must surround the path name with single quotes as well.

---

- 2 Select Main Menu > Load KB.
- 3 Enter the complete pathname of the *gdaapps* KB, including double quotes.

---

**Tip** If you do not know the pathname of the file you want to load, enter the name of a directory and press Return to display a list of the contents of the directory. Select a subdirectory or filename from the list that appears and press Return again until you have located the desired file.

---

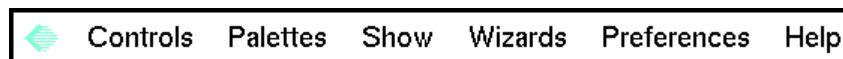
For example, the pathname might look something like this on Windows:

"C:\Program Files\Gensym\g2-8.3r0\gda\gdaapps.kb"

G2 reports its progress and displays a message in the Operator Logbook when each module is finished loading.

- 4 Select Main Menu > Start.

When you start GDA, the top-level menu bar appears:



If you do not see the GDA top-level menu and you issued the Start command, you may be running GDA with other Gensym products. Select the Gensym menu and choose Available Menus > GDA. The GDA menu should appear.

## Renaming the Top-Level Module

When you initially load GDA, you load the knowledge base contained in the file *gdaapps.kb*. This creates an empty top-level module named *gdaapps*, which loads all the required GDA modules.

Typically, you rename the top-level module for a specific application. When you rename the top-level module, you must also reassign any top-level workspaces to the new module.

### To rename the top-level module and reassign its top-level workspaces:

- 1 Load the empty KB contained in the file *gdaapps.kb* located in the GDA directory, as described in [Loading GDA the First Time](#).
- 2 Load the empty KB contained in file *gdaapps.kb* located in the GDA directory.



- 3 Select Main Menu > Inspect to display the G2 editor.
- 4 Enter show on a workspace the module hierarchy in the editor.  
This shows a workspace containing the module hierarchy for the GDA application. The top-level module is called `gdaapps`.
- 5 Click on the top-level module and edit its name.
- 6 Enter a new name for the top-level module that describes your application.
- 7 Select Main Menu > Inspect again and enter show on a workspace every kb-workspace assigned to module `gdaapps`.  
You will see an object representation labeled a kb-workspace.
- 8 Choose `table` on the object that represents a kb-workspace and edit the Module-assignment attribute to refer to the new top-level module name.

You load this new top-level KB the next time you want to use your application.

## Saving an Application

To save an application, you save the top-level module to a `.kb` file. The default file name for an application matches the name of the top-level module. We recommend that you always save your application to a file name that matches the top-level module. The required, lower-level modules exist as separate KBs. You never need to save GDA and its supporting modules.

Before you can save your application, you must ensure that all top-level workspaces are assigned to the new top-level module; otherwise, you cannot save your application as a modularized KB.

If you save an application to a file that does not have the same as the top-level module, you can create a module map to associate a module name with a particular KB file. For more information about modules and modularized KBs, see the *G2 Reference Manual*.

Saving the top-level module automatically saves your configuration as well; all customizations to the colors, settings, and queues are automatically saved in the KB file. See [Customizing Menus](#).

### To save the top-level module:

- 1 Select Main Menu > Save KB.  
G2 prompts you to save the top-level module, using the name you specified. The default file name has the same name as the top-level module.
- 2 Click End or press Return.

G2 reports progress while saving and displays a message in the Operator Logbook when the save is complete.

## Starting GDA with Your Application

Once you have configured the module search path, renamed the top-level module, and saved your application, you can start GDA with your application. You can either load the application manually or you can use a command-line option to load the KB when G2 starts up.

### To start GDA with your application:

- 1 Start G2.
- 2 Select Main Menu > Load KB and specify your top-level module KB.

### To load your application automatically:

➔ Start G2, using the `-kb` command-line option, specified as a string.

For example, from a command window or shell:

---

<b>Windows</b>	<code>g2 -kb "c:\Program Files\Gensym\ g2-8.3r0\gda\myapp.kb"</code>
----------------	--

---

<b>UNIX</b>	<code>g2 -kb "/usr/gensym/g2-8.3r0\gda/myapp.kb"</code>
-------------	---

---

**Tip** You may need to include the module search path as one of the command-line options as described above.

---

## Using System Tables

Certain system table parameters enable GDA to run more efficiently, improve the appearance and convenience of running GDA applications, or run without encountering a particular error.

These parameters are set in `gdaapps`. If an application is not developed using `gdaapps`, or if you have changed parameter values, these parameters may not be set properly. GDA checks for some of the parameter values at startup. For these parameters, if the values differ from the expected values, GDA displays a message that advises that the value be changed.

### Parameters GDA Does Not Check at Startup

GDA does not check the value of these parameters at startup. The description of each parameter indicates what happens if the value is different than that set in `gdaapps`.

## Timing Parameters

The `Uninterrupted-procedure-execution-limit` parameter sets a limit on the amount of consecutive execution time a procedure can use. Particularly during startup and initialization, some GDA processes may run longer than the G2 limit. This parameter is set to 1 minute to avoid procedure aborts due to exceeding the limit value. You can increase this parameter further if, for large applications, the timeouts continue to occur.

## Color Parameters

The `Color-on-1st-level-color-menu` parameter specifies which colors appear on the first level menu. A setting of `all` puts all the colors on the color palette. Also, a setting of `3` for the `Number-of-columns-on-1st-level-color-menu` provides an easy to use palette display.

## Font Parameters

The `Font-for-attribute-tables` parameter determines which font G2 uses for text in attribute tables. A value of `small` corresponds to the setting that GDA was designed around. Making the attribute table text larger may have unintended layout effects.

## Menu Parameters

The `When-to-allow-multiple-menus` parameter determines whether you can display on a workspace more than one menu, or more than one copy of the same menu at a time. In `gdaapps`, a value of `for different selections` is used, which displays more than one menu at a time if the menus are for different items.

## Miscellaneous Parameters

The `Initial-margin-for-workspaces` parameter specifies how close to the edge of a workspace you can place objects. Developing in GDA often results in the creation of large workspaces. A setting of `10` (the default is `30`) narrows the margins on workspaces and reduces the amount of the screen used.

The `Backward-compatibility-features` parameter allows you to revert certain changes made in G2 since previous versions. The `Ignore-duplicate-list-element-error` value causes G2 to disregard a change made to the `insert` action, which signals an error when inserting a duplicate element into a list that disallows duplicates. If the parameter value does not include `Ignore-duplicate-list-element-error`, GDA will display errors and could prevent GDA from working properly under certain circumstances (for example, alarms might not reset).

The `Connection-caching-enabled?` parameter enables expressions that reference connections to execute faster, but slows down the process of changing connections. A value of `yes` enables connection caching and improves GDA's execution.

## Parameters GDA Checks at Startup

GDA checks the values of these parameters when starting up an application. If the value is different than the value set by `gdaapps`, GDA displays a message that suggests that you change the value. The description of each parameter discusses the reason for the parameter value.

### Timing Parameters

The `Scheduling-mode` parameter defines the timing mode of the scheduler and how tasks are scheduled. If the scheduler mode is `simulated time` at startup, GDA will prompt you to change the value. If you choose to change the value, GDA sets it to `real time`.

The `Minimum-scheduling-interval` parameter specifies the length of time for a clock tick, which determines how long the scheduler has to perform tasks between clock ticks. The value is set to `0.05`, a setting that is necessary for GDA's own timing parameters to work properly.

The `Milliseconds-to-sleep-when-idle` parameter only affects VMS platforms. It controls the response of VMS to graphical events. A value of `10` generally works well.

### Drawing Parameters

The `Allow-scheduled-drawing?` parameter specifies whether drawing is a scheduled task. The value `yes` (the G2 default) selects scheduled drawing, which is more efficient than immediate mode.

The `Paint-mode?` parameter specifies whether the Paint or the XOR drawing mode is in effect. A value of `yes` specifies paint mode. This ensures that the layered graphics in GDA appear as intended.

## Building a Diagram

This section discusses important concepts and tasks you use when building a diagram.

### Some Important Practices

As you build a GDA diagram, you should be aware of these important practices:

- Don't edit a diagram while GDA is reset.
- Make sure to enable data input.
- Save the application periodically to protect your work.

## Don't Edit a Diagram While GDA Is Reset

Do not attempt to edit a diagram after you use the Main Menu > Reset menu choice; if you do, the results will be unpredictable. Before adding, deleting, or modifying blocks on a workspace when GDA is reset, you *must* start G2 using Main Menu > Start.

## Make Sure To Enable Data Input

When you run a GDA application, data input is disabled, preventing data from entering the application through entry points, or being generated by Signal Generator or Clock blocks. To ensure that data flows into the application, make sure you select Controls > Enable Data Input.

## Save the Application Periodically

To protect your work, save the application periodically. As with any programming or application development software, you should acquire the habit of saving your work regularly to protect it against hardware or software problems.

Also, you should save your work before making potentially damaging changes so you can return to that version of your application. G2 does not provide the capability of undoing deletions or changes automatically so you should be careful when deleting objects that might require a substantial amount of effort to re-create. GDA does display a confirmation dialog box when you delete an object, which provides you the opportunity to cancel the deletion.

This manual discusses saving the application later in this chapter. See [Saving an Application](#).

## Creating Application Workspaces

Most applications have numerous workspaces, organized in a hierarchy. Typically, an application consists of a single top-level workspace, which contains navigation buttons that provide access to subworkspaces. These subworkspaces contain the diagrams and supporting definitions and programmatic elements that together make up the application.

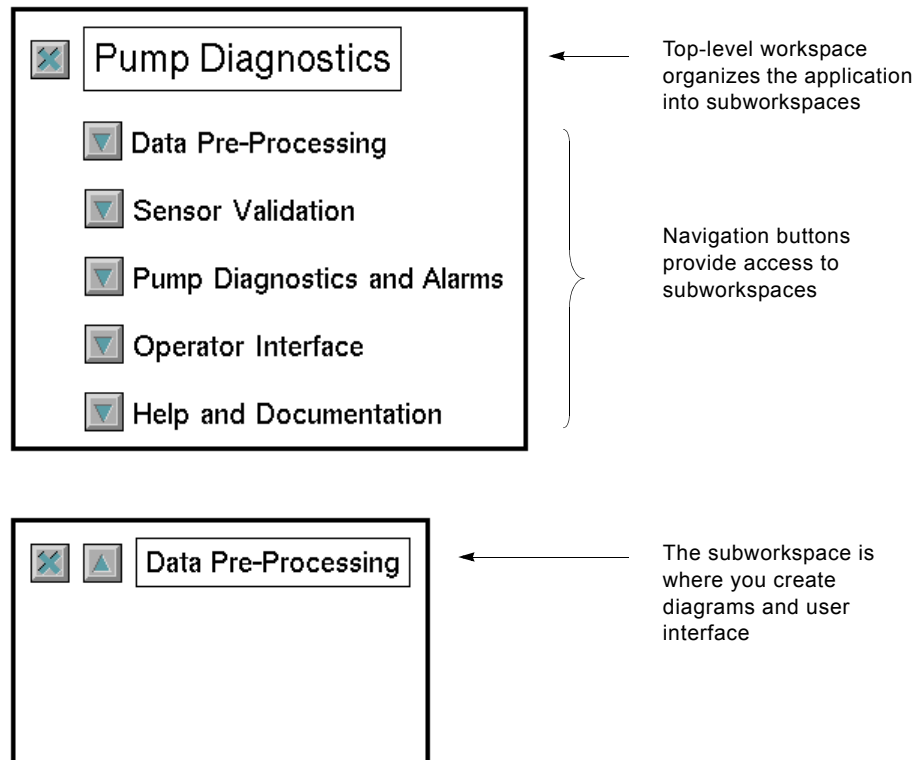
**To set up the workspace hierarchy:**

- 1 Create a new workspace by selecting Main Menu > New Workspace.
- 2 Click on the new workspace and select Setup Application Workspace to convert the workspace to an application workspace. GDA displays this workspace:



- 3 Click on the text labeled Enter title here, select the edit menu choice, enter a new title, and press Return.
- 4 Create a top-level menu choice to provide access to this workspace by clicking on the title and choosing add to menu. GDA adds a menu choice to the Applications menu; the name of the menu choice is the same as the title you added in the previous step.
- 5 Click on the background of the workspace and select Add Subworkspace Button to add subworkspaces for your application. Repeat this step to add additional subworkspaces.
- 6 Go into Administrator mode.
- 7 Click on the text of each subworkspace button, and select edit to edit the name of the subworkspace.
- 8 When you have created labels for all your subworkspaces, return to Developer mode.
- 9 Click on a subworkspace button to display its subworkspace.
- 10 Edit the title of the subworkspace to match the subworkspace button (if you want the subworkspace to have a label that matches the text of the button).

For example, here is the top-level workspace for a Pump Diagnostics application and the Data Pre-Processing subworkspace, which is initially empty:




---

**Note** If you delete a top-level workspace whose menu choice has been added to the Applications menu, GDA automatically removes the associated menu choice.

---

## Cloning Blocks

You add a block to the workspace by cloning it:

- If you know the palette on which the block resides, you can clone it from the palette.
- If you do not know the palette on which the block resides, you can search for the block.
- You can clone a block already on the workspace.

## Cloning a Block From the Palette

If you know the palette that contains the block you want to clone, you can clone the block directly from the palette.

### To clone a block from the palette:

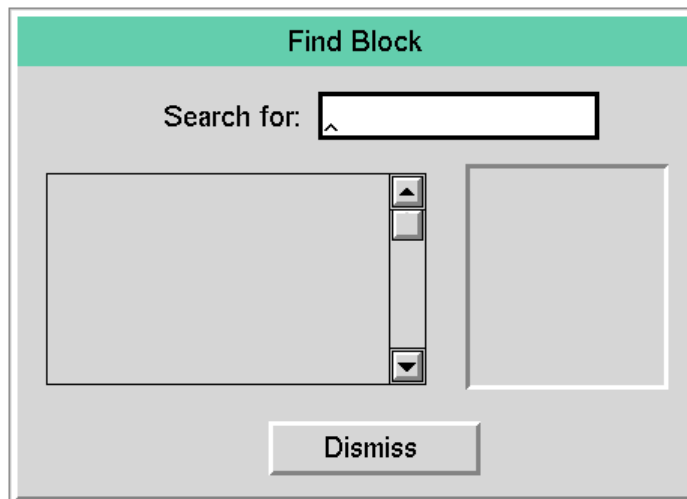
- 1 Select Palettes from the top-level menu, then Data, Inference, Action, or Other to see submenus of available palettes.
- 2 Choose the palette.
- 3 Click on a block on the palette to attach it to the mouse pointer.
- 4 Move the block to the desired location on the workspace and click the mouse again to place the block on the workspace.

## Searching for a Particular Block

If you do not know the palette that contains the block, use the Find Block feature to locate the block by name. You can enter the complete block name, or you can enter part of its name to see a list of all blocks that contain the partial name.

### To find a particular block:

- 1 Press the F2 key or select Help > Find Block to display the Find Block dialog:

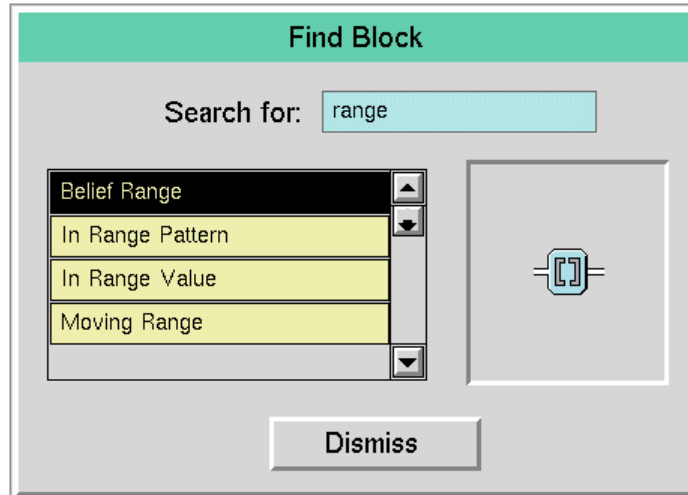


- 2 Enter the name of the block, or part of the name, in the Search for field, then press Return.

For example, suppose you want a block that operates on a range of values but you do not know the full name of the block. You could enter **range** in the Search for field to get a list of all blocks having range in their names.



- 3 Scroll through the list of blocks until you find the one you want, then select it. GDA displays the block in the area to the right of the list. For example, to be able to clone the Belief Range block from this list, select the block name:



- 4 Click on the block to attach it to the mouse pointer.
- 5 Position the block on your workspace and click the mouse again to place it.

You can change the key that GDA uses to invoke the Find Block feature. The key you choose must not be an alphanumeric key.

#### To configure the function key for the Find Block menu choice:

- 1 Select the Preferences > Settings > Environment menu choice to display the Environment Settings dialog.
- 2 Set Browser Key Binding to any allowable key binding, then press OK. The new key binding is effective when you restart G2.

### Cloning an Existing Block

Another way to create a block for use in a diagram is to clone an existing block already on a workspace.

#### To clone an existing block:

- ➔ Click on a block on a workspace, choose the **clone** menu choice, and click to place the block on the workspace.

---

**Note** G2 must be running for you to be able to clone an existing block.

---

The cloned block has the same number of input and output paths, and the same configuration settings as the original block. However, any attribute values of the

original block that were determined since GDA began running (that are not set on the block's configuration dialog box) are not cloned.

## Configuring Block Attributes

Most blocks have attributes that you can configure to specify the behavior of the block. All attributes have default values that enable them to evaluate without being configured. However, you may want to configure your blocks while creating a diagram.

You can configure the attributes of blocks in these ways:

- Editing the attributes of blocks in the configuration dialog box
- Editing attribute displays of blocks
- Editing attributes using the block's API

---

**Note** When overriding a block's value, you cannot simultaneously configure the block.

---

### Configuring Attributes in the Configuration Dialog Box

You use configure a block's attributes using the block's **configuration dialog box**.

#### To configure the attributes of a block using its configuration dialog box:

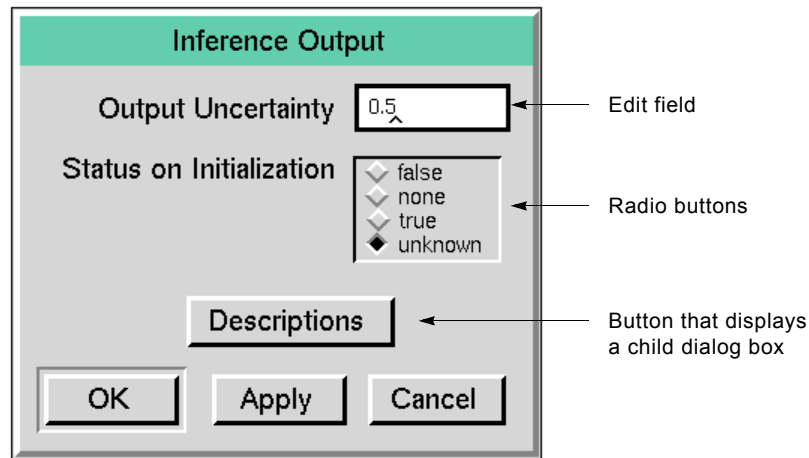
- 1 Select **configure** from the block's menu.  
If a block does not have any configurable attributes, **configure** does not appear as a menu choice for the block.
- 2 Modify the attribute values on the configuration dialog box.

---

**Note** You can specify a block's name and the Dp-out attribute of entry points on the block's table. Otherwise, you cannot use a block's table for entering attribute values into the block, even in Administrator mode.

---

The following figure shows the types of user interface controls you can use to modify attribute values on the Inference Output block's configuration dialog box:



**To enter a value in an edit field:**

- 1 Place the mouse pointer in the field. The background color of the edit box changes, indicating it is active.
- 2 Delete or edit the current value.
- 3 Enter a new value.
- 4 Press the Return key. The background color of the edit field changes to its original color.

If the value you entered is not of the correct type, GDA displays an error message when you press the Return key. GDA also validates dependencies between block attributes when you enter values. For example, for blocks that specify upper and lower thresholds, GDA verifies that the upper threshold is greater than the lower threshold.

When you enter text strings in a configuration dialog box, you do not need to enter quote marks. You only need to enter quote marks in a configuration dialog box when entering an expression, as described in [Evaluating Expressions in Attributes](#).

**To select a new value using a radio button:**

- ➔ Click on the desired button. Only one radio button can be selected at a time.

**To display a child dialog box:**

- ➔ Click on the button within the configuration dialog box.

### To accept or reset the values on the dialog box:

➔ Click on the OK, Apply, or Cancel button at the bottom of the dialog box. The table below describes the actions taken by each of these buttons.

This button...	Performs this action...
OK	Applies the entered values and closes the configuration dialog box
Apply	Applies the entered values and leaves the configuration dialog box open
Cancel	Discards the entered values and closes the configuration dialog box

## Deleting Blocks

Sometimes you need to delete a block from a diagram.

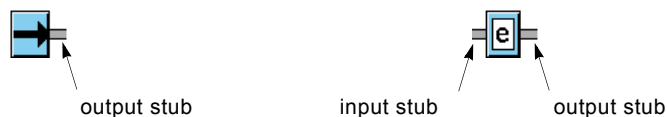
### To delete a block:

- 1 Select **delete** from a block's menu.  
A confirmation message is displayed.
- 2 Click OK to confirm deletion of the block.

## Connecting Blocks

A **stub** is a short connection located on the outside of a block that has not yet been connected to another block. The blocks on the palettes contain varying numbers and types of input and output stubs, depending on the block.

For example, a Numeric Entry Point has only an output stub because it is the starting point for data entering into a diagram. A First Order Exponential Filter has an input stub and an output stub because it takes an input value and passes an output value:



GDA blocks name all their connection stubs. A named connection stub is called a **port**. An **input port** carries data to a block; an **output port** carries data from a block. If you add a connection stub to a block interactively, the stub is not named.

Some blocks have multiple input paths, where the order and meaning of the input stubs is significant. For example, a Difference block subtracts the value on the bottom input data path from the value on the top input data path:



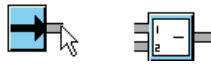
**To connect two blocks together:**

- 1 Click on the output stub of the upstream block.

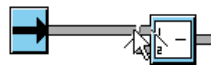
---

**Note** Data flows from the output stub of an upstream block to the input stub of a downstream block.

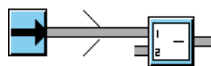
---



- 2 Move the mouse pointer to the input stub of the downstream block. It is not necessary to hold down the mouse button while you move the mouse. Similar to the way a block is attached to the mouse pointer when you clone it from a palette, clicking on the output stub of the upstream block attaches the path to the mouse pointer.



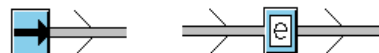
- 3 Click on the input stub of the downstream block. The blocks are now connected. You could also connect these blocks in reverse order, connecting the input stub of the downstream block to the output stub of the upstream block.



Once you have connected two blocks together, the connection between the blocks is called a **path**. If the path is long enough, it has an arrow to indicate the direction of flow of the data.

**To see the direction of flow of any stub:**

- ➔ Drag the stub out from the block or move the blocks further apart:



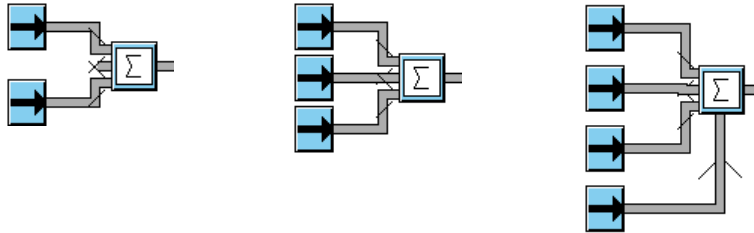
## Connecting to Peer Input Blocks

Some blocks have a fixed number of input stubs, where the order of the inputs is important. For example, the Difference block, shown in the previous section, requires that inputs be in a particular order. Blocks that require that the inputs be arranged in a specific order are called **non-peer input blocks**.

Other blocks can have any number of input paths, where the order of the inputs is unimportant. For example, the Summation block adds any number of inputs; the order of the inputs does not affect the result. Blocks that can have any number of inputs and do not require that the inputs be arranged in a specific order are called **peer input blocks** because their inputs are treated alike, or as “peers.”

When connecting to a peer input block, you can connect to an existing input stub of the block and you can connect to the block itself to create a new stub.

This figure shows Summation blocks that are connected to two, three, and four Entry Points. Notice in the first example, one input stub is not connected. In the last example, the bottom Entry Point is connected directly to the block, creating a new stub:



### To connect to a peer input block:

- ➔ Click on the output stub of the upstream block, move the mouse to an input stub of the downstream block, and click to make the connection.

You can connect as many input paths into the block as are required for the block’s operation. You can also delete existing input stubs as required to improve the appearance of the block.

You can increase the size of a peer input block to accommodate many inputs. For more information, consult the *G2 Reference Manual*.

---

**Note** Peer input blocks ignore input ports that are not connected to other blocks. For more information, see [Determining How Blocks Use no-value Inputs](#).

---

## Deleting Paths and Stubs

Sometimes you need to delete the path between two blocks to connect to other blocks.

### To delete the path between two blocks:

- ➔ Click on the path between the two blocks to display the path menu, then choose **delete**. The path is deleted, leaving a stub.

### To delete an unused input stub on a peer input block:

- ➔ Drag the stub into the block, then release the mouse button. The stub disappears.

You cannot delete the built-in stubs on a non-peer input block. If you drag an unused stub into this kind of block, GDA displays a message indicating that you are deleting a required stub.

---

**Note** You cannot delete the output stub of any block. All output stubs are named ports. The block evaluator uses the port name when executing.

---



---

**Note** When you delete the input stubs of peer input blocks, you must leave at least one input stub on the block. If you attempt to delete the last stub, GDA displays an error message and enables you to restore the stub.

---

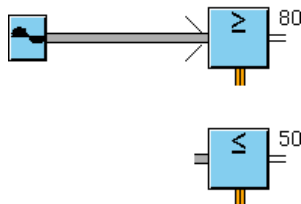
## Connecting One Output to Two Blocks

Sometimes an upstream block provides input to more than one downstream block. GDA displays the junction using a **junction block**.

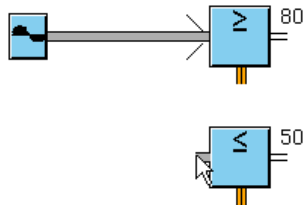
For example, you might want the data coming in through a single entry point to be processed by several different observation blocks.

### To use one block as input to several blocks:

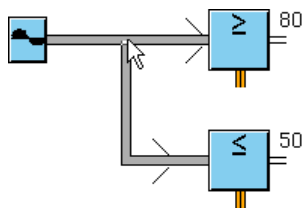
- 1 Connect the data source block to one of the other blocks.



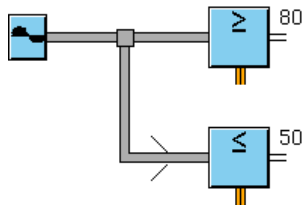
- Place the mouse pointer on the input stub of the unconnected block. Press and hold down the mouse button.



- Move the mouse pointer so it is over the path between the two blocks that are already connected.



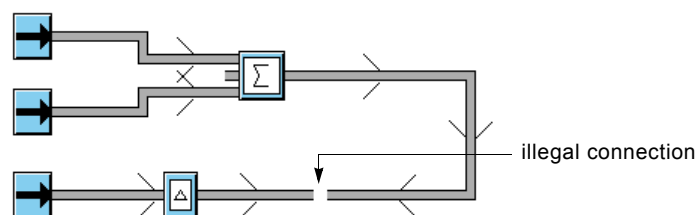
- Release the mouse button. GDA draws a junction block at the junction.



## Rules for Connecting Blocks

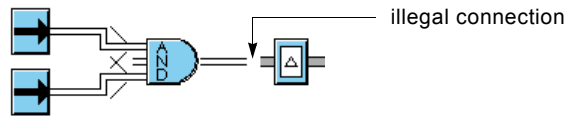
GDA enforces these rules for connecting blocks:

- An output stub of one block must be connected to an input stub of another block.** For example, in this figure an attempt is made to connect the output stub of the Summation block to the output stub of the Changeband Filter block. GDA generates this error message to the Operator Logbook: "can't join directed connections with opposing directions!"

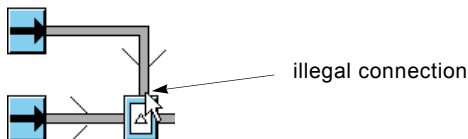




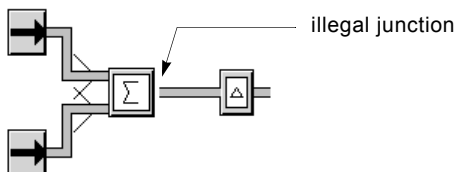
- **The stubs must be the same type.** For example, you can only connect a data path to another data path, or an inference path to another inference path. In this figure, an attempt is made to connect stubs of different types. GDA generates this error message to the Operator Logbook: “can’t join connections with incompatible cross sections!”



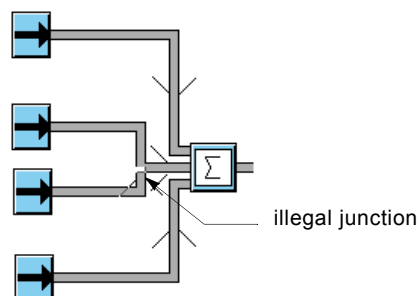
- **You cannot drag a stub into a non-peer input block.** If you try to connect a stub to a non-peer input block directly, GDA displays a message indicating that the connection is illegal.



- **You must always connect to an existing output stub.** For example, you cannot delete an existing output stub on a block and then drag a stub into the block to create the output. GDA displays a message indicating that the connection is illegal.



- **You cannot create junctions into an input path.** You cannot create a junction by connecting the input path from one block into the input path of another. However, you can create this kind of connection by using one of the Connector blocks.



## Improving the Appearance of a Diagram

GDA provides several ways for you to improve the appearance of a diagram:

- Using a snap grid to align objects
- Creating vertices to control the paths between blocks
- Adding annotations

### Aligning Blocks Using a Snap Grid

A snap grid is an invisible grid on a workspace. When the grid is “on,” blocks placed on the workspace “snap” to the closest intersection of vertical and horizontal grid lines.

You can specify these grid settings:

- Whether the snap grid is turned on or off. By default, the snap grid is off.
- The space between grid lines, in pixels. By default, the space between grid lines is 30 pixels.
- The workspaces that the grid controls.

If you turn the grid on for a workspace, that workspace and all subworkspaces below it in the workspace hierarchy use the same grid setting. By default, the snap grid is off.

---

**Note** The snap grid feature does not work if you are in Administrator mode.

---

#### To use a snap grid for a workspace:

➔ In Developer mode, select the KB Workspace > Toggle Snap Grid menu choice on the workspace. This menu choice only appears when the workspace contains GDA blocks.

---

**Note** The snap grid feature does not work with Telewindows2.

---

#### To set the resolution of the snap grid:

- 1 Choose Preferences > Settings > Environment from the top-level menu.
- 2 On the Environment Settings dialog, set the Snap Grid Resolution and click OK.
- 3 If the grid is turned on, you need to turn it off, then on again before the new setting is applied. If the grid is turned off, turn it on.

---

**Note** A snap grid aligns blocks accessible on most GDA palettes. It does not align paths, workspace buttons, messages, connection posts, or alarm readouts.

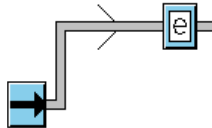
---

**Note** The snap grid exposes a G2 feature using a GDA menu choice. GDA sets the Item-Configuration attribute on the workspace by specifying the constrain moving ... such that the item aligns on a grid clause. For more information, see the *G2 Reference Manual*.

---

### Creating Vertices in Path Connections

Sometimes, a diagram requires a connection other than a straight line. One way to accomplish this is to drag a connected block to a new location to create a **vertex** automatically in the existing path connection:



Sometimes, the automatically created vertex is not exactly in the desired location, and sometimes you need more than one vertex. You can create custom vertices when you create the path between two blocks.

#### To create path vertices when you create the path:

- 1 Click on the output stub of the upstream block and hold down the mouse button.
- 2 Drag the mouse pointer to the location of the first vertex. While still holding down the mouse button, drag the pointer at a right angle to the path to create the first vertex.
- 3 Move the mouse pointer to the location of the second vertex and release the mouse button. Keep the mouse pointer in this location.
- 4 Click and release the mouse button to create another vertex, then move the mouse until you reach the next vertex. Each time you click the mouse button, GDA creates another vertex. When you click on the input stub of a block, GDA connects the stubs.

## Labeling Diagrams

When you create a schematic diagram, it is very useful to label different portions of the diagram to make it easier to understand. There are numerous techniques for labeling your diagrams, including:

- Using free text labels
- Naming objects
- Displaying attribute values

You can also comment your blocks, as described in [Adding Comments to a Block](#).

### To label a diagram using free text:

- 1 Click on the background of a workspace, and select **New Free Text**.
- 2 Another menu appears, offering you a choice of free text objects. Select **free-text** to insert text with a border, or select **borderless-free-text** to insert text without a border.
- 3 Enter the text for the free text object using the G2 editor. You can include any character, including spaces. You do not need to enter quotation marks.
- 4 Press Return to attach the text to the mouse pointer, and click the mouse button to place the free text object on the workspace.

---

**Note** If you use free-text objects to add labels to your diagrams, consider creating a subclass of free-text and create instances of that class instead. Doing this enables you to control the appearance of all such labels in your diagram by defining attributes on the new class without affecting existing free text objects.

---

### To label an object by assigning a name:

- 1 Click on the object to display its menu.
- 2 Select **name** to display the G2 editor.
- 3 Enter the name of the object and press Return.

---

**Note** G2 names cannot contain spaces; by convention, G2 names having more than one word contain hyphens in place of spaces.

---

The name appears in upper case below the object. If the object is a block that has a stub below the block, the name appears on a side of the block that has no stub.

- 4 To move the name, drag the text to a new location.
- 5 To change the name, click on the text, select **edit**, and enter a new name.

**To hide the name:**

- Click the name to display its menu, and select **hide name**. To redisplay the name, click on the object and select **name**, then press Return.

**To change the default fonts used for text:**

- 1 Click on the background of the G2 window to display the Main Menu.
- 2 Select System Tables > Fonts to display the fonts menu, as shown.

FONTS	
Notes	OK
Authors	none
Change log	0 entries
Item configuration	none
Font for attribute tables	large
Font for attribute displays	small
Font for statements	large
Font for free text	large
Font for editing	large
Font for descriptions	small

- 3 Edit the font size of any of these attributes.

The options are: **small**, **large**, and **extra-large**.

For more information about these settings, see the *G2 Reference Manual*.

**Editing Attribute Displays**

Some blocks display attribute values near the icon, and enable you to edit the value directly. For example, this figure shows a High Value observation block displaying the Threshold attribute near the icon, and the portion of the configuration dialog box on which the attribute is specified:



**To configure the attribute of a block by editing its attribute display:**

- ➔ Click on the attribute display next to the block, enter a new value in the type-in area that appears, then press Return.

## Displaying a Block's Table

Each block uses a table to display status information about the block and enables you to enter comments to describe the block. Blocks that enable you to display attribute values also list those attributes on their tables. Some blocks display more attributes in their tables.

---

**Note** Some block tables display more attributes in Administrator mode than in Developer mode.

---

Some blocks define an attribute that contains a variable that keeps a history of the value of the block's output path. For example, all entry points contain attributes for this purpose: dp-out, ip-out, or cp-out and sensor-value. Signal generators contain an attribute named output-value that stores the current sensor value. For more information about using these attributes, see the discussion of entry points in the *GDA Reference Manual*.

**To display the table for a block:**

- ➔ Select **table** from the block's menu.

For example, the table for the Numeric Entry Point looks like this:

a gdl-numeric-entry-point	
Notes	GDL-NUMERIC-ENTRY-POINT-XXX-304: OK
Error	""
Comments	none
Sensor value	5
Dp out	1

The **Notes** and **Error** attributes provide information about the error status of the block. The **Comments** attribute contains user-defined notes about the block. For more information about using these attributes, see [Basic Block Behavior](#).

## Displaying the Value on a Path

Every path has a value that is propagated downstream if data input is enabled. The value stored on the path depends on the path type:

Paths of type...	Store...
Data	Data-value, which is a number, symbol, or text string
Inference	Status-value, which is .true, .false, or unknown  Belief-value, which is a floating point number between 0.0 and 1.0, where 0.0 is .false, 0.5 is unknown, and 1.0 is .true (these values can be configured)

Controls paths do not store any data values.

You determine the value on a path in one of two ways:

- Displaying the path's table
- Adding a path display to the diagram

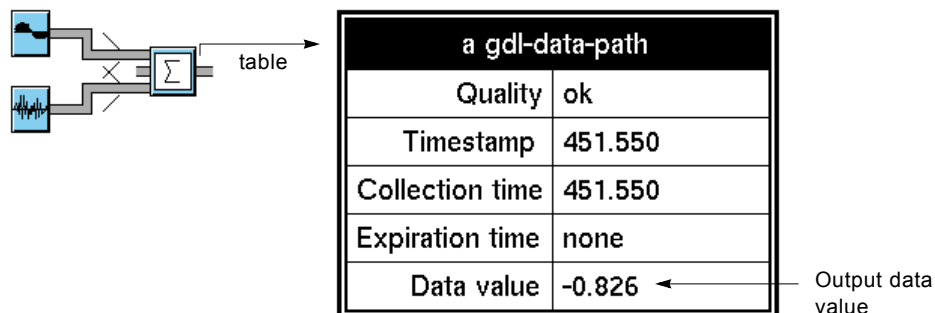
### Displaying the Path's Table

You display a path's table when you only need to see the path value temporarily. The path's table disappears when GDA is reset.

#### To display a path's table:

➔ Click on the path, and select table.

This figure shows the table for output path of the Summation block, which sums the output data values generated by the Sine Wave and White Noise blocks:




---

**Note** You cannot enter data into this table; the attributes are read-only.

---

The path's table contains other information about the path as well:

This path attribute...	Indicates...
Quality	Whether the data is ok, expired, manual, or no-value
Timestamp	The time at which the block received the data
Collection-time	The time at which the data arrived into the diagram
Expiration-time	The time at which the data has expired or will expire

For more information about these path attributes, see [Using Path Attributes](#).

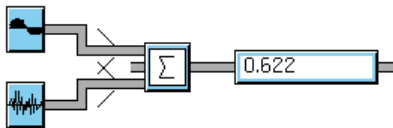
### Adding a Path Display

A Path Display is a block that shows its input value. You add a Path Display block to a diagram when you want the display to persist through restarts and reloads.

**To add a path display to a diagram:**

- 1 Clone a path display of the appropriate type from the Path Displays palette, and place it to the right of the block whose path value you want to display.
- 2 Connect the upstream block's output data path to the Path Display.

The current value is displayed in the path display:



You can also use the Alarm Readout block to display the value on a path. For more information, see the *GDA Reference Manual*.

## Running an Application

Once you have created a schematic diagram, you run the diagram to monitor the data and cause the diagram to take whatever actions are necessary. To run a diagram, you must do the following:

- Load the application (if it is not already loaded)
- Start G2 (if it is not already running)
- Enable data input



You also have the option of toggling animation on or off.

You load your application by using the **G2 Load KB** menu choice. You run your application by selecting menu items on the **GDA Controls** menu.

Once you have developed an application and renamed the top-level module, you load the KB associated with your top-level module to load your application. GDA automatically loads the KB files required to run the product.

**To load a GDA application:**

- 1 Select **Main Menu > Load KB**.
- 2 Enter the complete pathname of the KB you want to load, including double quotes.

---

**Tip** If you do not know the pathname of the file you want to load, enter the name of a directory and press Return to display a listing of the contents of the directory. Select a subdirectory or filename from the list that appears and press Return again until you have located the desired file.

---

For example, the pathname might look something like this for a Windows computer:

```
"C:\Program Files\gensym\g2-8.3r0\gda\myappl.kb"
```

This loads the top-level module that you renamed, which loads all the required modules associated with GDA. G2 reports its progress by displaying a message in the Operator Logbook as each module is finished loading.

- 3 Select **Main Menu > Start**.

## Running Your Own Procedure When You Start G2

You can perform your own routine when you start G2. To do this, you create a G2 procedure that takes no arguments.

**To run your own procedure when you start G2:**

- 1 Select **Preferences > Settings > Startup** to access the Startup Settings dialog.
- 2 Enter the name of your procedure in the Post Global Initializer field, then press OK.

GDA calls the specified procedure after it has finished all its other initialization steps, and before it restarts the scheduler.

**To run your own procedure after resetting:**

- 1 Select Preferences > Settings > Startup to access the Startup Settings dialog.
- 2 Enter the name of your procedure in the Post Global Resetter field, then press OK.

GDA calls the specified procedure after you reset all blocks using the Controls > Reset All Blocks menu choice.

## Controlling the Flow of Data in an Application

You can control the flow of data in a GDA application in these ways:

- You can inhibit data from being propagated or generated by entry points, Signal Generator blocks, or Clock blocks.
- You can inhibit data from being propagated by individual blocks by locking the blocks.
- You can inhibit evaluation (execution) of individual blocks or of all blocks on a workspace.

You can control the flow of data into an entire application, all blocks on a single workspace, or a single block.

### Enabling and Disabling Data Input

When a GDA application starts running, data input is disabled. Entry points, Signal Generator blocks, and Clock blocks do not propagate data while data input is disabled. For those blocks to propagate data, data input must be explicitly enabled.

---

**Note** GDA is configured by Gensym with data input disabled, although an application can be configured with data input enabled (the steps required to do this are described below). As a result, you may find that in your application, data input is enabled.

---

#### To enable data input when it is disabled:

- ➔ Select Controls > Enable Data Input. A check mark appears next to the selection to indicate that data flow is enabled.

When a diagram is running, you can disable data input, which prevents data from flowing through blocks onto their output paths. You might disable data input for these reasons:

- You are building an application and prefer to turn data entry on and off to test and revise the diagrams
- An error is occurring and you want to stop analysis to diagnose the problem

**To enable data input automatically at startup:**

- 1 From the Preferences menu, select Settings > Startup.
- 2 On the Startup Settings dialog box, select Enable Data Input at Startup, then press OK.

**To disable data input when it is enabled:**

- ➔ Select Controls > Enable Data Input when it is enabled. When data input is disabled, no check mark appears next to the selection.

When data input is disabled, you can manually evaluate portions of your diagram by using the **evaluate** or **override** menu choices on blocks in the diagram.

When data input is enabled, you can select a data source for an entry point, external sensor, or internal simulator, or you can disable or enable individual signal generators, clocks, or entry points to control the data introduced into your application. For more information, see [Evaluating Blocks](#) and [Overriding Block Values](#).

**Inhibiting Data Flow Through a Block**

You can inhibit data from being propagated by a block by locking that block. You can only lock blocks that you can manually override. You can lock a block in two ways:

- By selecting **lock** from the block's menu
- By connecting the action link from a Lock block from the Action blocks palette to the block, which locks the block when the Lock block evaluates

For more information about locking blocks, see [Locking and Unlocking Blocks](#).

**Enabling and Disabling Evaluation of a Block**

You can prevent a block from evaluating in two ways:

- By selecting **disable evaluation** from the block's menu
- By connecting the action link from a Disable block from the Action blocks palette to the block
- By calling the disable evaluation API (see the *GDA API Reference*)

For more information, see [Enabling and Disabling Evaluation](#).

---

**Caution** Workspaces provide a **Disable** menu choice. You should not disable a workspace that contains GDA blocks. When a workspace is disabled, G2 considers that objects on the workspace do not exist. GDA does not initialize or compile these objects, which may cause unpredictable results. If you disable a workspace that does not contain blocks, make certain that the workspace does not have any subworkspaces that contain blocks. Also, when in administrator mode, do not choose the **disable** menu choice on an individual block.

---

## Toggling Animation

Animation causes these changes in a diagram:

- Blocks flash as they evaluate.
- Inference paths indicate their states by displaying color.
- Blocks having numeric displays update their values.

By default, GDA animates blocks as they evaluate to provide you with feedback about your running process. Animation causes an application to run more slowly, so you can improve the performance of your application by turning animation off.

The **Controls > Animate** menu choice appears checked or unchecked, depending on whether animation is enabled or disabled.

### To start animation when it is off:

- ➔ Select **Controls > Animate**. A check mark appears next to the menu choice to indicate that animation is turned on.

### To stop animation when it is on:

- ➔ Select **Controls > Animate** when it is checked. The check mark next to the menu choice is removed, indicating that animation is turned off.

Blocks remain highlighted for the duration defined by the Animation Delay.

### To set the animation delay:

- 1 Select **Preferences > Settings > Environment** to display the Environment Settings dialog.
- 2 Change the Animation Delay value, then press OK.

### To disable animation on startup:

- 1 Select **Preferences > Settings > Startup** to display the Startup Settings dialog.
- 2 Specify Animation at Startup as no.

## Allowing Other Processing

By default, when GDA encounters an allow other processing statement in a procedure, it can process ten blocks. You can change the number of blocks.

**To set the number of blocks that can process when allowing other processing:**

- 1 Select the Preferences > Settings > Environment menu choice to display the Environment Settings dialog.
- 2 Set Blocks between Allow-other-processing, then press OK.

## Setting the Maximum Timeout for Data Seeking

Some blocks use data seeking to obtain values: entry points, signal generators, filters, and charts. By default, the maximum data seeking timeout is 30.0 seconds. You can set the maximum amount of time GDA will seek a value before timing out.

**To set the maximum timeout for data seeking:**

- 1 Select the Preferences > Settings > Environment menu choice to display the Environment Settings dialog.
- 2 Set Maximum Data Seeking Timeout, then press OK.

## Connecting to Remote Processes

When you use the Network blocks in GDA to connect to other G2s, you are using a remote process. You can control how long GDA looks for a remote process before returning an error and how often GDA should retry connecting to the remote process.

**To set the remote process timeout period and retry period:**

- 1 Select the Preferences > Settings > Remote Process menu choice to display the Remote Process Settings dialog:

- 2 Set the value or values, then press OK.

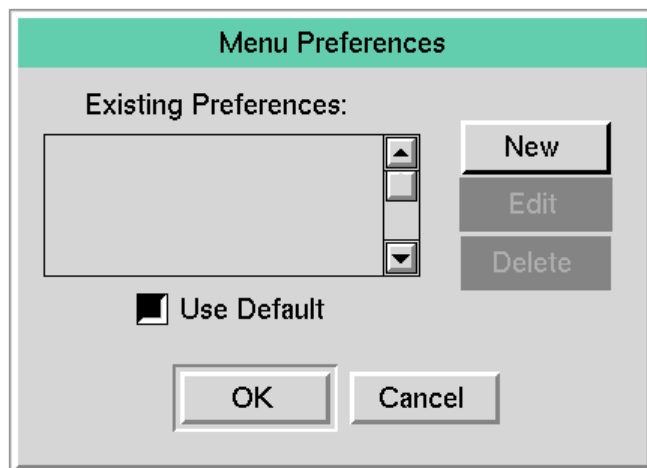
# Customizing Menus

## Using Menu Preferences

You can specify a menu preference that sets the font size, language, initial menu bar, and colors of the text and background that GDA uses in its top-level menu bar. You can create multiple preferences.

**To create a new menu preference:**

- 1 Select Preferences > Menus to display the Menu Preferences dialog.



- 2 To create a new set of menu preferences, click New.

To use the default as a base, click the Use Default button (if it is not selected) then click New. The Menu Preference Attributes dialog appears.

You can also use an existing preference as a base for the new menu preferences. First, select a menu preference in the Existing Preferences list, then click New. The Menu Preference Attributes dialog appears.

**Menu Preference Attributes**

Preference Name

Font Size

Language

Initial Menu Bar

Normal Text  
 Normal Background  
 Highlighted Text  
 Highlighted Background  
 Header Text  
 Header Background  
 Disabled Text

OK Cancel

- 3 To name a menu preference, enter its name in the Preference Name field.

---

**Note** You must specify the Preference Name. Also, do not use spaces in the name of the preference.

---

- 4 To specify the size of the characters in the menu bar, enter one of the following in the Font Size field: **small**, **large**, or **extra-large**.
- 5 To specify the language, type the name of the language, such as **english**, in the Language field.

To use the default language, type **none**.

- 6 To specify the menu bar that appears when you start GDA, enter the name of the menu in the Initial Menu Bar field.

The name of the menu bar must match a G2 Menu System (GMS) user key value in a GMS menu bar template. To use the default initial menu, type **none**. For more information, see the *G2 Menu System User? Guide*.

- 7 To specify the colors of the menu text and its background, click on the color squares shown at the right of the dialog.

You can customize four types of text:

- Normal - unselected text and its contrasting background in the menu bar.
- Highlighted - selected text.
- Header - text in GMS popup menus (not used in GDA).
- Disabled - text that is not selectable, such as user-defined palettes when none have been defined.

When you click the color square, you see a display of available colors. Select one. The color of the square reflects your choice.

- 8 To complete specifying your menu preferences, click OK in the Menu Preference Attributes dialog. The dialog closes.
- 9 To activate your set of menu preferences, make sure it is selected, then click OK in the Menu Preferences dialog.

**To edit an existing menu preference:**

- 1 Select Preferences > Menus to display the Menu Preferences dialog.
- 2 Click on the name of the preference, then click Edit.  
The Menu Preference Attributes dialog appears.
- 3 Follow the steps for creating a new menu preference, beginning with Step 4.

**To use the default menu preference settings:**

- 1 Select Preferences > Menus to display the Menu Preferences dialog.
- 2 Click the Use Default button in the Menu Preferences dialog, then click OK.

**To delete a menu preference:**

- 1 Select Preferences > Menus to display the Menu Preferences dialog.
- 2 Select the preference to delete in the Existing Preferences box in the Menu Preferences dialog, click Delete, then confirm the deletion on the confirmation dialog that appears. Then, click OK to close the Menu Preferences dialog.

## Creating New Menu Preference Objects

When GDA is installed, it contains two menu preferences objects: `Gms-default-configuration`, included in the GMS module, and `gda-menu-configuration`, included in the GDA module.

GDA determines which menu preferences object controls the menu preferences by using the GMS API.

`Gms-default-configuration` is initially defined with a priority of 0; `gda-menu-configuration` is initially defined with a priority of 1. By default, a GDA application displays a menu with its preferences defined by `gda-menu-configuration` because it has the higher priority.

When you modify menu preferences by using the Menu Preferences dialog in GDA, GDA creates a new preferences object and assigns it the name you specify. GDA assigns the priority of this new object to be 1 greater than the priority of the previously active preferences object. You choose a set of menu preferences to control your application's menu appearance by selecting its name from a list of menu preferences objects.



---

**Note** It is important to manage menu preferences exclusively within the Menu Preferences dialog supported by GDA, or using the API supported by GMS. Because the GDA dialog manages preferences by using only priority, the GDA menu may not fully override a preference set using the GMS API. Conversely, if you have given your GMS meaningful priority values, and then use GDA to choose a new preference, GDA automatically modifies the preferences object's priority and may make it difficult to maintain the meaning of the priorities you established.

---

If you create a new preferences object outside of GDA, it is not necessary to specify a name for this object. However, if you have an unnamed preference which you then try to access through the GDA menu, GDA will display an error message indicating that the object cannot be referenced. This behavior should be fixed in an upcoming release of GDA.

## How GDA Manages the Default Menu Preferences

End users can choose to override menu preferences by reverting to the default preferences. Each menu preferences object is derived from a “root” preferences object. This root object supplies the default preferences.

- If the application user selects a preferences object created in GDA and chooses the Use Default check box, the menu preferences defined by `gda-menu-configuration` apply, because this is the root preferences object.
- If the application user selects a preferences object created in GMS and chooses the Use Default check box, the root preferences object is the User Preferences Object from which the selected preferences object was derived.

In addition to this section, for more information about customizing menu preferences, see the *G2 Menu System User's Guide*, Chapter 14, “Customizing the GMS Interface to the User”.

## Extending Main Menu Choices

You can extend the menu choices in the main menu bar. The main menu bar is constructed using the G2 Menu System (GMS) utility. To provide for menu translation, GMS uses the local text resources feature of the G2 Foundation Resources (GFR) utility.

Using GMS and GFR, you can create extensions or modifications to the GDA menu bar. You can display the graphical representation of the GDA menus via the menu bar.

If you want to add your own custom menu choices to the menus in the GDA menu bar, begin by displaying the GMS graphical representation of the menu bar.

**To display the GMS menu bar diagrams:**

→ Choose Main Menu or Additional Menu in the Show > Menus menu.

For more information about GMS, see the *G2 Menu System User? Guide*. For more information about GFR, see the *G2 Foundation Resources User? Guide*.

## **Translating Menu Choices**

You can use the GFR (Gensym Foundation Resources) module of G2, which is included with GDA, to translate menu choices to other languages. For more information, see the *G2 Foundation Resources User? Guide*.

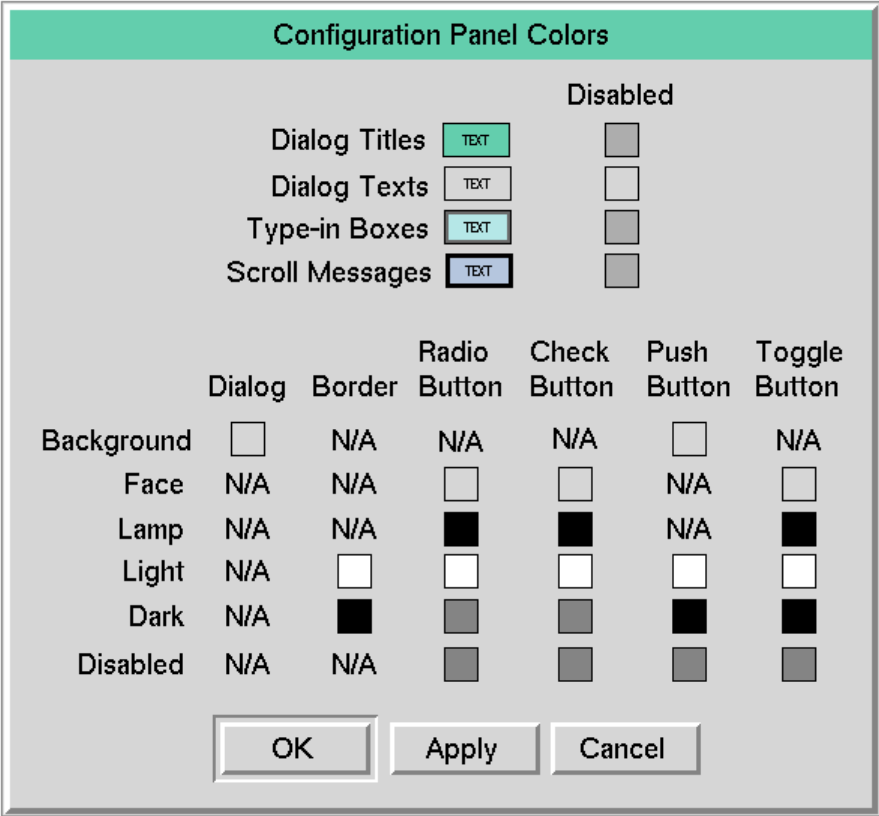
# **Miscellaneous Features**

## **Setting Configuration Panel Colors**

The Configuration Panels menu item lets you customize the colors of configuration dialogs that GDA displays when you select the `configure` menu choice and when you select the `override` menu choice.

**To customize the colors of configuration panels and override dialogs:**

- 1 Select Preferences > Colors > Configuration Panels to display the Configuration Panel Colors dialog:



- 2 To customize the color of dialog titles, text, type-in boxes, and scroll areas, click on the appropriate box, choose a region, and choose a color from the G2 color palette.

The regions are border-area, background-area, and text-area.

**Note** You typically choose the same background and border color for dialog titles.

- 3 To customize any of the other colors, click on a color to display the G2 color palette.
- 4 Continue configuring colors as desired, and click OK or Apply.
- 5 Display a configuration panel to see the new colors.

## Setting the Color for Titles, Type-in Boxes, Text, and Scroll Messages

The following table describes the four top color settings in the palette. All colors refer to items within configuration panels displayed using the `configure` menu choice.

<b>The color setting labeled...</b>	<b>Changes...</b>
Dialog Titles	The background, text, and border color of the title bar.
Dialog Texts	The background, text, and border color for labels.
Type-in Boxes	The background, text, and border color for type-in boxes.
Scroll Messages	The background, text, and border color for type-in boxes displayed in a scrollable region.

When you click on a color, choose one of these regions:

<b>The region named...</b>	<b>Specifies...</b>
border-area	The border color for the item; use <code>transparent</code> for no border or <code>background</code> color for no border.
background-area	The background color for the item.
text-area	The color of the text for the item.

You can also set the color that GDA uses to indicate a disabled dialog title, dialog text, type-in box, or scroll message by clicking on the colors `Disabled Colors` for each of the settings described above.

## Setting the Color for Display Items Within Configuration Panels

You can adjust the color of these display items within a configuration dialog:

<b>The display item...</b>	<b>Controls the color of...</b>
Dialog	The overall configuration panel.
Border	The outer border of the configuration panel.

<b>The display item...</b>	<b>Controls the color of...</b>
Radio Button	The diamond-shaped buttons used for attributes with distinct choices, e.g., Logic, which can be <b>discrete</b> or <b>fuzzy</b> .
Push Button (Action Button)	The OK, Apply, and Cancel buttons at the bottom of the configuration panel.

---

**Note** GDA only uses the display items shown in the table above; the other display items appear for future compatibility only.

---

The following table describes each category of color in the bottom portion of the dialog:

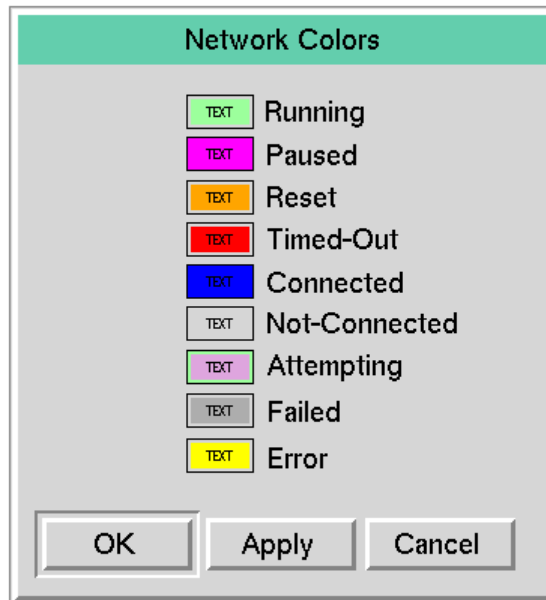
<b>The color setting labeled...</b>	<b>Changes...</b>
Background	The background color of the dialog and push buttons.
Face	The interior color of radio, check, and toggle buttons.
Lamp	The color used when radio, check, and toggle buttons are selected.
Light	The top and left borders of a button that is not selected, the bottom and right borders of a button that is selected, or the bottom and right borders of a display item.
Dark	The bottom and right borders of a button that is not selected, the top and left borders of a button that is selected, or the top and left borders of a display item.
Disabled	The color used for disabled buttons.

## Setting Network Colors

The Network Colors dialog lets you change the colors that represent the status of a Remote G2 Process block, described in the *GDA Reference Manual*.

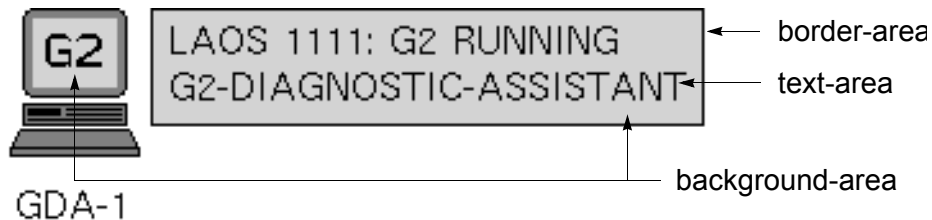
**To edit the default colors of network objects:**

- 1 Select Preferences > Colors > Network to display the Network Colors dialog:



- 2 Click on one of the colors on the panel to display a table that lets you set three colors for the regions in the icon for a Remote G2 Process object.

This figure shows you what the regions are:

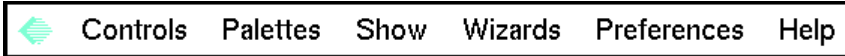


- 3 Choose a region to display the G2 color palette.
- 4 Choose a color and click OK or Apply.
- 5 Display a network object to see the new colors.

This table describes the colors on this panel:

<b>The object labeled...</b>	<b>Lets you set the colors for a G2 Remote Process object...</b>
Running	Whose remote G2 is running.
Paused	Whose remote G2 is paused.
Reset	Whose remote G2 is reset.
Timed-Out	Which timed-out while trying to establish a connection.
Connected	Which is establishing a connection to the remote G2.
Not-Connected	Which has not tried to establish a connection.
Error	Which failed to establish a connection.

GDAgdaapps



GDA



-





# Using Blocks and Paths

---

*Describes the common features of all blocks and describes how to use the basic types of paths and their attributes.*

Introduction	48
Basic Block Behavior	48
Using Paths	60
Using Path Attributes	70
Creating Customized Path Connections	78
Specifying Initial Values	82
Maintaining a History of Values	83
Specifying How to Handle Multiple Values	91
Specifying and Generating Explanations	92
Specifying Fuzzy Logic Attributes	100
Using Variables and Parameters	104
Evaluating Expressions in Attributes	117
Using the GXL Spreadsheet to Edit Data	123
Understanding the GDA Block Evaluation Engine	125



# Introduction

This chapter describes the features of blocks and paths, and explains how to interpret the attributes of each path type. This chapter also discusses attributes and menu choices common to blocks that are on more than one palette.

For information about attributes and menu choices common to blocks that are on the same palette, see the chapter that describes blocks on the palette in the *GDA Reference Manual*. For information about attributes and menu choices that appear on only one block see that block's description in the *GDA Reference Manual*.

## Basic Block Behavior

Most blocks share several menu choices and attributes that enable you to:

- Read status notes and error messages, described on .
- Add comments to a block, described on .
- Reset the block, described on .
- Evaluate the block, described on .
- Override the block's value, described on .
- Enable and disable evaluation for a block, described on .
- Clear the block error, described on .

This section describes how these basic menu choices and block attributes work.

For information on common attributes that you can configure for blocks, see:

- [Specifying Initial Values](#)
- [Maintaining a History of Values](#)
- [Specifying How to Handle Multiple Values](#)
- [Specifying and Generating Explanations](#)
- [Specifying Fuzzy Logic Attributes](#)

## Reading Notes and Errors

The Notes and Error attributes of a block display the status of the block. These attributes appear in the table for the block. You cannot edit these attributes.

**To display the notes and error for a block:**

- ➔ Select table from the block's menu.

The following figure illustrates the table for a High Value observation block:

a gdl-high-value-observation	
Notes	OK
Error	""
Comments	none

The **Notes** attribute is defined for all G2 items. The **Error** attribute is defined for all GDA objects.

Most of the errors you encounter while developing an application are described in a block's **Error** attribute. When GDA finds an error, it turns the color of the affected block to the error color (yellow, by default) and writes a description of the error to the **Error** attribute of the block and to the Error Queue. After you fix the error, select **reset** from the block's menu to clear the **Error** attribute.

You might sometimes find messages in the **Notes** attribute, which is used for status and error information. It contains the text **OK** if G2 finds no errors in the block but may contain **OK** even if GDA finds an error in the block.

## Adding Comments to a Block

The **Comments** attribute enables the application developer associate a comment with a block. For example, a comment would be useful to describe how a block is configured. You can also use G2 ProTools to document an application.

### To add a comment to a block:

- ➔ Display the table for the block, and click on the value of the **Comments** attribute. Enter a value in the editor. To create a line break, enter a Ctrl-J.

The following figure shows the table for a Bias block with a text string in the **Comments** attribute:

a gdl-fixed-bias	
Notes	GDL-FIXED-BIAS-XXX-107: OK
Error	""
Comments	"adjust for weight of tank"

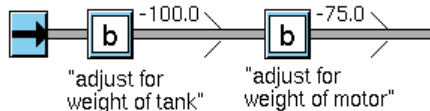
**To show the comment associated with a block near the block icon:**

- 1 Display the table for the block.
- 2 Click on an empty place within the area where the value of the Comments attribute appears to display the table item menu.
- 3 Select the show attribute display menu choice:

a gdl-fixed-bias	
Notes	GDL-FIXED-BIAS-XXX-107: OK
Error	""
Comments	"adjust for weight of tank"

table item
edit
transfer
show attribute display
hide table

- 4 The comment appears below the block, as this figure shows:



## Resetting Blocks

When you first start G2, GDA automatically resets all the blocks in the application, restoring all knowledge in the KB to its initial state.

Sometimes you need to reset individual blocks, such as when an error occurs. You might also want to reset all the blocks in the application. You can reset any block manually by using a menu choice.

**To manually reset an individual block:**

- ➔ Select reset from the block's menu.

**To reset all the blocks in your diagram:**

- ➔ Select Controls > Reset All Blocks. GDA asks whether you also want to reset all paths.

GDA displays a confirmation message before it resets all blocks and reports the status of all blocks it finds and resets them.

When you **reset** a block, GDA does the following:

- Sets the block to its initial state. If the block has a Status on Initialization or Value on Initialization attribute, GDA also propagates the block's initial value. For more information, see [Specifying Initial Values](#).
- Clears any error condition. The block's color is changed from yellow to blue and the Error attribute is set back to a null string (""). You can also clear errors for a block without resetting; see [Clearing Block Errors](#).
- Unlocks the block if it was locked, as described in [Locking and Unlocking Blocks](#).
- Erases the block's history, if the block maintains a history and if the attribute Erase History When Reset is **yes**, as described in [Specifying What Happens to History Upon Reset](#).

Some blocks do more things when reset. Some blocks that perform an action will undo that action when reset. For example, the Show Workspace block hides its workspace when reset. For more information about how resetting a block affects the block, see the *GDA Reference Manual*.

Resetting blocks does *not* set the values of the attributes in the configuration table to their default values. Also, resetting a block does not reset its output path value. For information on resetting the path value, see [Resetting Paths](#).

## Evaluating Blocks

When you test an application, you can **evaluate** a block manually even though it has not received new data.

When you manually evaluate a block, the block uses its existing input values. Thus, for certain blocks, such as observation blocks, manually evaluating the block has no effect.

When you evaluate an entry point, the current value is propagated with a new timestamp.

When you evaluate a block with a single input control path, the block acts as if it has received a new control signal.

### To evaluate a block manually:

- ➔ Choose **evaluate** from the block's menu.

## Overriding Block Values

When you test an application, you can **override** a block's output value to make sure your application responds to it correctly, or to pin-point exactly where an error is occurring. Also, you override an entry point to test a diagram that uses entry points that have not yet been connected to a real data source.

You can manually override these block's output value: Entry Points, Signal Generators, Observations, Conditions, User Query Control Switches, Network Entry Points, and the Conclusion block.

Overriding the value of a block does these things:

- It propagates the manually entered value.
- It locks the block, which prevents the propagation of values the block receives from other data sources.
- It propagates a Quality of manual onto the path.

Once a block is locked, only the manual override value is propagated downstream.

---

**Note** When overriding a block's value, you cannot simultaneously configure the block.

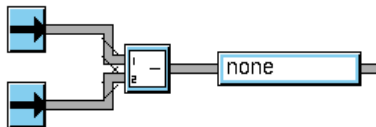
---

**To override the value of a block:**

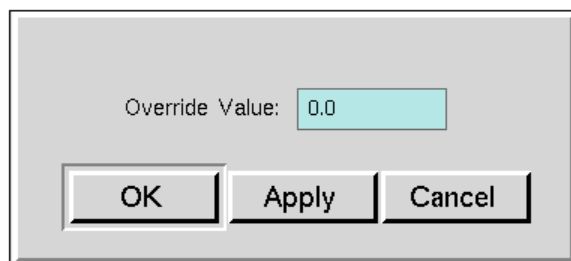
- 1 Select **override** from the block's menu.
- 2 Enter a value in the edit field of the dialog and press Return.
- 3 Click OK to apply the value and close the dialog.

Otherwise, to apply the value and leave the dialog open, select **Apply**. To prevent applying the value you have entered and close the dialog, select **Cancel**.

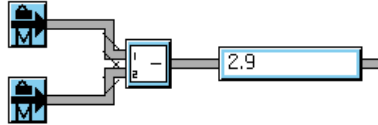
For example, this diagram shows two Numeric Entry Point blocks providing input to a Difference block:



When you select **override** from an entry point's menu, this dialog box appears:



After overriding the values of both entry points and evaluating the Difference block, the diagram looks like this. GDA displays the letter M to indicate manual override, and a lock symbol to indicate that the block is locked:



The Quality of the output path value of the block you override is **manual**. The Quality of the output path value for downstream blocks that calculate values based on a manual value is also **manual**. For example, here is the table for the output path of the top Numeric Entry Point block in the example:

a gdl-data-path	
Quality	manual
Timestamp	160.550
Collection time	160.550
Expiration time	none
Data value	4.2

Depending on the type of block whose value you are overriding, GDA displays one of several different dialogs:

When overriding...	You get an override dialog with...
A data block	An edit box for entering a number, symbol, or text
An inference block that uses discrete logic	Radio buttons for selecting <b>.true</b> , <b>.false</b> , or <b>unknown</b>
An inference block that uses fuzzy logic	A belief slider from 0.0 to 1.0 for entering a fuzzy belief value

GDA displays a dialog box that lets you enter a value. There are four different override dialogs:

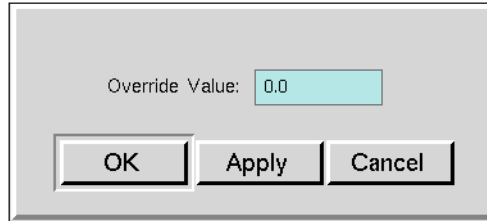
- The **data override dialog** lets you enter a number, string, or symbol.
- The **control override dialog** lets you pass a single control signal.
- The **discrete inference dialog** lets you enter the value **.true**, **.false**, or **unknown**.
- The **fuzzy inference dialog** lets you enter a value from 0.0 to 1.0.

## Overriding a Data Block

A data block displays the dialog box in the following figure. The dialog contains a type-in box where you can enter a number or string.

### To enter a value:

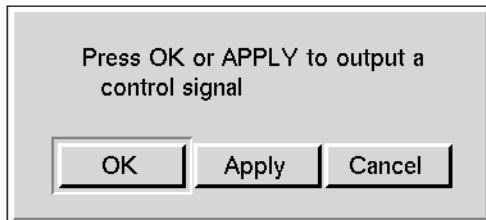
- 1 Click in the edit box, enter the value, and press the Return key.
- 2 To pass the value, click OK or Apply.



To change the text in the dialog box, attach a Dialog Restriction to the block, and edit the attributes Override Text and New Value Prompt in the restriction. For more information, see the description of the Dialog Restriction block in the *GDA Reference Manual*.

## Overriding a Control Block

Overriding a control block sends a control signal down the block's output path. When you select override from the block's menu, this dialog box appears:



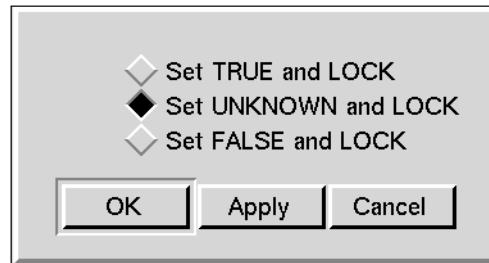
## Overriding a Discrete Inference Block

An inference block whose Logic attribute is set to **discrete** displays the dialog box in this figure. It contains a list of three radio buttons: one each for **.true**, **unknown**, and **.false**.



**To override the value:**

→ Click on the value you want, then click OK or Apply.



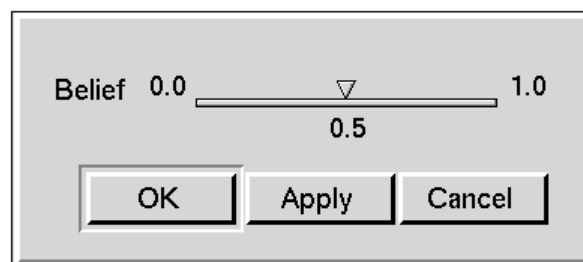
If a block has the attributes Description When True, Description When Unknown, and Description When False, you can change the text of its override dialog. Enter the descriptions for the three choices in these attributes. For more information, see [How to Describe Why a Block Passes True, False, or Unknown](#).

**Overriding a Fuzzy Inference Block**

An inference block whose Logic attribute is set to fuzzy displays the dialog box in this figure. The dialog contains a slider that lets you choose a value between 0.0 and 1.0.

**To override the value:**

- 1 Click on the triangle above the line, and drag it along the line until the number below the line displays the value you want.
- 2 To pass the value, click OK or Apply.

**Locking and Unlocking Blocks**

When you override a block's value, the block displays a lock symbol and an "M" on the icon, indicating that it is locked and the override is manual.

When a block is **locked**, it does not respond to input or pass another value.

**To stop a block from passing its value:**

→ Choose lock from the block's menu.

**To let a block that you have overridden pass values normally again:**

→ Choose either **unlock** or **reset** from the block's menu.

You can only lock and unlock blocks that you can also override. Typically, to prevent data from propagating downstream, you disable evaluation, as described in the next section.

## Enabling and Disabling Evaluation

You can stop a block from passing its output value and responding to new input values by disabling block evaluation. For example, while you are developing your application, you may want to disable part of the application so you can concentrate on another part.

You can **enable** and **disable evaluation** of any individual block in a diagram or all blocks on a workspace.

You can also use the **Disable** and **Enable** blocks to toggle a block or workspace between evaluation and no evaluation. For more information, see the *GDA Reference Manual*.

---

**Caution** Enabling and disabling blocks and workspaces for evaluation is different from enabling and disabling blocks and workspaces in G2. Do not use the G2 menu choices **enable** and **disable** for blocks or **KB Workspace > Enable** and **KB Workspace > Disable** for workspaces.

---

By default, all blocks are enabled for evaluation. Enabling or disabling a block enables or disables any attached capabilities and restrictions as well.

**To disable evaluation of a single block:**

→ Click on a block that is enabled and select **disable evaluation**.

Selecting this menu choice causes the block to stop evaluating and prevents data from flowing downstream from the block.

**To enable evaluation of a single block:**

→ Click on a block that is disabled and select **enable evaluation**.

Selecting this menu choice causes the block to evaluate. It may also cause data to flow downstream again.

**To disable evaluation of all blocks on a workspace:**

→ Click on a workspace whose evaluation is enabled and select **KB Workspace > Disable Evaluation**.

Selecting this menu choice causes all blocks on the workspace to become disabled.

**To enable evaluation of all blocks on a workspace:**

- Click on a workspace whose evaluation is disabled and select KB Workspace > Enable Evaluation.

Selecting this menu choice causes all blocks on the workspace to become enabled.

---

**Note** When you enable or disable evaluation of blocks on an Encapsulation or Single-source Encapsulation block's workspace, you enable or disable all blocks on the workspace, including blocks on any subworkspaces of nested encapsulations.

---

## Clearing Block Errors

When a block produces an error when it evaluates, the block turns yellow and an error message appears. You can clear the error on an individual block without resetting the block. Resetting a block also clears any errors in the block, as described in [Resetting Blocks](#).

**To clear an error in a block:**

- Select the clear error menu choice for the block.

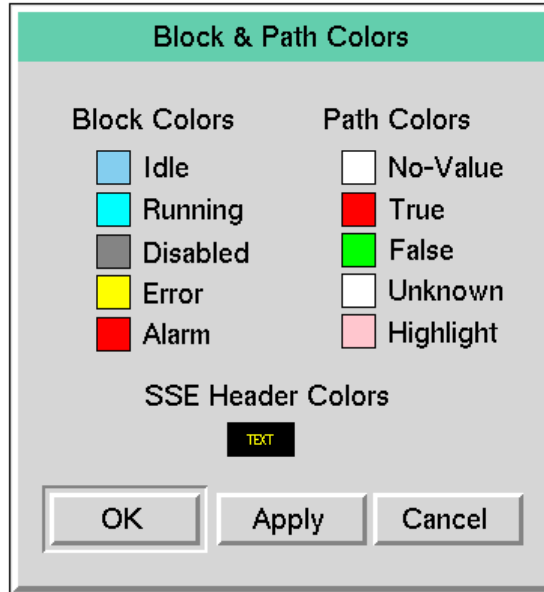
## Setting Block Colors

You can change the colors used to show the status of blocks in your diagram. For example, blocks show a different color when they are on the palette, when you first clone them from the palette, when they are active, and when they are idle.

When animation is enabled, GDA uses the color preferences you specify for blocks. When animation is disabled, GDA uses the colors blocks define in their icon descriptions.

### To customize the colors of blocks:

- 1 Select the Preferences > Colors > Blocks & Paths menu choice to display the Block & Path Colors dialog:



- 2 The Block Colors on the left of the dialog control the colors that indicate block states, as described in this table:

Block Color	Description
Idle	Blocks that are idle.
Running	Blocks that are evaluating.
Disabled	Blocks that are disabled for evaluation by selecting disable evaluation on the block or workspace that contains the block.
Error	Blocks in an error state.
Alarm	The lines around an Alarm capability that indicate that the alarm is active.

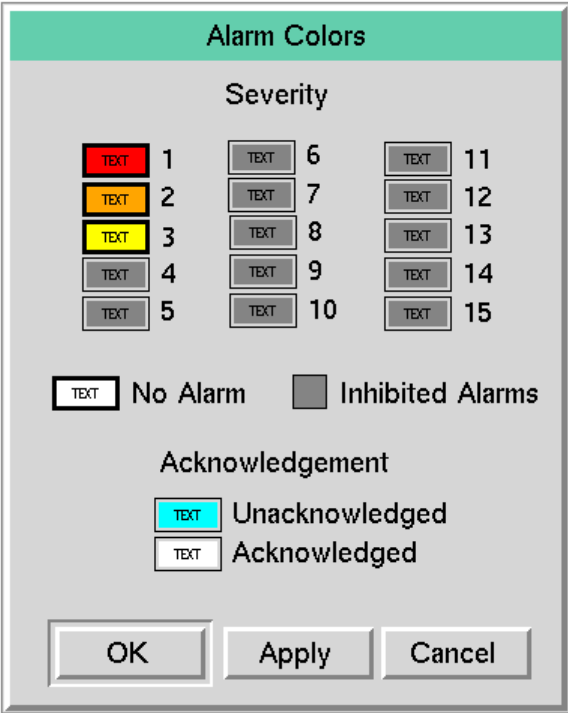
- 3 Choose a block state to change its color. When you click on the colored square next to the block state, a palette of colors appears.
- 4 Select a color from the palette of colors.
- 5 Continue customizing the colors as desired, and click OK to accept the changes and dismiss the dialog.
- 6 Toggle animation off and then on to see the change by using the Controls > Animate menu choice.

# Setting Alarm Colors

You can change the colors that represent the severity of alarms in Alarm Panels.

**To edit the default colors of network objects:**

- 1 Select Preferences > Colors > Alarms to display the Alarm Colors dialog:



- 2 Click on one of the colors on the panel to display a table that lets you set three colors for the regions in the icon for an Alarm Panel.
- 3 Choose a region to display the color palette.
- 4 Choose a color and click OK or Apply.
- 5 Display an Alarm Panel to see the new colors.

This table describes the colors on this palette:

The object labeled...	Lets you set colors for...
No Alarm (Alarm Readouts)	An Alarm Readout that is associated with an alarm that is not active.
No Alarm (Alarm Displays)	An Alarm Panel that is associated with an alarm that is not active.

The object labeled...	Lets you set colors for...
No Alarm (Alarm Messages)	A message in an Alarm Queue that is associated with an alarm that is no longer active.
Selected Alarms	A message in an Alarm Queue that you have selected.
Inhibited Alarm	An alarm capability you have inhibited with the inhibit command and all the displays (Alarm Panels and Readouts) associated with that alarm. This object brings up the G2 color palette directly.
Severity 0 Alarm to Severity 15 Alarm	An alarm that is active and has the specified severity, and all the displays (Alarm Panels and Readouts) associated with that alarm.

---

**Note** For more information about Recurring Alarms, Local Alarm Panels, and Group Alarm Panels, see the *GDA Reference Manual*.

---

## Using Paths

When a diagram is running, blocks evaluate once per sweep if the block has received a new value on one of its input paths. Each block processes its input, then passes the resulting output onto its output path or paths. Any block connected to that path then uses that value as its input.

Each path carries data compatible with the path type:

This path type...	Carries this type of information...
Data paths	Numeric, textual, or symbolic values.
Inference paths	Truth values (.true, .false, or unknown) or fuzzy belief values (a number between 0.0 and 1.0)
Control paths	Signals which cause blocks to evaluate.
Item paths	Instances of items of a particular class.

Each path has an associated table with attributes, which provide information about the current path value, the quality of the data, and the time at which the data arrived. For detailed information about the attributes of paths for the various path types, see [Using Path Attributes](#).

In addition to these path types, GDA has **links**, for use in conjunction with action blocks and capabilities.

By default, GDA represents the different types of paths using different colors:

<b>Paths of type...</b>	<b>Have this color...</b>
Data paths	Gray
Inference paths	Red when <code>.true</code> Green when <code>.false</code> White when unknown
Control paths	Orange

For information about customizing path colors, see [Setting Path Colors](#).

## Using Data Paths

**Data Paths** can pass numeric or text data, or symbols. This table describes data path attributes:

<b>Attribute</b>	<b>Description</b>
Data-value	Numeric, textual, or symbolic value.
Collection-time	The time that the data first entered the diagram through an Entry Point, entered G2, or was concluded. See <a href="#">The Collection-Time Attribute</a> .
Quality	The status of the path's data. See <a href="#">The Quality Attribute</a> .
Timestamp	The time that the path received its value. For more information, see <a href="#">The Timestamp Attribute</a> .
Expiration-time	The time that the current value for the path expires. For more information, see <a href="#">The Expiration-Time Attribute</a> .

## Using Inference Paths

**Inference Paths** carry two types of values: **belief** and **status**. A belief value is a number from 0.0 to 1.0, where 0.0 is completely false and 1.0 is completely true. Blocks derive a status value from the belief value, and can be one of the symbols `.true`, `.false`, or `unknown`. For more information on how a block derives these values, see [Specifying the Type of Logic to Use](#).

This table describes the inference path attributes:

Attribute	Description
Status-value	One of the symbols <code>.true</code> , <code>.false</code> , or <code>unknown</code> .
Belief-value	A number from 0.0 to 1.0, where 0.0 is false and 1.0 is true.
Quality	The status of the path's data. See <a href="#">The Quality Attribute</a> .
Timestamp	The time that the path received its value. For more information, see <a href="#">The Timestamp Attribute</a> .
Collection-time	The time that the data first entered the diagram through an Entry Point, entered G2, or was concluded. See <a href="#">The Collection-Time Attribute</a> .
Expiration-time	The time that the current value for the path expires. For more information, see <a href="#">The Expiration-Time Attribute</a> .

In diagrams, inference paths change color to display the status values they are carrying. This table lists the standard inference path colors:

Status-value	Standard Color
<code>.true</code>	red
<code>.false</code>	green
<code>unknown</code>	white

To change these colors, see [Setting Path Colors](#).

### Filtering Data Passed to Inference Paths

If the value of a variable or parameter connected to an inference path does not change, the variable or parameter does not pass a new value onto the path.



The same thing is not true for data paths or control paths; variables and parameters always pass data values and control signals onto the path, regardless of whether the data value changes.

## Using Control Paths

**Control Paths** carry control signals, which do not have values, but which pass an evaluation signal between blocks.

This table describes the control path attributes:

Attribute	Description
Timestamp	The time that the path received its value. For more information, see <a href="#">The Timestamp Attribute</a> .
Quality	The status of the path's data. See <a href="#">The Quality Attribute</a> .

## Using Item Paths

You use **item paths** with custom blocks to pass any G2 item through a block. Blocks supplied with GDA do not have item paths.

You use item paths for:

- Discrete event processing, for example, to process individual items in an assembly line
- Processing complex data, for example, to process multiple items using an item list or item array

In general, an item path can only have a single item on it at one time. For information on how to process multiple items on an item path, see the *GDA API Reference*.

### Creating Item Paths

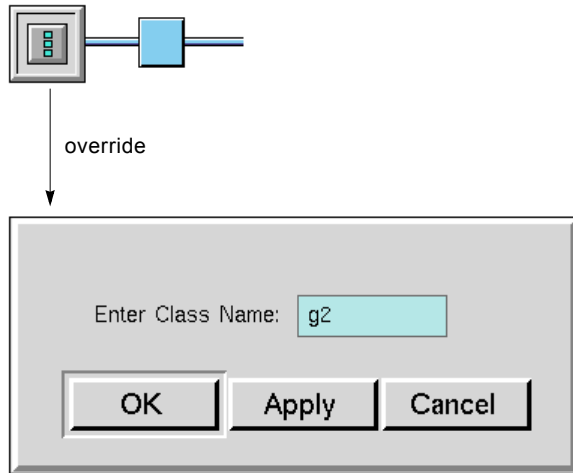
You add items paths to custom blocks by editing the class definition of the block, as described in [Editing an Existing Custom Subclass](#).

### Placing an Item Onto an Item Path Interactively

**To place an item onto an item path interactively:**

- ➔ Connect a symbolic variable or parameter to the item path and manually override its value.

The following figure illustrates a symbolic variable connected to a custom block with an input and output item path. The custom block simply passes the item through the block. Enter the name of an item or a symbol representing an existing class into the Type-in override dialog. GDA places an instance of the item or class onto the item path.




---

**Note** If there is already an item on the output path and that item is transient, overriding a symbolic variable or parameter breaks the G2 relation between the item and the path and deletes the item. If there is an item on the output path and that item is permanent, however, GDA removes the item from the path but does not delete it.

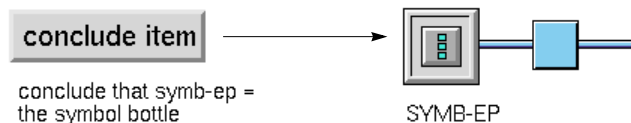
---

## Placing an Item Onto an Item Path Programmatically

**To place an item onto an item path programmatically:**

- ➔ Connect a symbolic variable or parameter to the item path, and use a procedure, a rule, or an action button to conclude a value into the symbolic variable or parameter.

The following figure illustrates a symbolic variable connected to a custom block with an input and output item path. The custom block simply passes the item through the block. The action button concludes a value for the symbolic variable `symp-ep`. The new value is the symbol `bottle`, which represents an object class.



When creating custom blocks, you can also use an API procedure to set an item onto an item path programmatically. In addition, you can define a function that creates a default item for the path when using an API procedure to get an item

from a path. For more information on creating custom blocks using item paths, see the *GDA API Reference*.

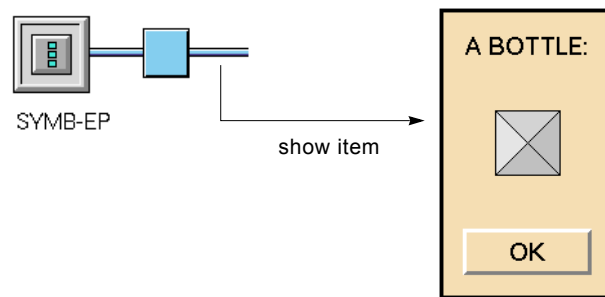
## Displaying the Item on the Path

**To display the item on the path:**

→ Select **show item** from the path's menu.

GDA displays the item in a dialog.

The following figure shows this dialog for a custom block with input and output item paths. The item on the path represents the `bottle` class using the default icon.



If the item on the path is an instance of the `bottle` class, selecting **show item** goes to the original workspace on which the instance is defined and highlights the item with an arrow.

## Resetting Paths

You can reset the values of a specific path or all paths. Resetting the value of a path sets it to the Value on Initialization of the block.

**To reset the value on a particular path:**

→ Select **reset path** from the path's menu.

**To reset the values of all blocks and paths:**

→ Select **Controls > Reset All Blocks**, then choose to reset blocks and paths.

## Using Links

A **link** is a special-purpose type of connection you can use to add a feature or behavior to a block, or to perform an action on a block. For example, you use links to add a chart capability to a block.

GDA provides these types of links:

- Action Links

- Capability Links
- Restriction Links

An **Action Link** connects a block that performs an action on a target block to the target block. For example, you can use a Reset block to reset a target block using a control signal by attaching the target block to the Action Link associated with the Reset block. When the Reset block receives a control signal, GDA resets the target block. Action Links appear on many of the Action blocks.

A **Capability Link** add features to blocks, such as charts and clocks. Capability Links appear on some of the blocks in the Capabilities palette.

A **Restriction Link** customizes a block's capabilities. For example, you can customize the override dialog and determine the source of input for a GDA diagram. Restriction Links appear on some of the blocks in the Capabilities palette.

---

**Note** Although they have the same cross-section, Capability and Restriction Links are not the same. For example, if you create a Capability and then delete it, leaving the Capability Link stub attached to the block, you cannot drag a Restriction Link into the stub. You must delete the Capability Link first and then connect the Restriction using its own stub.

---

**To connect any of the three types of links to a block:**

➔ Select the link attached to a block and click in the middle of the block to which you want to connect the link.

The block to which the link is connected need not have a pre-existing stub.

## Using Connection Posts

Connection Posts enable paths to cross workspace boundaries so you can create diagrams that span several workspaces. G2 passes the data from a named sending connection post to all receiving connection posts having the same name. You can also use connection posts to communicate data within the same workspace.

---

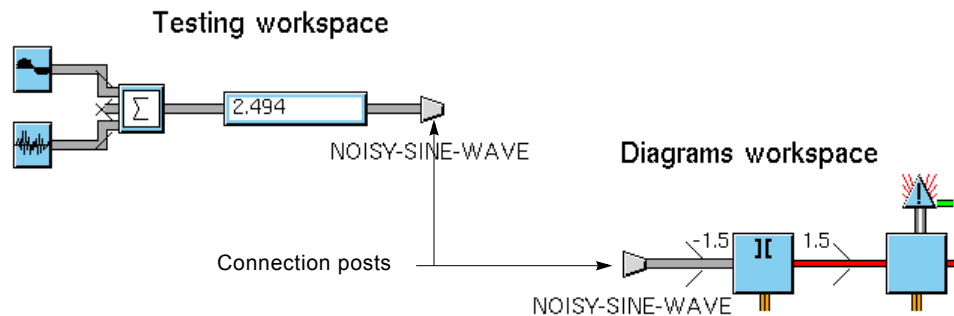
**Note** Unlike other blocks, you specify a name for a connection post by configuring it.

---

GDA enforces these rules for connection posts:

- Connection posts that are communicating data must have the same name.
- More than one connection post cannot send data to the same receiving connection post.
- The different types of connection posts correspond to the different types of paths. You must connect connection posts to paths of the same type.

For example, this diagram passes the noisy sine wave signal from the Testing workspace to the Diagrams workspace using connection posts:

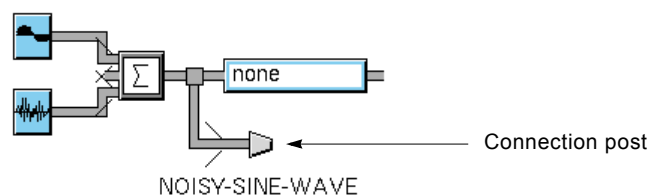


### To create a connection post that connects two workspaces:

- 1 Clone a connection post of the desired type from the Connections palette.
- 2 Connect the input stub of the connection post to the output stub of the block whose value you want to flow to another workspace.
- 3 Drag the output stub of the connection post into the connection post to delete the stub. (It is not necessary to delete the unused stub.)
- 4 Configure the connection post and specify its name.
- 5 Clone another connection post of the same type, and place it on the workspace where you want to receive the data.
- 6 Connect the output stub of the connection post to the input stub of the block on that workspace to which you want data to flow.
- 7 Drag the input stub of the connection post into the connection post to delete the stub. (Again, it is not necessary to delete the unused stub.)
- 8 Configure the connection post and specify the same name as the sending connection post.

### To create a connection post that connects to an existing path:

- 1 Clone a connection post of the desired type from the Connections palette.
- 2 Connect the input stub of the connection post to the path between two blocks whose value you want to flow to another workspace.
- 3 Follow steps 3 through 8 in the previous sequence of steps.



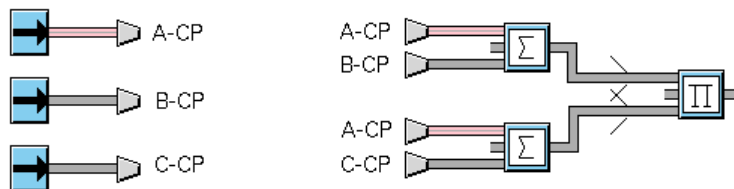
## Highlighting Paths and Connection Posts

If your diagram is large and contains connection posts, you might want to highlight a path to see where it goes. Also, you can find matching connection posts quickly by highlighting them.

### To highlight a path:

- Choose highlight from the path's menu. GDA colors the path using a contrasting color.

This diagram shows the paths connected to the A-CP connection posts highlighted:

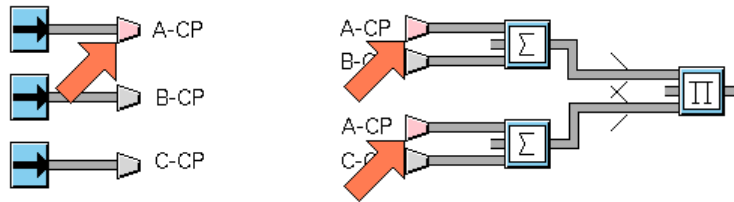


You can also highlight connection posts directly, which also displays large arrows next to the connection posts.

### To highlight a connection post:

- Choose highlight from the connection post's menu. GDA colors the connection post a contrasting color and places an arrow next to each connection post of the same name.

This diagram shows the A-CP connection posts highlighted:



### To stop highlighting a path or connection post:

- Choose do not highlight from the path or connection post's menu.

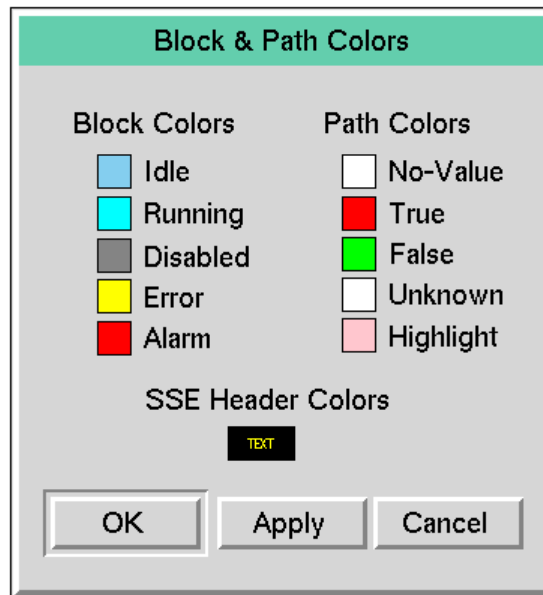
## Setting Path Colors

You can change the colors used to show the status of inference paths in your diagram. For example, inference paths display red when true, green when false, and white when unknown. You can also change the color GDA uses to highlight paths.

When animation is enabled, GDA uses the color preferences you specify for paths. When animation is disabled, GDA uses the colors paths define in their icon descriptions.

**To customize the colors of paths:**

- 1 Select the Preferences > Colors > Blocks & Paths menu choice to display the Block & Path Colors dialog:



- 2 The Path Colors on the right of the dialog control the colors that indicate inference path states and the color of a highlighted path, as described in this table:

Path Color	Description
No-Value	No value on the path.
True	The path has a Status-value of <code>.true</code> .
False	The path has a Status-value of <code>.false</code> .
Unknown	The path has a Status-value of <code>unknown</code> .
Highlight	The path is highlighted with the highlight menu choice.

- 3 Choose a path state to change its color. When you click on the colored square next to the path state, a palette of colors appears.
- 4 Select a color on the palette of colors.
- 5 Continue customizing the colors as desired, and click OK to accept the settings and dismiss the dialog.

- Toggle animation off and then on to see the change by using the Controls > Animate menu choice.

## Using Path Attributes

Each type of path has associated **path attributes** that provide information about the status of the value on the path. This table summarizes the attributes available for the basic types of paths:

Data paths have...	Inference paths have...	Control paths have...	Item paths have...
Quality	Quality	Quality	none
Timestamp	Timestamp	Timestamp	
Collection-time	Collection-time		
Expiration-time	Expiration-time		
Data-value	Status-value		
	Belief-value		

The sections that follow explain the possible values for the Quality, Timestamp, Collection-time, and Expiration-time attributes. For information on the Data-value, Status-value, and Belief-value path attributes, see [Using Data Paths](#) and [Using Inference Paths](#).

### The Quality Attribute

Quality is a system-generated attribute for data, inference, and control paths that specifies the status of the path's data. Quality can have any of the values in the following table:

A Quality value of...	Means...
ok	The value is current or is derived from current data and is not entered manually.
manual	The value is current and is entered manually or is derived from a value that is entered manually. An example is a value you enter using the override menu choice for an entry point.



A Quality value of...	Means...
ok	The value is current or is derived from current data and is not entered manually.
expired	The value is expired based on the Expiration-time attribute.
no-value	The path does not have a value yet. A path has a Quality of no-value before it has received any data, or in some cases, after the block has been reset; thereafter, the Quality will never be no-value again. For an explanation of how different types of blocks handle no-value inputs, see <a href="#">Determining How Blocks Use no-value Inputs</a> .

The order of values for Quality shown in the table above defines the **quality hierarchy**, where **ok** is the highest quality and **expired** is the lowest. GDA uses the quality hierarchy to resolve the output path quality for blocks with multiple inputs, as explained in [Determining Output Path Attributes for Peer Input Blocks](#).

Some blocks optionally require a full history of values before passing a value. These include all of the blocks on the Time Series palette, as well as the Process Capability Index block on the SPC palette. For these blocks, if the attribute Require Full History is **yes**, the output path has a Quality of no-value when the history is not full. For more information on this attribute, see [Specifying What to Do With Partial History](#).

## The Timestamp Attribute

Timestamp is a system-generated attribute for data, inference, and control paths that specifies the current real subsecond time at which a path receives a value. Each time the path receives a new value, Timestamp is updated. Timestamps are unique.

## The Collection-Time Attribute

Collection-time is an attribute for data and inference paths that indicates the time at which data originally either entered the diagram, entered G2, or was concluded.

When variables are used as entry points, GDA uses the G2 current time as the Collection-time. For more information, see [Using Variables and Parameters](#).

For a description of how GDA resolves the Collection-time for blocks with multiple inputs, see [Determining Output Path Attributes for Peer Input Blocks](#).

## The Expiration-Time Attribute

Expiration-time is an attribute for data and inference paths, which you use with G2 variables to determine whether and when the current value for a path is expired.

Expiration-time can have either of the values in the following table:

<b>If Expiration-time is...</b>	<b>Then...</b>
none	The value is always valid.
a number	The value will expire or has expired at the given timestamp.

If Expiration-time is a number, the current value may still be valid, depending on the status of the Quality attribute for the path. If Quality is not expired and Expiration-time is a number, then Expiration-time specifies the time at which the value will expire.

You can suppress the Expiration-time path attribute value to save computational resources or to achieve compatibility with earlier releases. By default, whenever a block's data expires, the block evaluates using the new expired input, which causes downstream blocks to evaluate. In your application, you might prefer not to have expired data cause blocks to evaluate.

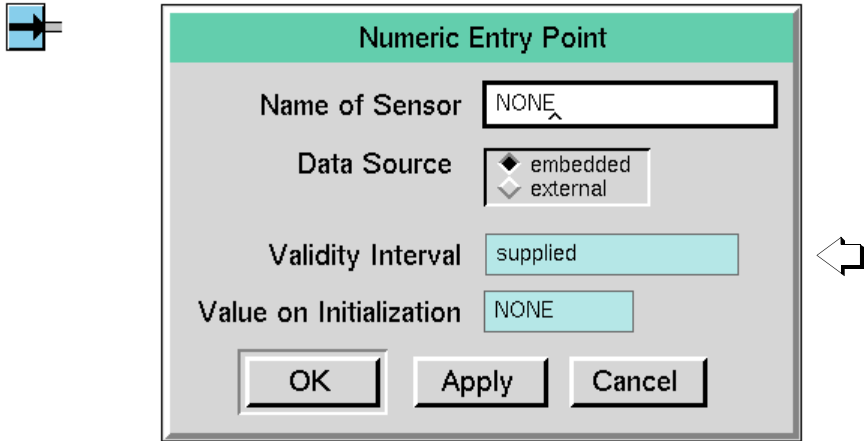
### **To prevent blocks from evaluating when data expires:**

- 1 Select the Preferences > Settings > Environment menu choice to display the Environment Settings dialog.
- 2 Set Propagate Expiration Events to no, then press OK.


### **Specifying Validity Interval for a Variable**

The Expiration-time depends on the Validity Interval, which determines how long the value remains valid. Validity Interval is an attribute for all entry points that you specify in the configuration panel for the block.

This figure shows how to specify Validity Interval for a Numeric Entry Point:



You can also set the Validity Interval for a variable connected to any block, as described in [Using Variables and Parameters](#). To do this, specify Validity Interval in the variable's table:



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

The following table describes the possible values for this attribute:

<b>If Validity Interval is...</b>	<b>Then...</b>
an integer and a time period, e.g., 5 seconds	The value is valid for the specified time period.
supplied	The inference engine computes the value using a rule or procedure.
indefinite	The value is always valid (default).

## Determining How Blocks Use no-value Inputs

Generally, blocks ignore input paths with a Quality of no-value. This has different implications, depending on the type of block.

Peer input data blocks ignore unattached stubs on the block, as well as connected paths that have never received a value. For example, if a Summation block has three input paths but only two of them are connected, the unconnected path has a Quality of no-value and is ignored.

Peer input logic blocks behave the same way as peer input data blocks, ignoring unattached stubs on the block, as well as connected paths that have never received a value.

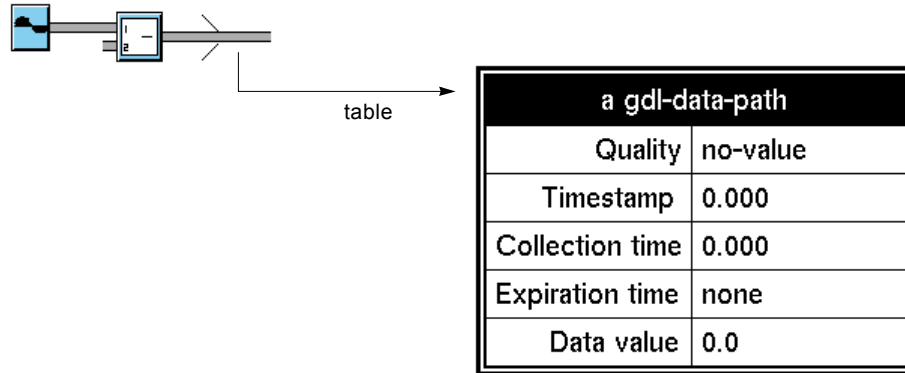
---

**Note** Peer input logic blocks *can* have an inference output path with a Status-value of unknown and a Quality other than no-value, e.g., ok. However, the only time a peer input logic block has an inference output path with a Status-value of unknown and a Quality of no-value is before the block receives a value.

---

Some non-peer input blocks, such as the Event Window Gate, do not require all of their inputs to evaluate. So, if one or more inputs have a Quality of no-value, the block can still evaluate.

Non-peer input blocks that require all of their inputs to evaluate do not place a value onto an output path if the block has a no-value input. For example, a Difference block can only evaluate if both of its input paths are specified. If this Difference block has an input whose Quality is no-value, the block does not evaluate.



## Determining Output Path Attributes for Peer Input Blocks

The attributes of the input paths for peer input blocks determine the attributes of the output path. This table summarizes the general rule for resolving output path attribute values for peer input blocks:

This attribute...	Is resolved in this way...
Quality	The lowest quality of all of the input paths based on the quality hierarchy (see <a href="#">The Quality Attribute</a> ).
Collection-time	The maximum collection time of all of the input paths, i.e., the most recent collection time.
Expiration-time	The minimum expiration time of all of the input paths, i.e., the expiration time of the first path to expire.

For custom blocks, GDA resolves output path attribute values using the API procedures described in the *GDA API Reference*.

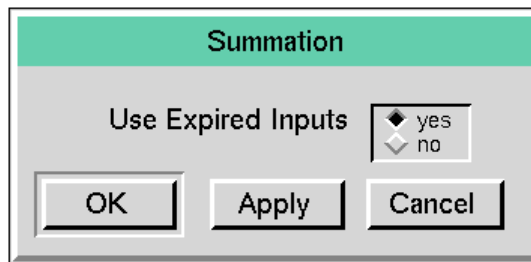
### Determining Whether a Block Uses Expired Inputs

You can determine whether a peer input block includes expired inputs in its internal calculation. For example, you can ignore input paths for a Summation block whose Quality attribute is expired. In this case, the resulting output path would never have a Quality of expired.

All peer input blocks include the following attribute in their tables for controlling whether expired data is to be included in the block's calculation:

The attribute...	Determines...
Use Expired Inputs	Whether to include expired values in the block's internal calculation, i.e., inputs with a Quality of expired. The default value is yes.

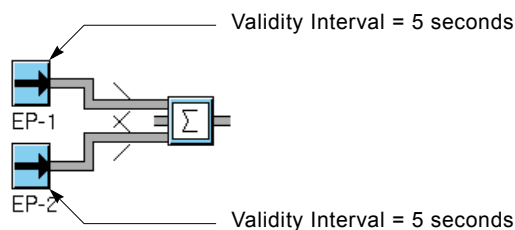
This figure illustrates the configuration dialog box for a Summation block, which shows the default value for Use Expired Inputs:



If all the inputs to a peer input block are expired, and Use Expired Inputs is no, the block does nothing.

### Example of Determining Path Attributes Using a Peer Input Block

Suppose you have two Entry Points connected to a Summation block. Each Entry Point uses an embedded variable, which specifies Validity Interval as 5 seconds.



The following figure compares the tables for the two input connection paths with the table for the output connection path for the Summation block whose Use Expired Inputs is **yes**. Notice that the Expiration-time of the Summation block's output path is the minimum of the two input Expiration-time attributes, and the Quality of the Summation block's output path is **expired**, which is the lowest quality of all of the input path qualities.

dp-out of EP-1

a gdl-data-path	
Quality	expired
Timestamp	3293.512
Collection time	3292.950
Expiration time	3297
Data value	2

dp-out of EP-2

a gdl-data-path	
Quality	ok
Timestamp	3297.526
Collection time	3297.250
Expiration time	3302
Data value	10

dp-out of Summation block

a gdl-data-path	
Quality	expired
Timestamp	3297.555
Collection time	3297.250
Expiration time	3297
Data value	12.0

The following figure shows the same tables when Use Expired Inputs is no. In this case, GDA ignores the input path associated with EP-1 in the calculation of the sum because its value has expired; thus, the output path attributes are identical to the EP-2 input path attributes.

a gdl-data-path	
Quality	expired
Timestamp	3897.512
Collection time	3897.450
Expiration time	3902
Data value	3

a gdl-data-path	
Quality	ok
Timestamp	3905.519
Collection time	3905.300
Expiration time	3910
Data value	3

a gdl-data-path	
Quality	ok
Timestamp	3905.545
Collection time	3905.300
Expiration time	3910
Data value	3.0

## Creating Customized Path Connections

You can create customized path connections to pass particular types of data for use with custom blocks. For example, you could create a customized item connection path for passing lists, arrays, or any other defined item class.

The reason for doing this is to prevent blocks with different path types from being connected.

### Creating a New Connection Subclass

The first step in creating a customized connection path is to create a new subclass of connection based on an existing GDA connection class. Typically, you create a new connection subclass based on the `gdl-item-path` connection class. However, you can also create customized connection subclasses based on `gdl-data-path`, `gdl-inference-path`, `gdl-action-link`, or `gdl-path`.



---

**Note** The name of the new connection post subclass must be of the form *connection-subclass-name-connection-post*, e.g., *custom-item-path-connection-post*, in order to be recognized by GDA when using with blocks.

---

**To create a new connection subclass:**

- 1 Click on the background of a workspace, and select KB Workspace > New Definition > class-definition > connection-definition.
- 2 Choose table from the connection definition to display its table.
- 3 Specify a unique symbol for Class-name.
- 4 Specify the Direct-superior-classes as *gdl-item-path*.

For example, the following figure shows the result of creating a new connection subclass for passing arrays:



FLOAT-ARRAY-PATH

FLOAT-ARRAY-PATH, a connection-definition	
Notes	OK
Authors	mcooperm (17 Oct 2000 1:49 p.m.)
Change log	0 entries
Item configuration	none
Class name	float-array-path
Direct superior classes	gdl-item-path
Class specific attributes	none
Instance configuration	none
Change	none
Class inheritance path	float-array-path, gdl-item-path, gdl-path-without-junctions, gdl-path, gdl-connection, connection, item
Inherited attributes	path-item-id is a text, initially is "", with an index
Attribute initializations	none
Cross section pattern	inherited
Stub length	inherited
Junction block	junction-block-for-float-array-path

## Customizing the Connection Path Regions

GDA can only animate GDA-defined connection path regions. Therefore, to customize the existing regions, specify new colors for the GDA-defined connection path region names in the **Cross-section-pattern** attribute. If you want to create new regions, you must define the custom regions in addition to the GDA regions.

The following table shows the names of the GDA-defined regions for each type of connection path:

The GDA class...	Defines these named regions...
gdl-data-path	sides, center, type
gdl-item-path	sides, center, light, dark
gdl-inference-path	sides, center
gdl-action-link	sides, center
gdl-path	sides, center

For example, to change the default color for the **sides** region of a custom connection path based on **gdl-data-path**, specify the attribute **Cross-section-pattern** as follows:

```
sides = blue, center = white, type = gray
```

To add new regions, you must specify the existing GDA regions first, followed by the new regions. The new regions can refer to the GDA-defined regions as colors:

```
sides = black, center = white, type = gray;
1 sides, 2 blue
```

## Adding Custom Connections to a Custom Subclass

To add custom connections to a custom subclass block, use G2 to edit the class definition of the custom class to use the custom connections.

### To add custom connections to a custom subclass:

- 1 Select Main Menu > Inspect to go to the class definition of your custom class, for example, go to **power-block**.
- 2 Use G2 to edit the **Stubs** attribute of the class definition to refer to the custom connection class, or any one of GDA's existing connection classes.

# Specifying Initial Values

Certain blocks let you specify the value they pass when you first start G2 or when you reset the block. This value is called the **initial value**.

Blocks that define these attributes pass initial values even if data input is not enabled.

## Specifying an Initial Data Value

You can set the initial value for blocks that pass data values.

**To set the initial value for data blocks:**

➔ Set the attribute Value on Initialization to one of these values:

When Value on Initialization is...	Then the block does this...
a number, string, or symbol	Sets the output path's dp-out to the value and sets the Quality of its output path to ok.
no-value	Sets the Quality of the output path to no-value and leaves the output path's dp-out unchanged.
none	Leaves the output path and dp-out unchanged.

## Specifying an Initial Control Value

The Control Entry Point lets you specify how many control signals it passes when you first start G2 or reset the block.

**To specify the number of control signals to send:**

➔ Set the attribute Value on Initialization to one of these values:

When Value on Initialization is...	Then the block does this...
an integer value	Passes this number of control signals on its output path.
no-value or none	Passes no control signals.

The Control Entry Block is the only Action block with this attribute.

## Specifying an Initial Status Value

You can set the initial value for certain blocks that pass inference values. You can also set the initial value for Belief Entry Points.

**To set the initial value for an inference block:**

→ Set the attribute Status on Initialization to one of these values:

When Status on Initialization is...	Then the block does this...
true, false, or unknown	Sets ip-out to the value and sets the Quality of its output path to OK.
none	Leaves the output path and ip-out unchanged

## Maintaining a History of Values

Many blocks can store input values and let you perform operations on this **history** of values. A history can store either data or inference values. Blocks can perform these operations on their history of values:

- Statistical computations on a history of data values. For example, the Moving Average block in the Time Series palette enables you to compute the average of the history of values. Other blocks in this palette enable you to perform other computations on the history of values.
- Operations on a history of inference values. For example, the Average Belief Gate block in the Temporal Gates palette enables you to compute the average of the history of inference values. Other blocks in this palette enable you to perform other operations on the history of inference values.
- Test whether values are within or outside a specified range of values. The In Range Pattern and Out of Range Pattern blocks in the Observations palette perform these operations.

You can control how many values are retained in the block history. You can also control how often the block performs its operation and propagates its result. For example, the Moving Average block can be defined to compute the average of 100 points and propagate the computed average every time it receives a new point, or when it receives every 100th point, or at some other interval.

This section describes how to set the size of the history, and how the block uses the history.

## How the History Feature Works

Each block that supports the history feature uses several attributes to define how it is used: Sample Type, Sample Size, Update Type, Update Size, Erase History on Reset, and Require Full History.

When a value arrives at a block that maintains a history of values, these steps occur:

- 1 GDA determines whether the value is monotonic. A value is monotonic if its collection time is newer than the collection time of the most recent value in the history. For more information, see [How GDA Handles Nonmonotonic Values](#).
- 2 If the value is monotonic, GDA determines whether any values in the block's history are to be deleted. The Sample Type and Sample Size attributes determine the number of values maintained in the block's history. For more information, see the next section, [Specifying the Size of the History](#).
- 3 GDA next determines whether the block gets evaluated. The Update Type and Update Size attributes determine when the block evaluates. For more information, see [Specifying When to Propagate Data](#).

Two other attributes affect the block history. The Erase History on Reset attribute determines what happens to the block history when the block is reset. Finally, the Require Full History attribute determines whether the block evaluates when the history is not complete.

All these steps and attributes are described in the sections that follow.

## Specifying the Size of the History

You can choose whether the history of values is to contain a specific number of values or all values arriving at the block within a specific amount of time. Also, you can define the history as containing all points arriving at the block since the block was last reset.

The Sample Type and Sample Size attributes determine how the history is defined. These attribute values are described in this table:

If Sample Type is...	The history is defined as...
points	A specified number of data or inference values (points). The Sample Size attribute specifies the number of points.
time	All points received within a specified period of time. The Sample Size attribute specifies the number of seconds.
fixed	If the Erase History on Reset attribute is true, the history is all the points received since the block was last reset. Otherwise, the history is all the points received since G2 was started. The Sample Size attribute is ignored.

For example, if Sample Type is **points** and Sample Size is **60**, the block stores the most recent 60 points. If Sample Type is **time** and Sample Size is **60**, the block stores the points received in the last 60 seconds.

### How GDA Handles Point-Based Histories

When the Sample Type for a block is **points**, the block collects its input points, storing both the value and collection time, until the number of points collected is equal to the Sample Size attribute value. When another input value arrives at the block, the oldest point is discarded. The history never contains more than the specified number of points.

### How GDA Handles Time-Based Histories

When the Sample Type for a block is **time**, the block collects its input points, storing both the value and the collection time. As each point is received, its collection time ( $T_{\text{newest}}$ ) is compared to the collection time of the oldest value in the history ( $T_{\text{oldest}}$ ) and the difference is compared to the Sample Size (SS):

- If any values in the block history are older than the newest value by more than Sample Size seconds, they are removed from the history:

If  $T_{\text{newest}} - T_{\text{oldest}} > SS$ , discard oldest point (repeat as necessary)

- If the oldest value is less than the Sample Size number of seconds older than the most recent value, the oldest value is retained in the history:

If  $T_{\text{newest}} - T_{\text{oldest}} \leq SS$ , retain oldest point

The history contains all points whose collection times fall within this range:

$[T_{\text{newest}} - SS, T_{\text{newest}}]$

---

**Note** Blocks accept nonmonotonic values (values whose collection times are earlier than the current time). For more information, see [How GDA Handles Nonmonotonic Values](#).

---

### **How GDA Handles Histories When the Sample Time is Fixed**

When the Sample Time is fixed, the block accumulates values in its history until either of these occurs:

- G2 is restarted.
- The block is reset (if the Clear History When Reset attribute is `true`; otherwise, resetting the block has no effect on the history).

---

**Caution** Depending on the application, if G2 can run indefinitely or if resetting the block depends on the occurrence of an unpredictable event, using a Sample Time of fixed can have a destructive effect on the application. Data accumulated by the block is stored in memory, and unrestricted accumulation of values in the block history can cause G2 to shut down. Instead, use the Sample Time settings of either points or time and set relatively large values of the Sample Size attribute.

---

### **Performance Issues**

Storing a history of values defined by a Sample Time of points is faster than when the history is defined by a Sample Time of time. When Sample Time is time, the collection time for each new point must be compared to the oldest point in the history; if the oldest point is discarded, the collection time for the new point is then compared to the next-oldest point, until no more points are deleted.

### **Deciding Which Sample Time Option to Choose**

Generally, determining whether to use a point-based or a time-based history depends on the application. Here are some suggestions:

- Setting the Sample Size attribute depends on how many points determine a significant sample size and how comfortable you are discarding data.
- If inputs are relatively regular (arrive at regular or almost regular time intervals), use points.



- If inputs are irregular, the choice may not be obvious. You may need to decide which option to use based on how meaningful old data is or how useful computations are that are based on few points.
- If you know you want to examine all data gathered within a fixed amount of time (such as the duration of a shift), use time and specify the number of seconds. This table can be useful:

<b>Hours</b>	<b>Seconds</b>
1	3600
2	7200
4	14400
8	28800

- Unless you reset blocks frequently, avoid using the fixed option.

## Specifying When to Propagate Data

You can control how often the block performs its operation and propagates the result using the Update Type and Update Size attributes. These attribute values are described in this table:

<b>If Update Type is...</b>	<b>The block performs its operation and passes a value...</b>
points	Every Update Size points it receives.
time	After Update Size seconds passed since the last time the block evaluated and the block receives a new value.

### How GDA Handles Point-Based Updates

GDA handles point-based updates by evaluating the block when it has accumulated Update Size points since the block last evaluated.

For example, if Update Type is `points` and Update Size is 10, the block performs its operation and passes one value after it has received 10 points, another value after it has received 20, and so on.

## How GDA Handles Time-Based Updates

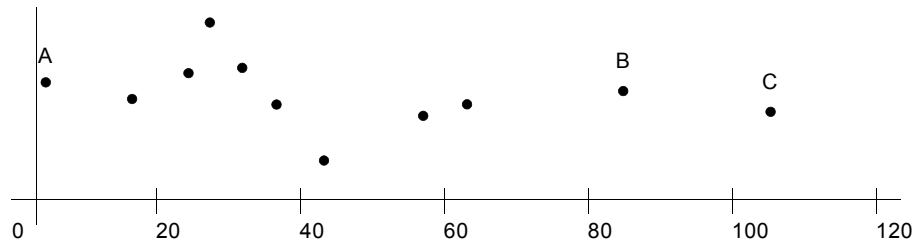
GDA handles time-based updates in a more complicated way. Each time the block evaluates, it stores the current time ( $T_{\text{current}}$ ). The block then computes the next time the block expects to evaluate:

$$T_{\text{next-eval}} = T_{\text{current}} + \text{Update Size}$$

Then, when the block receives an input value, it examines its collection time ( $T_{\text{newest}}$ ) and determines whether the block is to evaluate:

- If  $T_{\text{newest}} < T_{\text{next-eval}}$ , the block does not evaluate.
- If  $T_{\text{newest}} \geq T_{\text{next-eval}}$ , the block evaluates.

This figure illustrates how GDA determines which values in the block history are used in the block's computation when Sample Type is points, Sample Size is 10, Update Type is time, Update Size is 100, and new points enter the block:



As points enter the block, the block collects each data value and its collection time and checks to determine whether any points are to be deleted from the block history. Point B arrives at the block with a collection time of 85. Point B is the 10th point in the block history, so the history is now full (the Sample Size is 10). The block does not evaluate because fewer than 100 seconds have elapsed. Then, point C arrives at the block, with a collection time of 105. Because point C is the 11th point, point A is deleted from the history. Because the Update Size is 100, the block evaluates and stores the current time, 105.

The next data point that causes the block to evaluate is the first value having a collection time  $\geq 205$  (the last time the block evaluated + 100, the Update Size).

## Performance Issues

Normally, blocks with history perform their computation on all the values in the history each time they evaluate. For some blocks, another algorithm provides more efficient computations.

Whenever exactly one point is added to the block history since the last evaluation and either zero or one point is deleted from the history, these blocks use the alternative algorithm.

When GDA uses the standard algorithm, increasing the Update Size produces greater average execution speed. However, if increasing the Update Size causes a switch from the new algorithm to the standard algorithm, average execution time can increase. Computation times depend on both the type of block and the total number of points in history.

These blocks and gates use the new algorithm: Average Belief gate, Integrator block, Max Belief gate, Median Input Value block, Min Belief gate, Moving Average block, Moving Range block, Process Capability Index block, Root Mean Square (RMS) block, and Variance block.

## Specifying What Happens to History Upon Reset

When a block is reset, you can choose to retain or delete all values in the block's history.

For example, you have built a diagram to monitor the output of a manufacturing process. The diagram contains a Variance block that passes on the standard deviation of all points received each hour. You notice that after some time, the standard deviation has begun increasing, so you decide to halt the process to correct the problem. You direct the operator to disable evaluation until the correction is made, then reset the Variance block. In this situation, any data held in the Variance block's history are not useful for the calculation of the next standard deviation, so you choose to erase the block history when the block is reset.

**To specify what happens to a block's history when you reset the block:**

➔ Set the Erase History When Reset attribute according to this table:

<b>If Erase History When Reset is...</b>	<b>The block does this...</b>
yes	Erases its history when you reset.
no	Retains the values in its history when you reset.

## Specifying What to Do With Partial History

When a block evaluates, it performs its operation based on the points in its history. When an input value is received that causes the block to evaluate, the block has **full history** or **partial history** depending on these factors:

- If Sample Type is **points**, if the history contains the number of points specified by the Sample Size attribute, the block has a full history. Otherwise, the block has a partial history.
- If Sample Type is **time**, if the history contains values whose range of collection times meets or exceeds the amount of time specified by the Sample Size attribute, the block has a full history. Otherwise, the block has a partial history.

You can specify whether or not the block is to evaluate only with a full history.

### To specify whether a block computes a value only with a full history:

➔ Set the Require Full History attribute according to this table:

If Require Full History is...	The block does this...
yes	Evaluates and passes a value only when it has a full history.
no	Evaluates and passes a value even if it has a partial history.

For example, suppose your application has a block with these attributes:

- The Sample Type is **points** and the Sample Size is **10**.
- The Update Type is **time** and the Update Size is **60 seconds**.
- A point is received by the block about once every 10 seconds.

The block is scheduled to evaluate for the first time when it receives a value after 60 seconds have passed. At 60 seconds, the block history contains only 6 points.

- If Require Full History is **no**, the block passes a value computed using those six points. The block next evaluates after another 60 seconds has passed and the block receives another point.
- If Require Full History is **yes**, the block does not pass a value because the history is full when it contains 10 points. Each time the block receives a new value, GDA determines whether the block evaluates. In this example, the block evaluates for the first time when it has received 10 points.

## How GDA Handles Nonmonotonic Values

A nonmonotonic value is a value that has a collection time that is less than or equal to the collection time of a value already in the block's history. History blocks accept nonmonotonic values and insert these points into history, marking them with their collection time.

If an existing value has the same collection time as a new value, the new point replaces the existing one and does not increment the count of points in the history.

If no existing value has the same collection time as the new point, the new point is inserted into the proper place in the history and the count of points in the history is incremented. Depending on the Sample Type, GDA may take further action:

- If the Sample Type is **points** and the new value causes the number of value to exceed the Sample Size, the oldest value is deleted.
- If the Sample Type is **time** and the new value is older than the current time minus the Sample Size, the new value is discarded.

If the new point causes the block to evaluate, the block propagates its output with the collection time of the latest data point, regardless of the order of the collection times of the data points in history.

## Specifying How to Handle Multiple Values

Some blocks let you choose what they do when they receive an input value while they are already executing. These are called **multiple values**.

**To specify how to handle multiple control signals:**

- Specify a value for the attribute Multiple Invocations.

The table below lists the possible values for Multiple Invocations:

<b>If Multiple Invocations is...</b>	<b>The block does this when it receives a control signal and it is already executing...</b>
ignore	Discards the incoming values. The block executes only when it's idle and it receives an input value.
queue	The block increments the count of waiting values. When it's finished executing, it checks whether that counter is one or greater. If so, it decrements the counter and executes again, using the values on the input paths at the execution time.
ok	Allows simultaneous executions. The block increments the count of invocations running and starts another process to execute for it. When an execution finishes, the block decrements that count.

Generally, the blocks with the Multiple Invocations attribute are those that may take a significant amount of time to evaluate, such as the Data Delay and Control Counter.

## Specifying and Generating Explanations

Many GDA inference blocks enable you to define and display descriptions of their current output values. For example, you could specify that when a block passes true, its description is "the temperature is too high" and when the block passes false, its description is "the temperature is OK".

You can specify descriptions for most inference blocks, the Inference Output block (Control Action), the Belief Displays (Path Displays), the Belief Transmitter (Network Interfaces) and the Belief Entry Point and Belief Network Entry Point.

GDA uses these descriptions in these cases:

- As a clause in an explanation generated by the menu choices **current explanation**, **explain alarm**, **explain alarm absence**, **explanation of last true**, **explanation of last false**, and **explanation of last unknown**.
- To describe choices in a dialog box, such as the override dialog (the dialog is described in [Overriding a Discrete Inference Block](#))

The next two sections describe how to specify **descriptions** for blocks and how GDA uses these descriptions to generate **explanations** for a diagram's state.

## Specifying an Explanation

This section shows you how to define and display these descriptions for the blocks listed above.

- The attributes that describe why a block is passing a true, false, or unknown value.
- The attribute that describes a block's input value.

This section also describes how to concatenate text in these descriptions, to provide lengthy and precise explanations of complex conditions.

### How to Describe Why a Block Passes True, False, or Unknown

To specify the description:

- 1 Select configure from the block's menu.
- 2 Select Descriptions on the block's configure dialog.
- 3 On the Descriptions dialog, specify string values for one or more of the attributes When True, When False, and When Unknown that describe why the block is passing a specific value. The dialog looks like this:

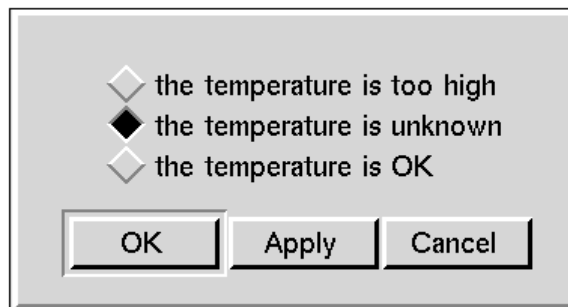
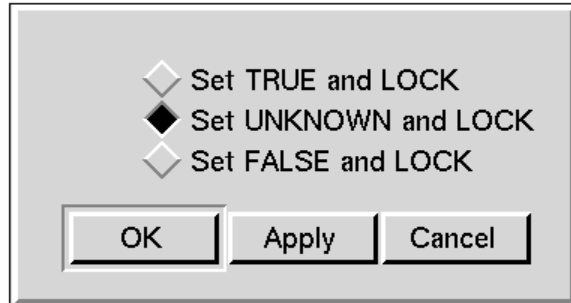
The image shows a dialog box titled "Descriptions". It contains four text input fields. The first field, labeled "When True", contains the character "^". The other three fields, labeled "When False", "When Unknown", and "Input", are empty. At the bottom of the dialog are three buttons: "OK", "Apply", and "Cancel".

By convention, the string is a short sentence with no capitalization or punctuation. Some examples are "the temperature is too high", "engine rpm exceeds recommended maximum", or "pressure of T128 is over the safe limit".

GDA uses these strings to create an explanation for a block's value. For example, if a High Value block has the When True attribute set to "temperature is too high" and a Conclusion connected to that block has a When True set to "tank may overheat", GDA produces this explanation when the Observation passes true:

tank may overheat because temperature is too high

GDA also uses these strings to describe the choices in many dialog boxes, including the override dialog. For example, this figure shows both unmodified and modified override dialogs:



### **Describing the Block's Input Value**

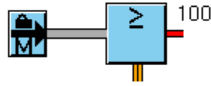
**To describe the block's input value:**

- 1 Select configure from the block's menu.
- 2 Select Descriptions on the block's configure dialog.
- 3 Set the Input attribute to a string that describes the block's input value.

By convention, the string is a lowercase, unpunctuated, descriptive noun; for example, "temperature", "engine rpm", or "pressure of T128".



GDA uses this attribute to create explanations for a block that does not have values for the When True, When False, and When Unknown attributes. For example, in this model, a temperature of 109 is passed to the High Value block. When you select **current explanation** from the High Value block's menu, GDA displays the Explanation Queue:



1 Explanation Queue <span style="float: right;">✖</span>	
Time*	Explanation Text
10/12/2000 11:17:04	12 Oct 2000 11:17:04.1 a.m. GDL-HIGH-VALUE-OBSERVATION-XXX-137 is TRUE because temperature = 109 (threshold: 100)

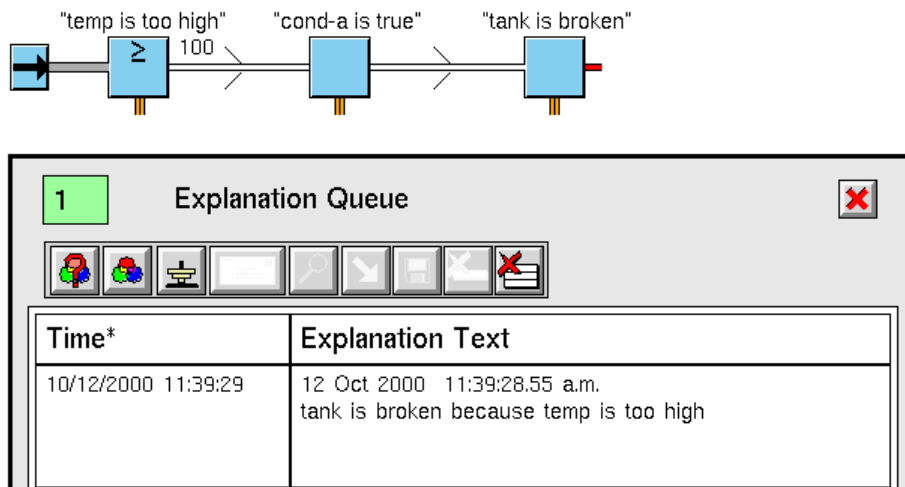
## Generating Explanations

GDA can produce explanations for an inference block's output value by examining your diagram. GDA uses the When True, When False, and When Unknown attributes to generate these explanations, and connects those clauses with the words and, or, and because.

### To generate the explanation:

- ➔ Select current explanation from the block's menu.

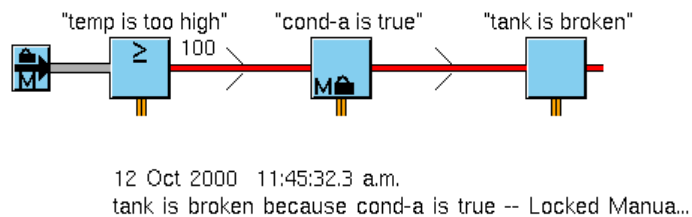
This figure shows how GDA creates a simple explanation that contains clauses associated with the High Value block and the right-most Condition block:



The first clause (**tank is broken**) describes the output value of the block. The second clause (**temp is too high**) explains the output value. The explanation contains the descriptions of the blocks that are upstream of the block you are explaining.

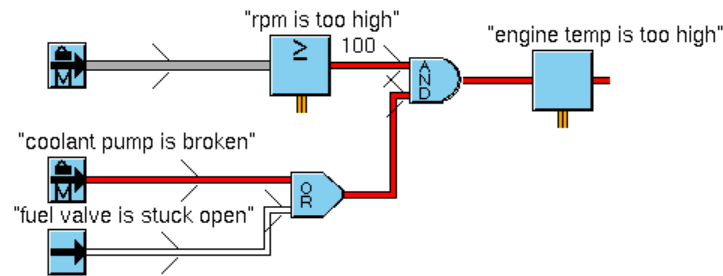
To generate the explanation, GDA searches upstream as far as it can for Conditions connected to the block being explained. It stops searching when it reaches the beginning of the inference path: a Condition without an inference input, a locked Condition, or a Condition with a Local Explanation Restriction. The block at the beginning of the inference path has the description **temp is too high**.

This figure shows a diagram in which a Condition block is locked. GDA creates the explanation from the locked block and not from the upstream High Value block. The locked block's When True description is marked "Locked Manually."



The next figure shows a more complicated explanation. Three paths lead to the block whose explanation is being displayed, so the explanation has four clauses: one clause for the block being explained, and one clause for each block upstream in the inference path. GDA combines the clauses with the words **and** and **or** because the paths are combined with an AND Gate and an OR Gate. To find out

how GDA produces descriptions for logic gates, see [Creating a Description for Logic Gates](#).



12 Oct 2000 12:06:36.2 p.m.  
 engine temperature is too high because (coolant pump is  
 broken -- Locked Manually  
 and  
 rpm is too high)

## Generating Explanations for Most Blocks

When GDA needs a description for an Observation, Conclusion, or Entry Point, it goes through the following sequence until it finds a method it is able to use:

- 1 Use the description attributes.** If the description attribute for the block's output value is defined, use that description attribute. The description attributes are When True, When False, and When Unknown.
- 2 Use the Input attribute.** If the block is from the Observation palette and its Input attribute (on the Descriptions dialog) is defined, generate an explanation from that attribute, its threshold or range, and its input value.

For example, if a High Value Observation has an Input of "temperature", Threshold of 100, and input value of 109, it generates this explanation:

temperature = 109 [threshold = 100]

And if an Out of Range Value Observation has an Input of "level", Lower Threshold of 50, Upper Threshold of 75, and input value of 47, it generates this explanation:

level = 47 [range: 50 to 75]

- 3 Use the block's name.** If the block has a name, generate an explanation of the form "*name is value*", where *name* is the name of the block and *value* is TRUE, FALSE, or UNKNOWN.

For example, if a Conclusion is named tank-broken and its output value is true, it generates this explanation:

TANK-BROKEN is TRUE

- 4 Use the block's type.** Otherwise, generate an explanation of the form "a *type* is *value*", where *type* is the type of the block and *value* is TRUE, FALSE, or UNKNOWN.

For example, if an unnamed Conclusion passes true, it generates this explanation:

A GDL-CONCLUSION is TRUE

If a block cannot provide an explanation, then GDA uses the clause "unable to derive an explanation". One case in which this occurs is if a non-Condition inference block is at the beginning of an inference path.

### Creating a Description for Logic Gates

When GDA needs to create an explanation for a Logic Gate, it uses the words and and or to combine the explanations from the inference paths connected to the gate.

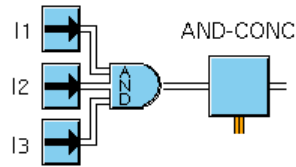
#### To generate the explanation:

- ➔ Select explanation of last true, explanation of last false, and explanation of last unknown from the logic gate's menu.

The following table describes how GDA combines the explanations. The gate's explanation does not contain explanations from all attached paths. Also note that the gate combines the explanations differently depending on the Gate's output value.

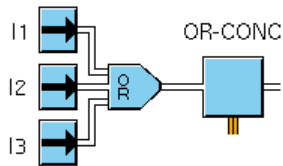
If this gate...	Passes...	It joins its...	With...
AND or N True	true	true inputs	and
AND or N True	false	false inputs	or
OR	true	true inputs	or
OR	false	true inputs	or
EOR	true	true input	none
EOR	false because more than one input is true	true inputs	and
EOR	false because no input is true	false inputs	and
AND, N True, OR, or EOR	unknown	unknown inputs	and

For example, the following table lists some explanations that the AND-CONC block could produce:



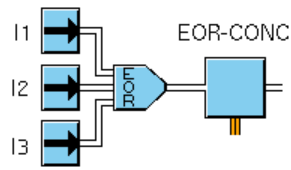
If I1 is...	and I2 is..	and I3 is..	The explanation is...
true	true	true	AND-CONC is TRUE .because. (I1 is TRUE .and. I2 is TRUE .and. I3 is TRUE)
false	true	false	AND-CONC is FALSE .because. (EP1 is FALSE .or. I2 is FALSE)
unknown	unknown	true	AND-CONC is UNKNOWN .because. (I2 is UNKNOWN .and. I3 is UNKNOWN)

The following table lists some explanations that the OR-CONC block could produce:



If I1 is...	and I2 is..	and I3 is..	The explanation is...
true	false	true	OR-CONC is TRUE .because. (I1 is TRUE .or. I3 is TRUE)
false	false	false	OR-CONC is FALSE .because. (I1 is FALSE .and. I2 is FALSE .and. I3 is FALSE)
false	unknown	unknown	OR-CONC is UNKNOWN .because. (I2 is UNKNOWN .and. I3 is UNKNOWN)

The next table lists some explanations that the EOR-CONC block could produce:



If I1 is...	and I2 is..	and I3 is..	The explanation is...
true	false	true	EOR-CONC is FALSE .because. (I1 is TRUE .and. I3 is TRUE)
false	false	false	EOR-CONC is FALSE .because. (I1 is FALSE .and. I2 is FALSE . and. I3 is FALSE)
true	false	false	EOR-CONC is TRUE .because. I1 is TRUE
false	unknown	unknown	OR-CONC is UNKNOWN .because. (I2 is UNKNOWN .and. I3 is UNKNOWN)

## Specifying Fuzzy Logic Attributes

Most GDA inference blocks can use either **fuzzy** or **discrete logic**. These sections describe how to choose which type of logic a block uses, and how to set attributes that control how blocks convert fuzzy belief values to discrete status values.

### Specifying the Type of Logic to Use

Most inference blocks let you choose whether they perform fuzzy or discrete logic.

#### To determine whether a block uses discrete or fuzzy logic:

→ Specify the Logic attributes as **discrete** or **fuzzy**.

In discrete logic, a block passes a **Status-value** of **.true**, **.false**, or **unknown** and a **Belief-value** of 1.0, 0.0, or 0.5, respectively. In fuzzy logic, a block passes a **Status-value** of **.true**, **.false**, or **unknown** and a **Belief-value**, which is a number between 0.0 and 1.0, where 0.0 is completely false and 1.0 is completely true.

For example, suppose the output value of an Entry Point tells you how true this statement is: It is raining outside. The following table lists the possible belief and

status values for the Entry Point and what they mean. Note that the belief value gives you more information than the status value. When the status value is `.true`, there could be a gentle rain or a downpour. But the belief value assigns `0.75` to a gentle rain and `1.0` to a downpour.

If the Belief-value is...	And the Status-value is...	The weather is...
0.0	<code>.false</code>	Sunny
0.25	<code>.false</code>	Cloudy
0.5	unknown	Drizzling
0.75	<code>.true</code>	Raining
1.0	<code>.true</code>	Downpour

All inference blocks pass a status value and a belief value, regardless of the value of the attribute Logic. The block computes its belief value then chooses the appropriate status value.

- If Logic is **discrete**, the belief value is either `0.0`, `0.5`, or `1.0`. The block assigns the status value `.true` to `1.0`, `unknown` to `0.5`, and `.false` to `0.0`.
- If Logic is **fuzzy**, the belief value can be any number from `0.0` to `1.0`. The block uses the attribute Output Uncertainty, described in [Specifying Uncertainty](#), to decide the status value.

You can use two-stage logic in GDA. Generally, GDA uses three-stage logic: a block can pass `.true`, `.false`, or `unknown`. However, if a block has Logic set to **discrete** and Output Uncertainty set to `none`, the block passes `.true` or `.false` only, and not `unknown`. It assigns `.false` to the belief value `0.5`.

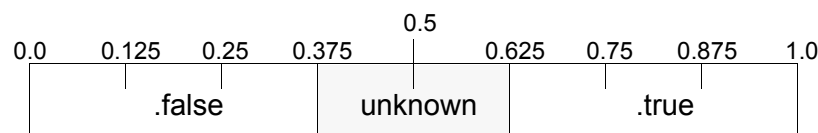
## Specifying Uncertainty

An inference block computes a belief value from `0.0` to `1.0` and then uses that value to compute its status value (`.true`, `.false`, or `unknown`). You can specify an **uncertainty** to determine the range of belief values that result in each status value.

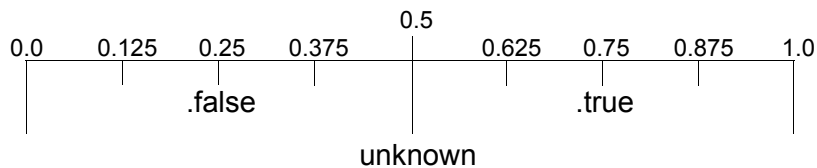
### To determine where to put the boundaries between the status values:

- ➔ Set the attribute Output Uncertainty, which defines a band around `0.5` that will have the status value `unknown`.

For example, in the first figure, Output Uncertainty is `0.25`:



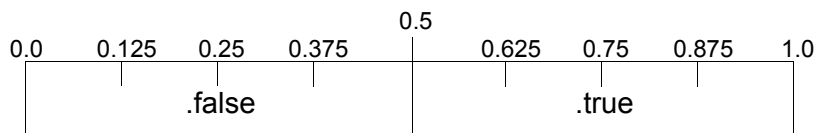
If you set Output Uncertainty to 0.0, a Belief-value of 0.5 is associated with a Status-value of unknown, as shown in this figure:



**To prevent the block from passing the value unknown:**

➔ Set Output Uncertainty to none.

A Belief-value greater than or equal to 0.5 corresponds to a Status-value of true, and a Belief-value less than 0.5 corresponds to a Status-value of false, as this figure shows:



## Specifying Hysteresis

A Condition block can filter out minor changes in its status value. If a block frequently changes to unknown from true or false, you can have the block ignore that change. The block instead continues to pass the previous value. Ignoring small changes like these is called **hysteresis**.

For example, suppose you are using hysteresis with a block that has an Output Uncertainty of 0.5. If the block's belief value changes from 0.8 (true) to 0.7 (unknown), it continues to pass .true. If the block's belief value changes from 0.1 (false) to 0.6 (unknown), it will continue to pass .false.

**To specify when the block performs hysteresis:**

➔ Set the attribute Hysteresis When to one of the values in the following table:

If Hysteresis When is...	The block performs hysteresis when its value changes to unknown from...
.true	.true
.false	.false
always	.true or .false
none	never

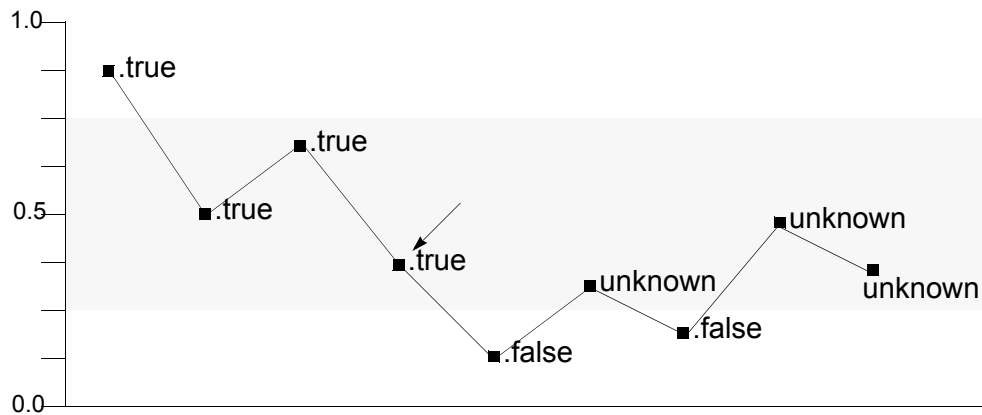


---

**Note** Only Condition blocks can perform hysteresis. The Condition blocks are on the Observations palette and the Conditions palette.

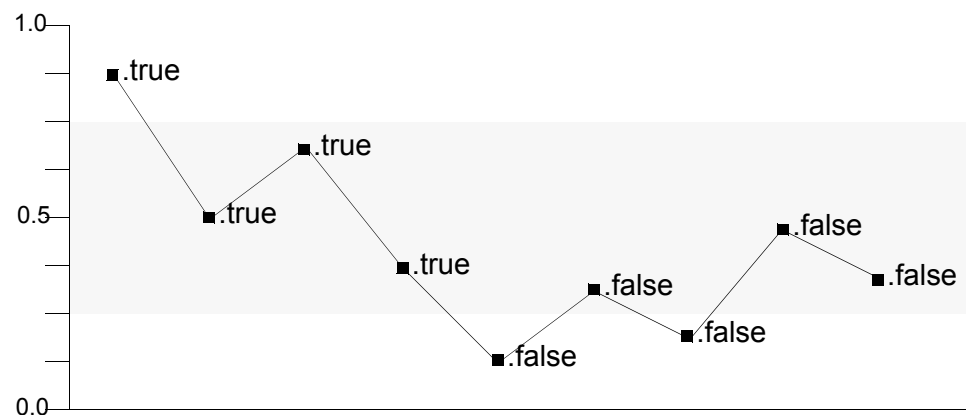
---

The following figure graphs the belief values of a block that has Hysteresis When set to `.true`. The corresponding status values are near each belief value. When the belief value goes from the true range to the unknown range, the block continues to pass `.true`. When the belief value goes from the false range to the unknown range, the block passes `unknown`.



Note that when the block has the belief value near the arrow, the block passes `true` although the value is under 0.5. Even if the belief value is closer to the `.false` range than the `.true` range, the block passes `.true` whenever the block's belief value goes from `.true` to `unknown`.

The next figure graphs the belief values of a block that has Hysteresis When set to `always`. The corresponding status values are near each belief value. When the belief value goes from the true range to the unknown range, the block continues to pass `.true`. When the belief value goes from the false range to the unknown range, the block continues to pass `.false`.



# Using Variables and Parameters

You use variables in a diagram to:

- Receive data from or send data to external data sources using G2 Gateway (GSI) or other bridges.
- Specify the validity interval for a value on a path, which determines when the value expires.

In addition, you can use variables or parameters to initiate forward chaining. Thus, use a **variable** when you need to connect to external data or use expiration times; otherwise, use a **parameter**.

## Choosing the Type of Variable or Parameter

There are a number of variable and parameter types available. Choose the type that is most specific for the type of data being passed.

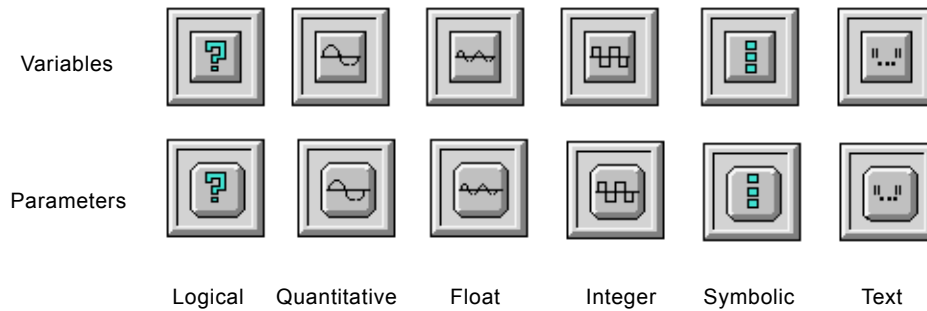
The following table describes the various types of variables and parameters:

<b>A variable or parameter of type...</b>	<b>Passes data in the form of...</b>
Logical	true or false
Quantitative	Floating point number, e.g., 1.2, or integer, e.g., 1
Float	Floating point number, e.g., 1.2
Integer	Integer, e.g., 1
Symbolic	Symbols, e.g., .true, .false, or unknown, high or low, or any class name when connected to an item path
Text	Text strings, e.g., "warning"
Sensor (variables only)	GSI sensor data

## Creating a Variable or Parameter

To create a variable or parameter:

→ Select KB Workspace > New Object > g2-variable or parameter, then choose the desired type:



## Using Variables to Connect to External Data

You can use variables to connect to external data when you use G2 Gateway (GSI), either to obtain data from an external source or send data to an external process. See the *G2 Gateway Bridge Developer? Guide* for information about G2 Gateway.

To identify the external data source or target:

- Specify one of the following values for Data-server in the variable's table:
- Inference engine
  - G2
  - GSI data server
  - Any data server alias

See the *G2 Reference Manual* for a discussion of the various types of data servers.

The following figure illustrates a logical variable and its table:



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

Specify the data server.

---

**Note** When you set the Data-server attribute to inference engine for a variable, GDA uses the G2 conclude action to change values locally within GDA. When you set the Data-server attribute to anything other than inference engine, however, GDA uses the G2 set action to set the value in the external system.

---

## Creating a Sensor

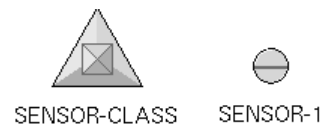
Sensors are a particular kind of variable that obtains or passes quantitative values from or to external data sources; sensors do not allow the specification of a formula.

**To create a sensor:**

- 1 Select KB Workspace > New Definition > class-definition > object-definition to create a new class of object.
- 2 In the new class's table, specify the Direct-superior-classes as sensor.

- 3 Specify the **Name** of the new class. Close the table.
- 4 Select **create instance** from the new class's menu to create an instance of the class. Specify the **Name** of the new instance.
- 5 Specify the **Data-server** for the sensor in the new object's table.

This figure illustrates the sensor class, a sensor instance, and the default table:



SENSOR-1, a sensor-class	
Options	do not forward chain, breadth first backward chain
Notes	OK
Item configuration	none
Names	SENSOR-1
Tracing and breakpoints	default
Data type	quantity
Initial value	none
Last recorded value	no value
History keeping spec	do not keep history
Validity interval	indefinite
Simulation details	no simulation formula yet
Initial value for simulation	default
Data server	G2 simulator
Default update interval	none

## Connecting a Variable or Parameter to a Block

**To connect a variable or parameter to a block:**

- ➔ Drag a connection stub from the block into the variable or parameter, and click inside the icon to make the connection.

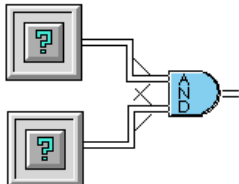
This connection establishes the link between the variable or parameter and the diagram.

---

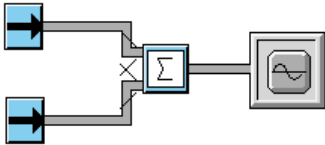
**Note** Be sure to select Controls > Enable Data Input to enable data to flow into the diagram.

---

You can connect variables and parameters to the input path of any block, for example, to feed external data into a diagram:



You can also connect variables and parameters to the output path of any block, for example, to send data to an external process or to initiate forward chaining:



---

**Caution** In general, do not create a variable or parameter that both receives data from an input path and sets data onto an output path, because GDA cannot guarantee the order of execution of the variable or parameter relative to other blocks.

---

## Overriding Values of Variables and Parameters

Any variable or parameter connected at the input end of a data, inference, or control path contains two additional menu choices:

- override
- lock or unlock

**To manually override a variable or parameter:**

➔ Use the override menu choice.

You use this feature to manually feed values into the diagram for testing purposes. For information on the various types of override dialogs that can appear, see [Coercing Data Using Variables and Parameters as Input](#).

Manually overriding the value of a variable or parameter automatically locks the object, which displays a lock icon associated with the object. A locked object prevents the propagation of values obtained from any data source other than manual. For example, if a rule or procedure concludes a value into a locked

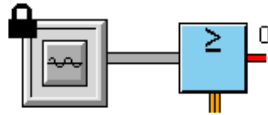
variable, or if an external data source passes a new value into the variable, the variable receives the value but does not propagate it onto the path. Manually overriding the value of a locked variable or parameter does propagate the value.

A locked object contains **unlock** in its menu; an unlocked object contains **lock** in its menu.

#### To unlock a locked object:

- ➔ Select **unlock** from the object's menu, or delete the lock icon by selecting **delete** from the icon's menu.

The following figure illustrates a simple diagram containing a float variable connected to a High Value observation block via a data path. The variable is locked because the user selected **override** from the menu.



## Coercing Data Using Variables and Parameters as Input

When using variables and parameters to set values onto a path, the override dialog and the resulting value that appears on the output path depend on two things:

- The type of variable or parameter.
- The type of path to which the variable or parameter is connected.

In the simplest case, the value on the path corresponds to the value stored by the variable or parameter to which it is connected. In the example above, the float variable contains a floating point number, which corresponds to the data type passed by the path to which it is connected. Thus, GDA presents the user with a Type-in override dialog for entering a new floating point value.

You can also connect variables and parameters to connection paths where the data types do not correspond. For example, you can connect a float variable to an inference path, which converts the float value to a status and belief value.

---

**Caution** Do not connect more than one output path to a variable or parameter, otherwise the results will be unpredictable.

---

The following chart summarizes the data coercion that takes place when you use the various types of variables and parameters as input to the four output path types. The cells indicate the type of override dialog that appears and the value on the output path. The shaded cells indicate combinations that do not have

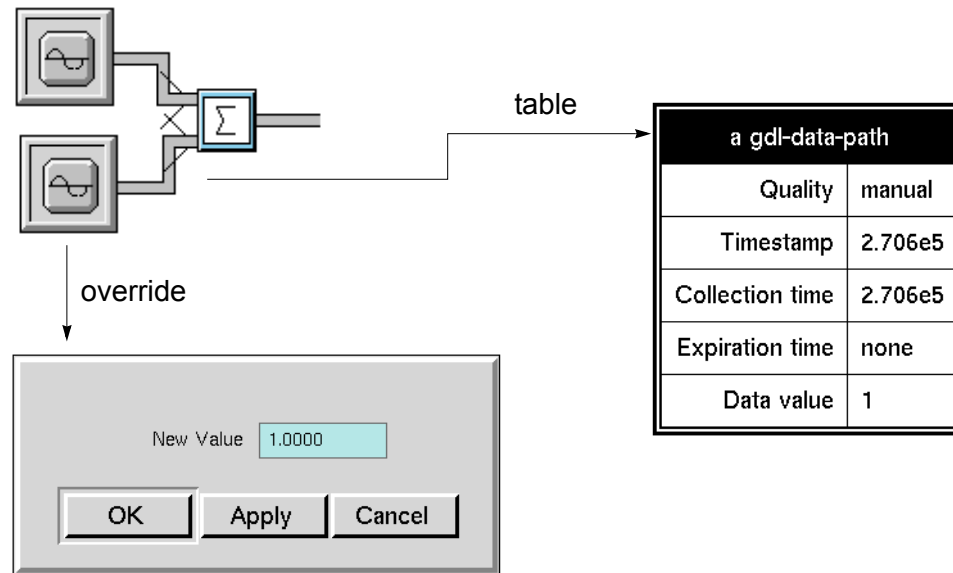
meaning. See [Examples of Data Coercion Using Variables and Parameters as Input](#) for examples of the various types of override dialogs.

Type	Output			
	Data Path	Inference Path	Control Path	Item Path
<b>Logical</b>		T/ F Radio Button  Discrete belief value (0.0 and 1.0 only)	Output control signal  Control signal	
<b>Float or Sensor</b>	Type-in Floating point data value	Belief Slider  Fuzzy belief value	Output control signal  Control signal	
<b>Integer</b>	Type-in Integer data value	Belief Slider (0 or 1 only)  Discrete belief value (0.0 and 1.0 only)	Output control signal  Control signal	
<b>Symbolic</b>	Type-in Symbolic data value	T/ F/U Radio Button  Discrete belief value (0.0, 0.5, or 1.0)	Output control signal  Control signal	Type-in Class or item name
<b>Text</b>	Type-in Textual data value		Output control signal  Control signal	

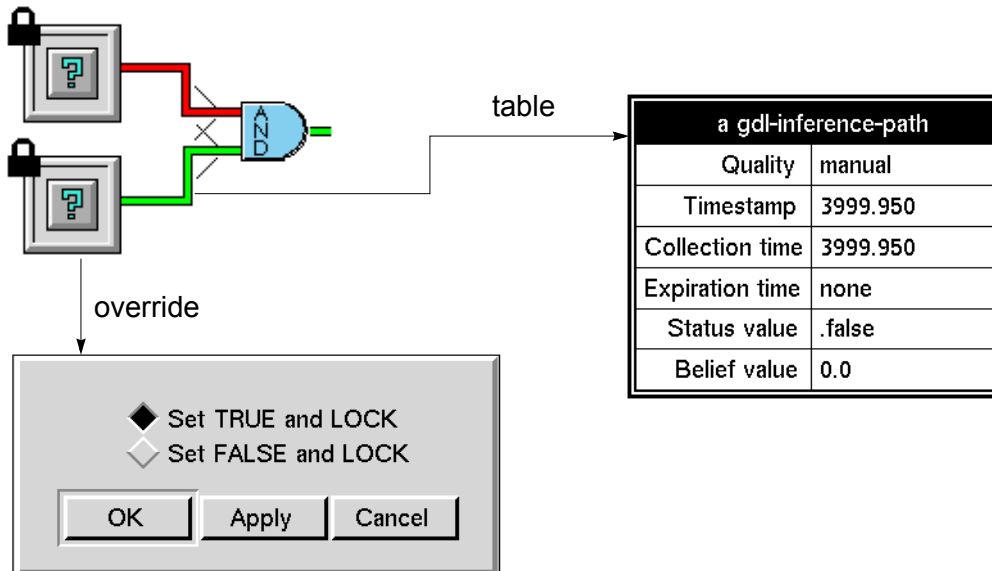


### Examples of Data Coercion Using Variables and Parameters as Input

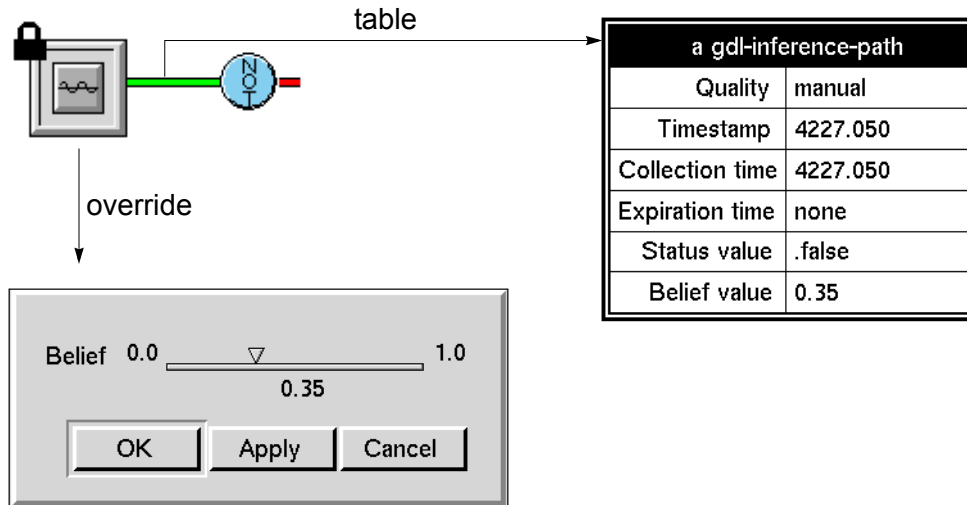
The following figure illustrates two quantitative parameters connected to a data path using a Summation block. The figure shows the Type-in override dialog for one of the parameters and the table for one of the data paths.



This figure illustrates two logical variables connected to inference paths using an And gate. The figure shows the T/F Radio Button override dialog for one of the variables and the table for one of the inference paths.

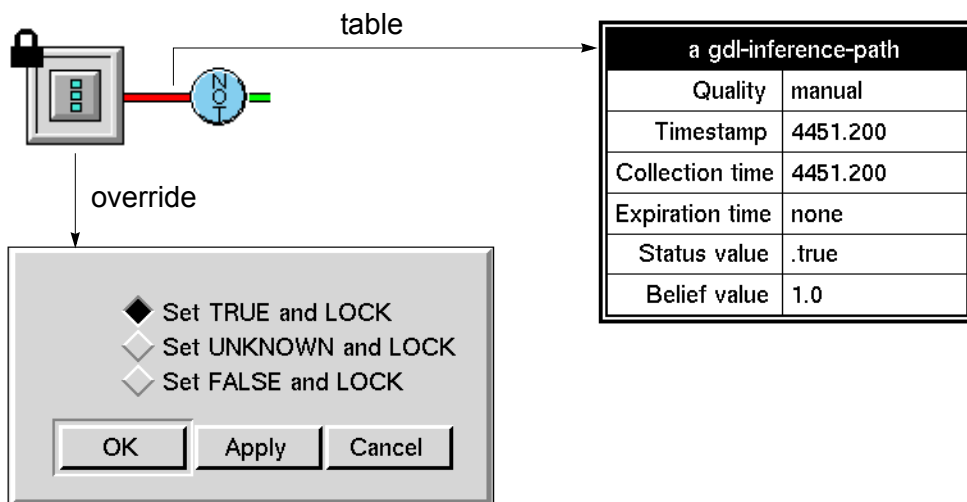


The next figure illustrates a float variable connected to an inference path using a Not gate. The figure shows the Belief Slider override dialog for the variable and the table for the inference path.

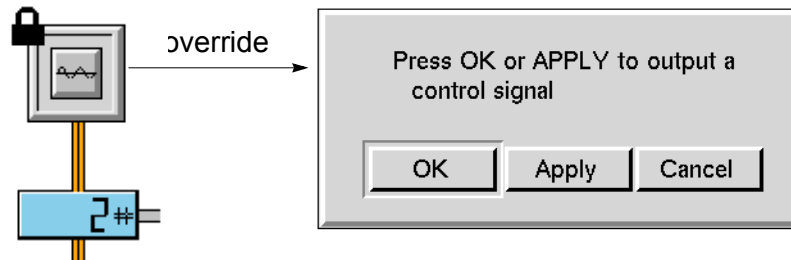


**Note** For an integer variable or parameter connected to an inference path, the Belief Slider only accepts values of 0 or 1.

This figure illustrates a symbolic variable connected to an inference path using a Not gate. The figure shows the T/F/U Radio Button override dialog for the variable and the table for the inference path.



The next figure illustrates a float variable connected to a control path using a Control Counter block. The figure shows the output control signal dialog for the variable.



For an example of connecting a symbolic variable to an item path, see [Placing an Item Onto an Item Path Interactively](#).

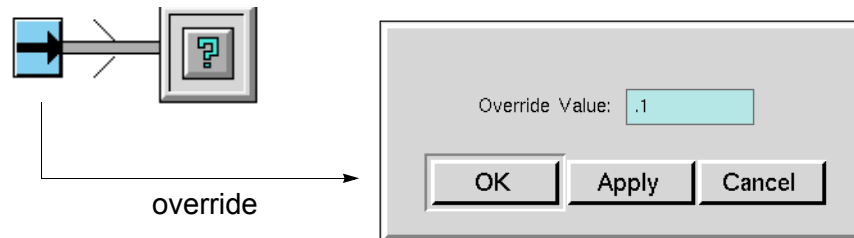
## Coercing Data Using Variables and Parameters as Output

GDA also coerces data passed into a variable or parameter from a path. The following chart summarizes the data coercion that takes place when you use the various types of variables and parameters as output from the four input path types. The cells indicate the value stored in the variable or parameter based on the input value. The shaded cells indicate combinations that do not have meaning. See [Examples of Data Coercion Using Variables and Parameters as Output](#) for examples.

Input	Type				
	Logical	Float	Integer	Symbolic	Text
<b>Data Path</b>	false when Data-value < 0.5; true when Data-value ≥ 0.5	Floating point Data-value	Data-value rounded to the nearest integer	Symbolic Data-value (a quantity gives an error)	Textual Data-value
<b>Inference Path</b>	true or false (unknown is ignored)	Belief-value	Discrete integer Belief-value (0 and 1 only, rounded to nearest)	Symbolic Status-value (.true, .false, or unknown)	Textual symbolic Status-value (".TRUE," ".FALSE," or "UNKNOWN")
<b>Control Path</b>		Adds 1.0 to Last-recorded-value	Adds 1 to Last-recorded-value		
<b>Item Path</b>				Class or item name of path item	

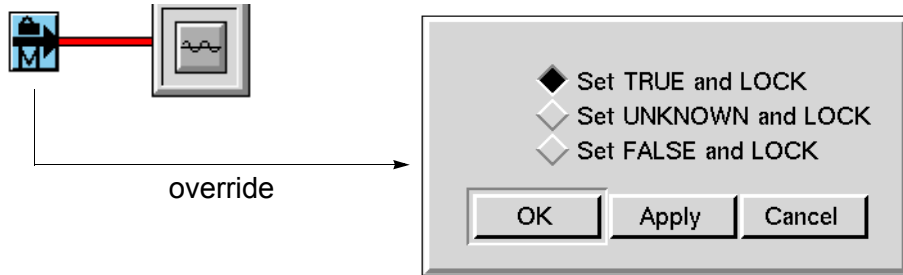
## Examples of Data Coercion Using Variables and Parameters as Output

The following figure shows a Numeric Entry Point connected to a logical variable. The figure shows the Type-in override dialog for the Numeric Entry Point and the table for the logical variable.



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

The next figure shows a Belief Entry Point connected to a float variable. The figure shows the T/F/U override dialog for the Belief Entry Point and the table for the float variable.



a float-variable	
Options	do not forward chain, breadth first backward chain
Notes	OK
Item configuration	none
Names	none
Tracing and breakpoints	default
Data type	float
Initial value	none
Last recorded value	1.0, valid indefinitely
History keeping spec	do not keep history
Validity interval	supplied
Formula	none
Simulation details	no simulation formula yet
Initial value for simulation	default
Data server	inference engine
Default update interval	none



## Evaluating Expressions in Attributes

In several blocks, you can set an attribute to an expression that the block will evaluate later. You must enclose the expression to evaluate in brackets and quotes ("*expression*"). For example, you could set the Name of Sensor attribute of an Entry Point to "[the temp of tank1]", where tank1 is an object with the variable or parameter attribute temp. Or you could set the When True attribute of a High Value block to "The temperature is over [threshold]" which the block may evaluate to "The temperature is over 100".

This section lists the attributes that contain expressions, describes the type of expressions they may contain, and shows examples of some diagrams that use this feature.

---

**Note** If you use expressions in attributes, GDA evaluates the expression at run-time each time the block evaluates, which degrades performance.

---

These attributes can contain an expression that evaluates to a string:

- When True, When False, When Unknown in the Observation and Conclusion blocks.
- Option 1 Description, Option 2 Description, and Option 3 Description in the User Query Control Switches on the Control Action blocks.
- Advice in the Alarm and Recurring Alarm Capabilities.
- Override Text and New Value Prompt in the Dialog Restriction and Manual Override Restriction.

These attributes can contain an expression that evaluates to a G2 variable or parameter, such as "[the volume of mixing-tank-3]":

- Name of Sensor attribute for a Numeric Entry Point, Text Entry Point, Symbolic Entry Point, Belief Entry Point, or Control Entry Point.
- Target Variable in the Data Output block.
- External Datasource in Residual.
- Expected Value in Standard CUSUM, Two-Sided CUSUM, EWMA, and SPC Run.

This attribute can contain an expression that evaluates to a symbol that names a graph or chart:

Chart Name in the Chart capability.

This attribute can contain an expression that returns a truth value:

Exit If in the Control Path Loop.

This attribute can contain an expression that evaluates to a GDA object:

Target in the Highlight block.

---

**Note** In the values of these attributes, include brackets only if the brackets are enclosing an expression. Any other use of brackets could cause an error.

---

You cannot include any arbitrary G2 expression in these G2 attributes. The table below lists the bracketed expressions that attributes can contain:

<b>This expression...</b>	<b>Is...</b>	<b>And is replaced by...</b>
[ <i>object-name</i> ]	The name of a G2 variable or parameter.	The value of the G2 variable or parameter.
[ <i>attr-name</i> ]	An attribute in this block.	The value of the attribute.
[the <i>attr-name</i> of <i>object-name</i> ]	An attribute in the named object.	The value of the attribute.
[the <i>attr-name</i> of the superior object]	An attribute in the object that owns the subworkspace that contains this block.	The value of the attribute.
[the superior object]	The object that owns the subworkspace that contains this block.	The object.
[( <i>g2-expression</i> )]	Any G2 expression you can use in a G2 formula.	The value of the expression.
[(call function <i>g2-function</i> )]	A call of a G2 function.	The value of the function.
[(call procedure <i>g2-procedure</i> )]	A call of a G2 procedure.	The return value of the procedure.

Note that *object-name* can be the name of any named G2 object, and *attr-name* can be any user-defined attribute or the system attribute **name**. The syntax for *g2-expression*, *g2-function*, and *g2-procedure* is discussed in the sections below.



## Using a G2 Expression

When you use a string of the form "[*(g2-expression)*]", you can use many G2 expressions in the attribute. GDA evaluates the expression and replaces it with the value returned.

To refer to the block that contains the attribute, your expression can use the term *the item*. For example, if you are setting the Description When True attribute of a High Value block, the string can contain "The value is over [(the threshold of the item \* 2)]".

---

**Note** You must enter the expression *the item* with a single space between the two words. If it contains a tab, line feed, or more than one space, GDA will not recognize it.

---

When you run your application in a run-time or restricted use G2, you cannot use a string of the form "[*(g2-expression)*]". Instead, create a new function or procedure that contains that expression and use a string of the form "[*(call function g2-function)*]" or "[*(call procedure g2-procedure)*]".

## Using a G2 Function or Procedure

With a string of the form "[*(call function g2-function)*]" or "[*(call procedure g2-procedure)*]", you can call any function or procedure from the attribute. Unlike using a G2 expression, you can call a function or procedure when your application is running in a run-time or restricted use G2.

The function or procedure must take one argument which will be the block that contains the attribute, and must return either a string or a variable or parameter, depending on what attribute it is used with.

This is the format for a function:

```
function-name (var-name) = ( . . . . )
```

This is the format for a procedure:

```
procedure-name (blk: class gdl-block) = (item-or-value)
begin
  . . .
  return expression;
end
```

## Examples

The picture below shows an Entry Point that uses an expression for the Name of Sensor. The expression refers to the volume of Tank-1, which is an attribute that is given by a variable class. The Data Source is **external**. The output data value for the entry point corresponds to the current value of the volume of the tank.



Volume 9.0

TANK-1

**Numeric Entry Point**

Name of Sensor: "[the volume of tank-1]"

Data Source:  embedded  external

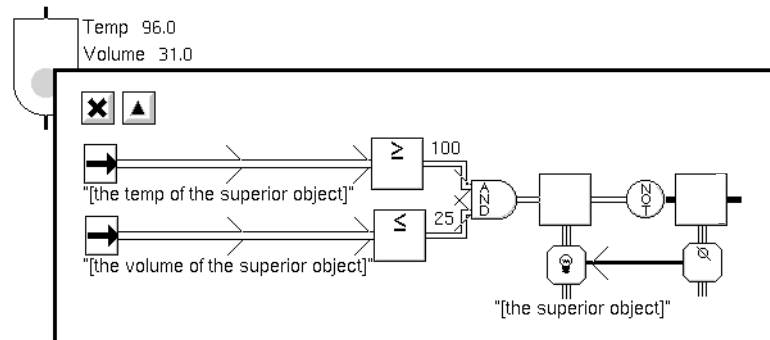
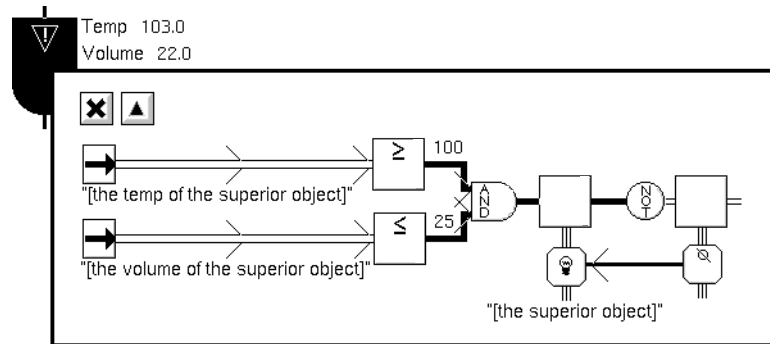
Validity Interval: supplied

Value on Initialization: NONE

OK Apply Cancel

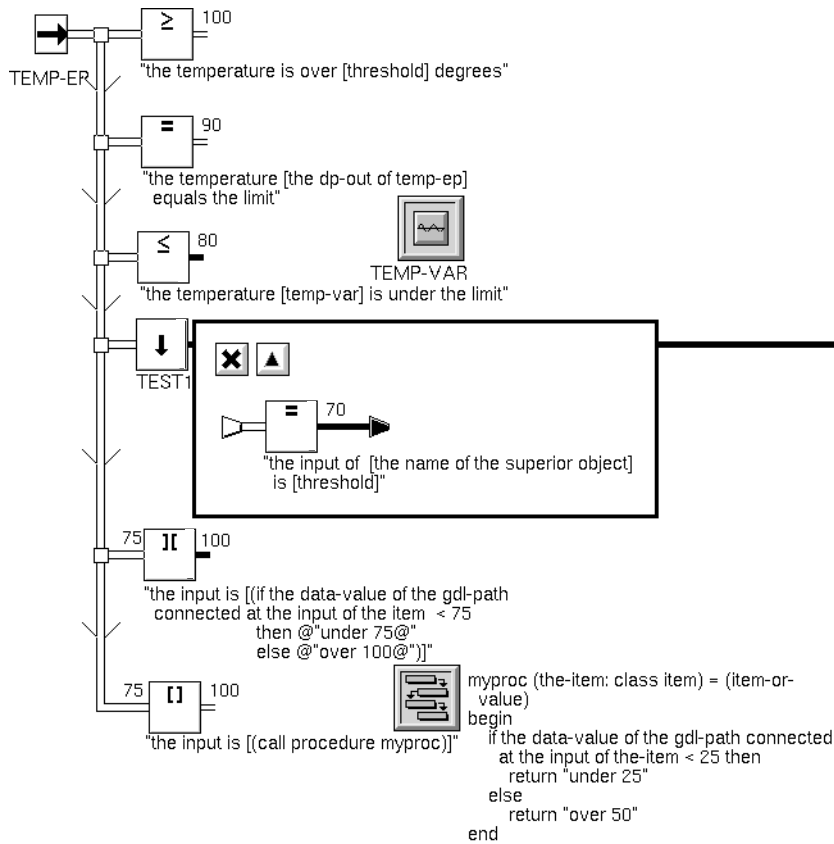
The following picture shows two tank objects with subworkspaces. Although the subworkspaces are identical, you can use indirect references to make sure the blocks on the subworkspaces refer to the appropriate tank. On each subworkspace is a GDA diagram that reads the temperature and volume from the tank and tests whether those numbers fall within certain limits. If they are outside

the limits, the diagram highlights the tank object. Otherwise, the diagram resets the object.



The picture below shows a diagrams with several blocks that use text concatenation. The Description When True attribute appears under each Observation block. The Entry Point TEMP-EP gets its value from the G2 variable TEMP-VAR. TEMP-EP passes that value to several Observation blocks and one

Encapsulation block named TEST1. The Encapsulation block has one Observation block on its subworkspace.



This table shows how the blocks could expand their descriptions:

This description...	Could expand into this...
"the temperature is over [threshold] degrees"	"the temperature is over 100 degrees"
"the temperature [the dp-out of temp-ep] equals the limit."	"the temperature 90 equals the limit"
"the temperature [temp-var] is under the limit"	"the temperature 80 is under the limit"
"the input of [the name of the superior object] is [threshold]"	"the input of test1 is 70"

**This description...**

"the input is [ (if the data-value of the  
gdl-path  
connected at the input of the item < 75  
then @"under 75@"  
else @"over 100@") ]"

"the input is [ (call procedure myproc) ]"

**Could expand into this...**

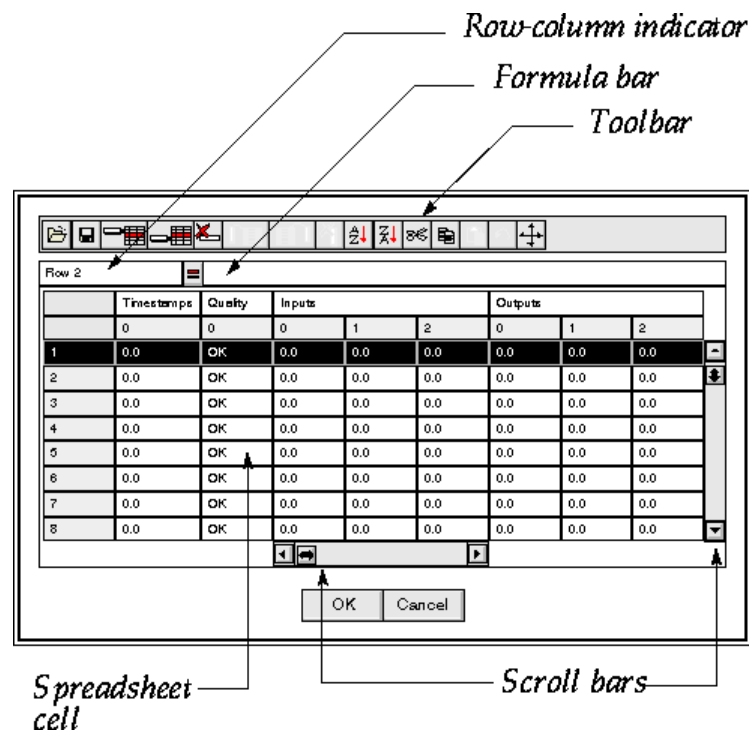
"the input is over 100"

"the input is under 25"

## Using the GXL Spreadsheet to Edit Data

Many blocks that operate on vectors, Data Pairs, and Data Sets let you edit data by using a GXL spreadsheet. GXL (Gensym Spreadsheet System) is a G2 module that provides spreadsheet editing capabilities for editing vector and data set blocks.

This spreadsheet is for editing a Data Set block with ten samples, five inputs, and three outputs:



The spreadsheet displays eight rows and three columns of data. If there is more data than will fit, vertical and horizontal scroll bars appear in the spreadsheet.

**To enter a value in the spreadsheet:**

➔ Click on the cell you want to edit, enter a number or click in the formula bar to enter a formula, and press Return to move to the next cell in the spreadsheet.

You can read and write data from and to a file, for individual cells, rows, or the entire spreadsheet. You can add and delete rows and columns, cut and paste cell values, and sort rows.

The toolbar buttons are, from left to right:

<b>Toolbar Button</b>	<b>Function</b>
Save	Saves the current selection to a file.
Load	Loads data from a file into the selected area.
Add row before	Inserts an empty row before the currently selected row.
Add row after	Inserts an empty row after the currently selected row.
Delete row	Deletes the selected rows.
Add column before	Inserts an empty column before the currently selected column.
Add column after	Inserts an empty column after the currently selected column.
Delete column	Deletes the selected columns.
Change color	Sets the text, background and border color of selected cells.
Cut	Transfers the current selection to the clipboard.
Copy	Copies the contents of the selected cells to the clipboard.
Paste	Copies the contents of the clipboard into the selected cells.
Undo	Reverses the last operation.
Sort ascending	Sorts the rows of the selection in ascending order according to the contents of a key column.
Sort descending	Sorts the rows of the selection in descending order according to the contents of a key column.

Once you have edited the spreadsheet, select OK to save the changes and close the spreadsheet. To close the spreadsheet and discard the changes you made, select Cancel.

For more information on using GXL spreadsheets, see the *G2 XL Spreadsheet User? Guide*.

## Understanding the GDA Block Evaluation Engine

GDA operates using an efficient block evaluation engine. Although distinct from the G2 scheduler, it operates in a similar way in that blocks are scheduled for evaluation, then executed within the GDA engine.

When a block requires evaluation, GDA “invokes” the block, flagging the block for later execution, accomplished during the next GDA cycle through your application.

To understand how blocks in a diagram evaluate, you need to understand:

- How GDA invokes an individual block.
- The cycling of the GDA Engine, by which GDA evaluates invoked blocks.
- The evaluation of blocks on an individual workspace.

### Invoking an Individual Block

GDA invokes a block when:

- An entry point receives a value from its data source.
- A value is propagated onto the input path of the block due to the behavior of an upstream block.
- You evaluate a block manually.
- A variable or parameter connected to a diagram receives a value.
- A programmatic call to `gdl-evaluate-block` occurs.

If a block that is located on the subworkspace of an encapsulation is invoked, GDA invokes that encapsulation block, and any superior encapsulations. The invocations propagate upwards through the workspace hierarchy to the top-most diagram that contains the encapsulation blocks.

When an upstream block propagates an inference value, GDA performs what is referred to as **inference path filtering**. This means a new input inference path does not invoke a downstream block if the belief value has not changed.

Normally when a block is invoked, the block is evaluated as part of the evaluation of the workspace. If a block is invoked outside the normal evaluation of workspaces, then that block has to be picked up by a sweep, and scheduled for evaluation. The evaluation of the blocks occurs asynchronously with the invocation of the blocks.

## Executing Diagrams

GDA executes an application by searching, workspace by workspace, for invoked blocks. It does this in a periodic fashion, executing all the blocks on each workspace and continuing until there are no more blocks left to evaluate.

While iterating through each workspace, GDA performs three distinct types of searches:

- Manual invocations - invocations in response to a manual evaluation or an override.
- Entry point invocations - sweeps that propagate new data obtained by entry points.
- System invocations - all other block invocations.

For each type of sweep, GDA follows certain rules to check blocks for invocations and evaluate these blocks.

### Executing Blocks that Have Been Manually Evaluated

When you evaluate a block manually by using either the `evaluate` or `override` menu choice, the block is executed separately from system-evaluated blocks. Manual invocations are never mixed with other types of invocations, because each manual invocation is associated with a specific client (for example a G2 window or Telewindows2 window).

Invocations resulting from manual overrides are grouped according to their client, and all evaluations associated with that client are resolved before moving on to the next client (if any).

### Executing Blocks that Have Been Launched by the System

After all manual evaluations have occurred, GDA finds all blocks that have been evaluated programmatically. GDA obtains all blocks that have been invoked since the last cycle, and then evaluates them on a workspace-by-workspace basis. Note that individual workspaces are evaluated all at once (as described in the Workspace Evaluation section). GDA continues processing all blocks, including blocks that are invoked by other workspace evaluation within the same cycle, until no more invocations remain. Thus, if a block is invoked during an evaluation cycle, GDA evaluates the block within the current cycle. If a block is invoked outside of a cycle, GDA evaluates the block in the next cycle.



Once all blocks have been evaluated, GDA determines whether it is time to evaluate entry points. GDA determines whether the appropriate Sweep Interval has passed since the last sweeping of entry points. If a sweep interval is warranted, GDA proceeds on to Sweeping Diagrams with Entry Points (see the next section). If it is not time for an entry point sweep and there are no remaining blocks to evaluate, GDA allows time for other tasks.

GDA performs other tasks in this order. First, GDA enters a wait state to allow G2 to process any non-GDA tasks that may be waiting. The GDL Engine determines the length of this wait. The GDL Engine **scheduling-granularity** attribute is the length of this wait in seconds. The default scheduling granularity of GDA is 0.1 seconds. The user can increase this value in an attempt to shift G2 resources away to non-GDA tasks. Because the G2 scheduler operates in .05 second intervals, making this value shorter has little meaning.

GDA also takes the time between evaluation cycles to perform other tasks, such as turning animation on and off, updating chart displays, and recompiling workspaces. This third task is explained in greater detail in [Evaluating Blocks on Individual Workspaces](#).

At this point, GDA starts again with another manual evaluation cycle.

## **Sweeping Diagrams with Entry Points**

An entry point sweep occurs when GDA evaluates all invoked entry points and propagates the entry point's current value. If the entry point receives several values between sweeps, only the last value that the entry point receives is passed downstream.

An entry point sweep only occurs if the appropriate Sweep Interval has passed. You can control the frequency of entry point sweeps in the Environment Settings under the Preferences menu. By default, entry point sweeps happen once every second. The next section explains how to change the interval.

## **Setting the Sweep Interval**

You can set the interval at which GDA sweeps the entry points in your diagram. By default, the interval is 1 second, which is the lowest recommended value.

### **To process entry point data in a diagram using a different sweep interval:**

- 1** Select the Preferences > Settings > Environment menu choice to display the Environment Settings dialog.
- 2** Set Sweep Interval, then press OK.

## Evaluating Blocks on Individual Workspaces

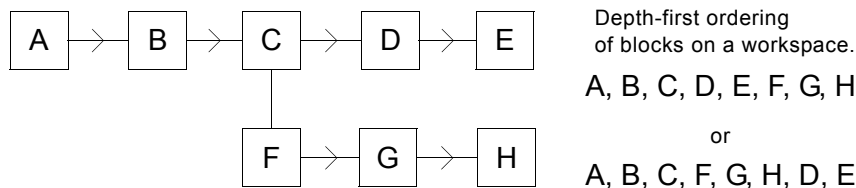
To understand the order of evaluation of blocks on an individual workspace, you can think of each block on a workspace as having a number, which is the order in which GDA checks for invocations on the workspace.

- If the blocks on a workspace contain no feedback loops, GDA searches for blocks in a depth-first order, ignoring all connection posts, except those that connect to connection posts on the same workspace. For a detailed description of depth-first ordering, see *Introduction to Algorithms*, T.H. Cormen, C. E. Leiserson, R.L. Rivest, McGraw-Hill, 1990.
- If a block branches to one of multiple output paths along multiple parallel lines, GDA arbitrarily chooses one path and continues numbering sequentially in a depth-first manner until it has ordered all blocks along the path. GDA then backs up to the block with a decision point and picks up the sequential numbering of blocks along the other path(s).
- If a diagram contains blocks with multiple input paths, in general, the block with multiple inputs is always evaluated after all of its input blocks evaluate, and any of its inputs can cause the block to evaluate. This means that, in general, blocks with multiple inputs only evaluate once per evaluation of a given workspace. The exception is blocks that have their **multiple-investigations** attribute set to **Queue** or **OK**, in which case the block can evaluate more than once per cycle.

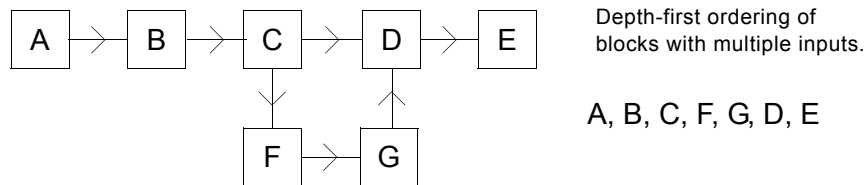
The result is a list in which all upstream blocks have a lower number than all downstream blocks, including blocks with multiple input paths. Numbering starts at zero at a block that has no upstream blocks and continues sequentially until every block on the workspace has been numbered. If a diagram contains multiple blocks not connected to upstream blocks, GDA arbitrarily chooses one of these blocks as a starting point.

Capabilities and Restrictions are always numbered sequentially directly after the blocks to which they are attached, so they always evaluate immediately after the block to which they are attached.

This diagram shows how GDA numbers blocks on a simple diagram with two parallel paths in a depth-first order:



This diagram shows how GDA numbers blocks on a diagram that contains a block with multiple inputs, in this example, block D:

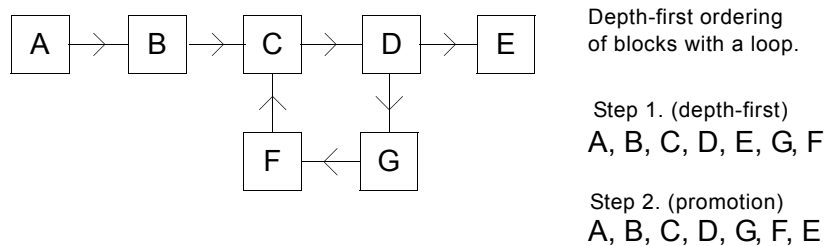


Inside of feedback loops, no upstream/downstream relationships exist; thus feedback loops are numbered in two steps.

- 1 First, the blocks in the loop are numbered via a depth-first search, according to the explanation above.
- 2 Next, GDA backs up to the block that begins the loop and "promotes" the blocks in the loop so that they appear consecutively in the list, without changing their relative order.

In this way, the loop processes until it is finished before the downstream blocks evaluate. When a diagram contains nested loops, the same "promotion" process assures that the inner loop finishes evaluating before each evaluation of the outer loop.

This diagram shows how GDA numbers blocks on a diagram with a loop:



The process of ordering the blocks in the diagram is referred to as a **compilation** of a workspace. As described previously, the recompilation of workspaces occurs in between evaluation cycles. Further, a workspace is only recompiled if it has been tagged as "invalid." Workspace invalidation occurs whenever the user adds a block to a workspace, removes a block from a workspace, clones an existing block, or makes or breaks a connection between blocks. A workspace can also be invalidated programmatically, which is necessary if the diagram has been programmatically modified.

Once the order of blocks has been determined, an individual workspace always executes the blocks in that order. When blocks evaluate, an "evaluation pointer" advances from block to block, following the serial numbering of blocks in ascending order. If the pointer arrives at an invoked block, the block evaluates. If the pointer arrives at a block that has not been invoked, the pointer moves to the

next block. The only time the pointer moves back in the series is when there is a path from a higher numbered block to a lower numbered block, at which point evaluation returns to the lower numbered block only if it has a pending invocation. Evaluation then continues from the lower numbered block. Evaluation on the workspace ends when the evaluation pointer reaches the last block in the list.

If a workspace contains an encapsulation block, GDA evaluates all the blocks on its subworkspace. When GDA has completely evaluated the blocks on the subworkspace, evaluation returns to the superior workspace.

Asynchronous blocks, such as the Delay block, do not hold up evaluation. When these types of blocks begin evaluation, the evaluation pointer moves directly to the next block. When the delay is finished, a new evaluation sweep begins.

For more information on asynchronous block evaluation, see [Multiple Invocations](#)

# Using GDA Queues

---

*Describes how to use the queues feature.*

Introduction	131
Summary of the Queue Features	132
Using the Alarm Queue	134
Using the Error Queue	162
Using the Explanation Queue	163
Using the Message Queue	164
Sending Messages to Queues Using the Queue Message Block	165



## Introduction

The GDA Queue system allows users to display important information about a GDA application and to communicate with each other within a G2 process or on remote G2 processes. A **queue** is a temporary, dynamic repository for messages. The messages posted to queues are called **entries**. Entries contain the text of the message and other relevant data, such as the creation time, and for alarm entries, severity and acknowledgement status.

GDA provides four basic queues that support these types of activities:

- The **Alarm Queue** holds entries generated by alarms.
- The **Error Queue** receives error messages that result while running GDA applications.

- The **Explanation Queue** holds explanations generated by inference blocks or Explanation Memory blocks.
- The **Message Queue** enables the sending and receiving of messages between users.

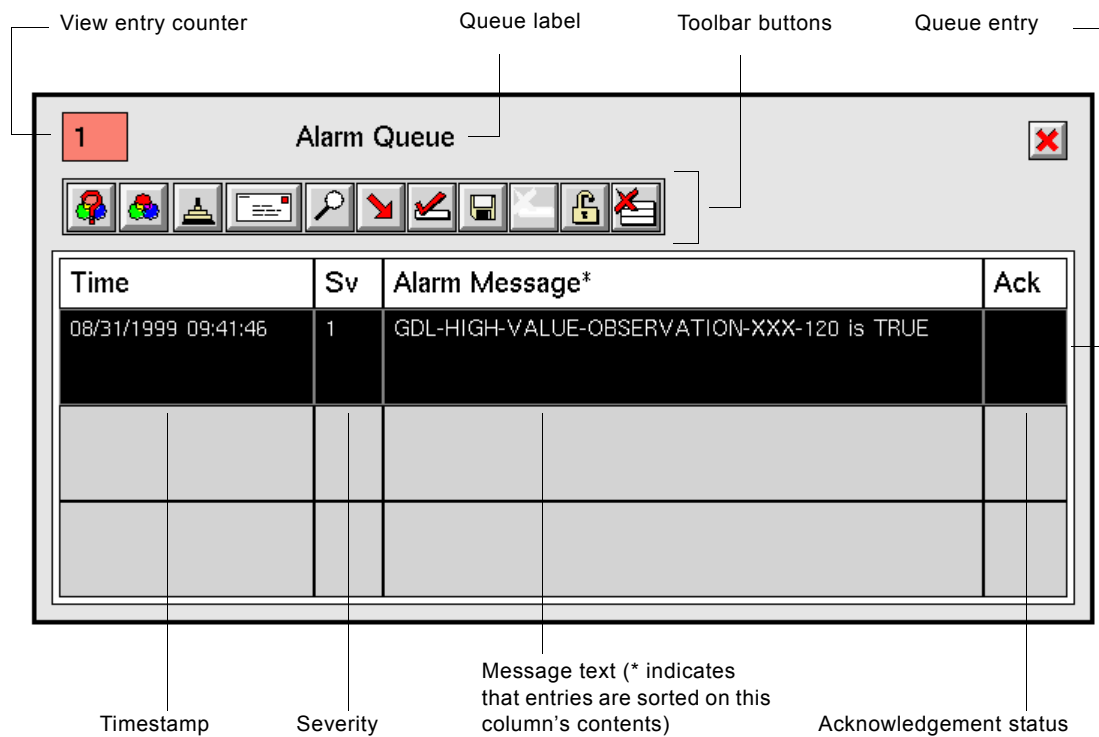
This chapter describes how the end user interacts with the queues. The following chapters describe configuring the queues:

- [Creating and Configuring Queues](#)
- [Creating Queue Views](#)

For information about programming queues, see the *GDA API Reference*.

## Summary of the Queue Features








This figure shows a view of an Alarm Queue with a selected entry. The figure contains labels for useful parts of the queue view:







The features of the Error Queue, Explanation Queue, and Message Queue are similar to the Alarm Queue except:

- The queue label indicates the type of queue.
- These queues do not include the Severity (Sv) and Acknowledged (Ack) columns.
- These queues do not include the buttons associated with acknowledging alarms and locking the view.
- The column header for the message text (“Alarm Message” for the Alarm Queue) varies according to the type of queue.

This table provides a summary of each toolbar button available on the Alarm Queue. You can view a tool tip for a button by holding the mouse button down over a button.

Queue Button	Button Name	Description
	Select Filters	Enables users to define filters and select filters to activate. Filtering displays queue entries based on specified criteria. See <a href="#">Filtering Queue Entries</a> .
	Apply Filters	Applies the currently selected filter or filters to the queue entries. See <a href="#">Filtering Queue Entries</a> .
	Sort Entries	Reverses the order in which the entries are sorted. See <a href="#">Sorting Entries</a> .
	Send Entries	Sends the selected entries to another queue. See <a href="#">Sending Entries to Another Queue</a> .
	View Details	Displays details about the selected entry. See <a href="#">Viewing the Details of an Entry</a> .
	Go to Source	Displays the workspace containing the source of the entry and places an arrow next to the source. See <a href="#">Showing the Source of an Entry</a> .
	Acknowledge	Acknowledges the selected entry. This button is only available on the Alarm Queue. See <a href="#">Acknowledging Alarms</a> .

Queue Button	Button Name	Description
	Save Entry	Writes the message text associated with the selected entry to a file. See <a href="#">Saving a Queue Entry</a> .
	Remove Entries	Removes the selected entries from the queue. See <a href="#">Removing Entries from the Queue View</a> .
	Lock View	Prevents the queue from displaying new entries. This button is only available on the Alarm Queue. See <a href="#">Locking the Alarm Queue View</a> .
	Clear View	Removes all the visible entries from the queue. On the Alarm Queue, also acknowledges the entries. See <a href="#">Removing Entries from the Queue View</a> .

## Using the Alarm Queue

Although the queues have similar features, the Alarm Queue provides more options than the other queues. This section discusses in detail all the features the Alarm Queue supports. Because many of these features are supported by other queue types, this section contains useful information for other queue types.

### Filtering Queue Entries

By default, when an entry is generated it appears on the queue view. You can limit the entries that appear on the queue view by using filters. For example, you might want to filter alarm entries based on their severity, displaying only Severity 1 alarms. You can also use filtering to limit the number of alarms generated at startup.

The filter criteria determine which queue entries should appear on a queue view. The entries that do not meet the criteria are hidden; they do not appear on the queue view, although they exist in the queue.

To filter queue entries, you must:

- Define the filter(s)
- Apply the filter(s)



## Permanent and Temporary Filters

You can create temporary filters that work for the duration of your GDA session, or you can create permanent filters that can be used during any GDA session. You can make temporary filters permanent.

### Creating a Permanent Filter

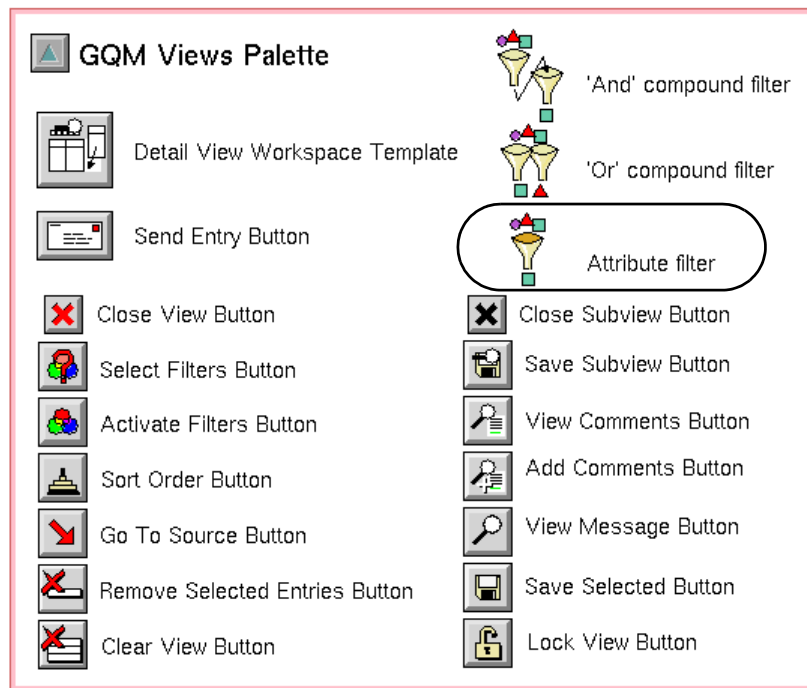
This section describes how to create a filter object that you can save, then reuse during any GDA session.

#### To create a permanent filter:

- 1 In Administrator mode, choose Main Menu > Get Workspace > GQMV-TOP-LEVEL to display the top-level workspace associated with the GQM Views module:



- Click the GQM Views Palette button to display this palette:



- Click the Attribute Filter item to attach it to the cursor, then click to place it on a workspace in your application.
- Display the table for the attribute filter and edit these attributes:

Attribute	Description
Names	The name of the filter. When you apply a filter to a queue, you select this name in a list on the Select Filter dialog box.
Attribute-name	The name of the queue entry attribute whose value determines whether the entry appears in the queue view. For the list of available attribute names, see <a href="#">Entry Attributes Used with Filters</a> .

Attribute	Description
Test	The relational operator to apply to test the value of the Attribute-name value against the Target-value. The operators are: equals, does-not-equal, contains, does-not-contain, greater-than, greater-than-or-equal-to, less-than, less-than-or-equal-to, exists, and does-not-exist.
Target-value	The value to compare to the value of the Attribute-name attribute, using the relational operator specified in Test. The value cannot be an expression.

For example, a filter named `permanent-severity-filter` filters alarm entries based on the `gda-severity` attribute being equal to 1. All queue entries whose `gda-severity` value is 1 appear on the view. The filter table looks like this:



PERMANENT-SEVERITY-FILTER

PERMANENT-SEVERITY-FILTER, a gqs-attribute-filter	
Notes	OK
Item configuration	none
Names	PERMANENT-SEVERITY-FILTER
Gfr uuid	"00609703e2f8-936727661.917-4146"
Attribute name	<code>gda-severity</code>
Test	<code>equals</code>
Target value	1

### Entry Attributes Used with Filters

This section contains two tables: the first lists attributes of queue entries supported by all queues; the second lists those attributes supported by the Alarm Queue.

---

**Caution** Do not use attributes whose names start with an underscore. These attributes are reserved for internal use and can be changed without notice.

---

All queue entries define these attributes:

<b>Queue Entry Attribute</b>	<b>Description</b>
Gqm-comments	The text of the comments associated with the queue entry, as a text-array. This attribute cannot be used in filters.
Gqm-message-text	The text of the queue entry, specified as a text. For Alarm Queue and Explanation Queue entries, GDA generates the message text based on the class name and output path value of the block to which an Alarm Capability or Queue Message block is attached. For Message Queue entries, the message text is generated based on the Entry Text attribute of the Queue Message block.  For example, you can filter entries based on the message text: "gdl-high-value-observation".
Gqm-creation-time	The time when the queue entry was created, specified as a float. The date is stored as the number of seconds since G2 started.
Gqm-priority	This attribute is not used by GDA. You can use it as a basis for filtering entries. Its value is specified as an integer.

Alarm Queue entries define these additional attributes:

<b>Attribute</b>	<b>Description</b>
Gda-require-acknowledgement	The value of the Require Acknowledgement attribute of the Alarm Capability that generates the alarm. The value can be <b>true</b> or <b>false</b> .
Acknowledged	Whether or not the alarm has been acknowledged. The value can be <b>true</b> or <b>false</b> .
S-acknowledge-time	The time when the alarm was acknowledged, specified as a float. See Gqm-creation-time in the previous table.

Attribute	Description
Gda-severity	The value of the Severity attribute of the Alarm Capability that generates the alarm, as an integer.
Gdl-filter-tag	This attribute is not used by GDA. It is provided specifically as an attribute for filtering. Its value is specified as a symbol.

You can subclass queue entry classes to add or modify entry attributes, then use these attributes in filters. For more information about subclassing queue entries, see the *GDA API Reference*.

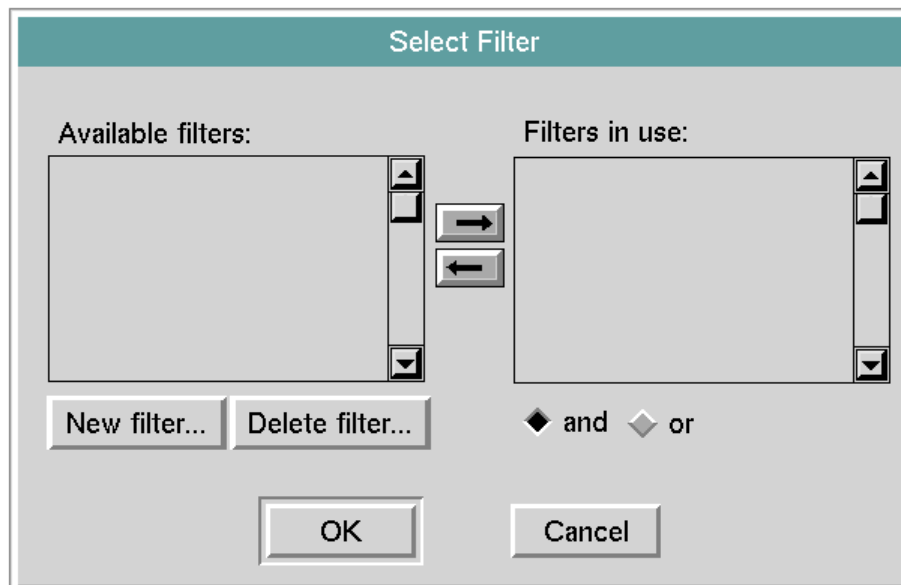
### Creating a Temporary Filter

You can create one or more filters and apply each individually or combine them by using logical operators. A filter is based on the attributes of the queue entry.

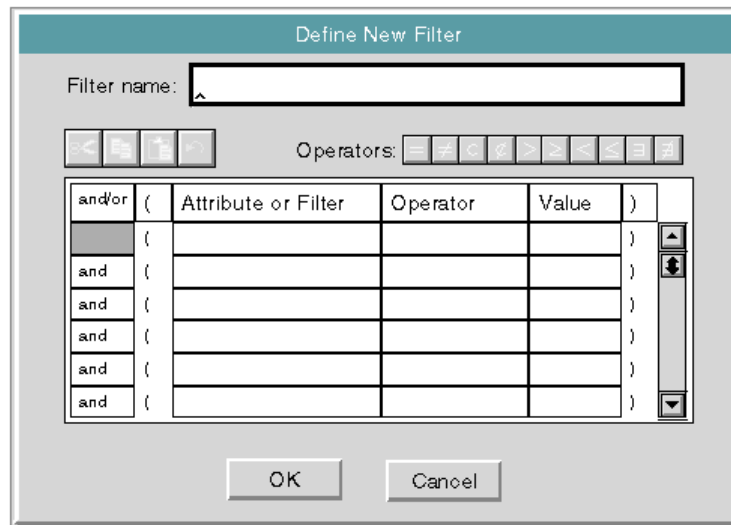
#### To define a filter:



- 1 On the queue view, click the Select Filter button to display this dialog:



- 2 Click the New filter button to display the Define New Filter dialog:



The image shows a dialog box titled "Define New Filter". At the top, there is a text field labeled "Filter name:" with a cursor. Below this is a row of navigation buttons (back, forward, etc.) and an "Operators:" section with a row of operator icons (equals, not equal, greater than, less than, etc.). The main part of the dialog is a table with columns: "and/or", "(", "Attribute or Filter", "Operator", "Value", and ")". The first row is pre-filled with "and/or", "(", "Attribute or Filter", "Operator", "Value", and ")". Below this are several empty rows, each starting with "and" and "(". To the right of the table is a vertical scrollbar. At the bottom of the dialog are "OK" and "Cancel" buttons.

and/or	(	Attribute or Filter	Operator	Value	)
and	(				)
and	(				)
and	(				)
and	(				)
and	(				)

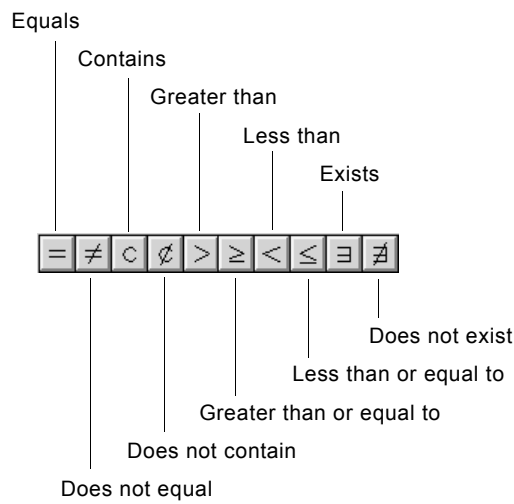
- 3 Enter a symbol for the Filter name, for example, `unacknowledged-alarms`, and press Enter. You use this name later when you apply filters. The filter name can be any length, although only 25 characters are displayed on the Select Filter dialog.
- 4 In the Attribute or Filter field, enter the name of a queue entry attribute, depending on the criteria you want to use.  
For example, to filter Alarm Queue entries based on their acknowledgement status, enter `acknowledged`.

---

**Caution** Be sure to press the Enter key after entering the value; otherwise, the value does not get set.

---

- 5 Click in the Operator field, then click one of the Operator buttons:



For example, to display alarm entries whose acknowledgement status is **false**, click the Equals button.

- 6 In the Value field, enter the value to apply the operator to and press the Enter key.

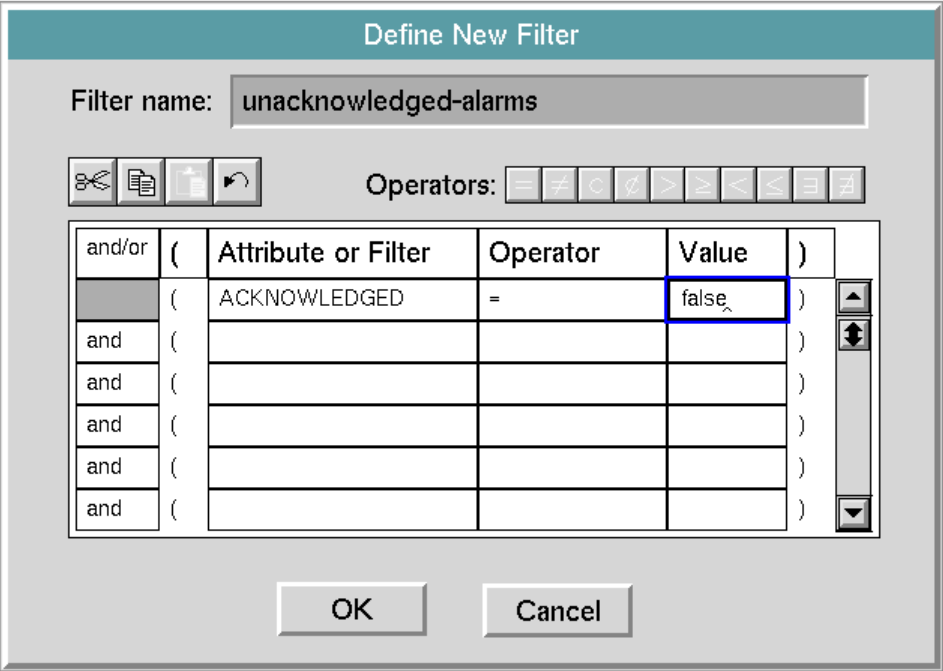
---

**Caution** Be sure to press the Enter key after entering the value; otherwise, the value does not get set.

---

For example, to display alarm entries whose acknowledgement status is **false**, enter **false**. and press Enter.

Here is a fully defined filter that displays unacknowledged alarm entries:



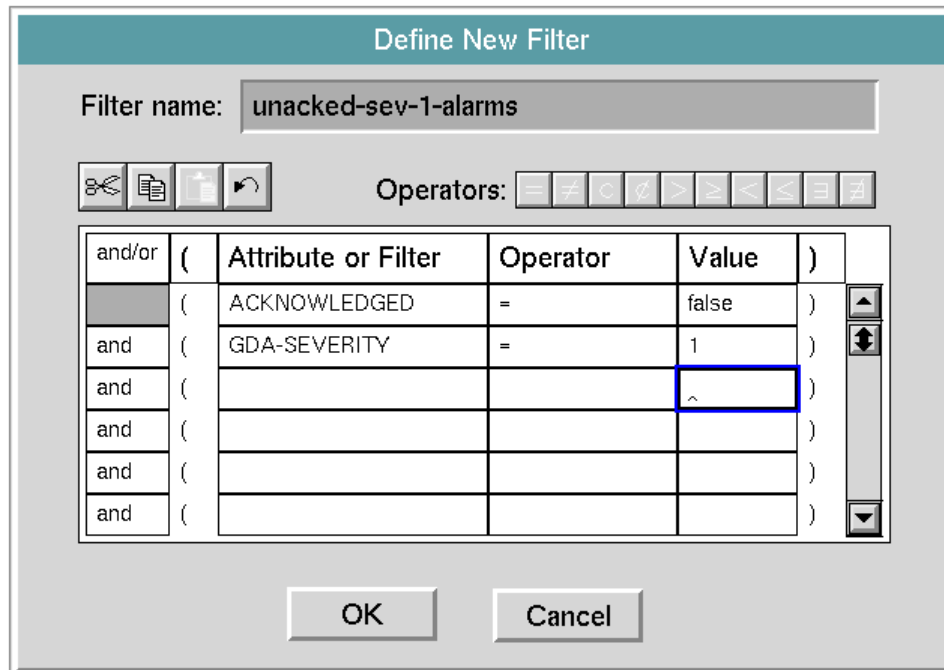
7 Specify additional filter criteria, as needed.

**Tip** You can use the Cut/Copy/Paste buttons to help you specify additional filter criteria. You can copy a cell, a row, or column, then paste it into a selected cell, row, or column.

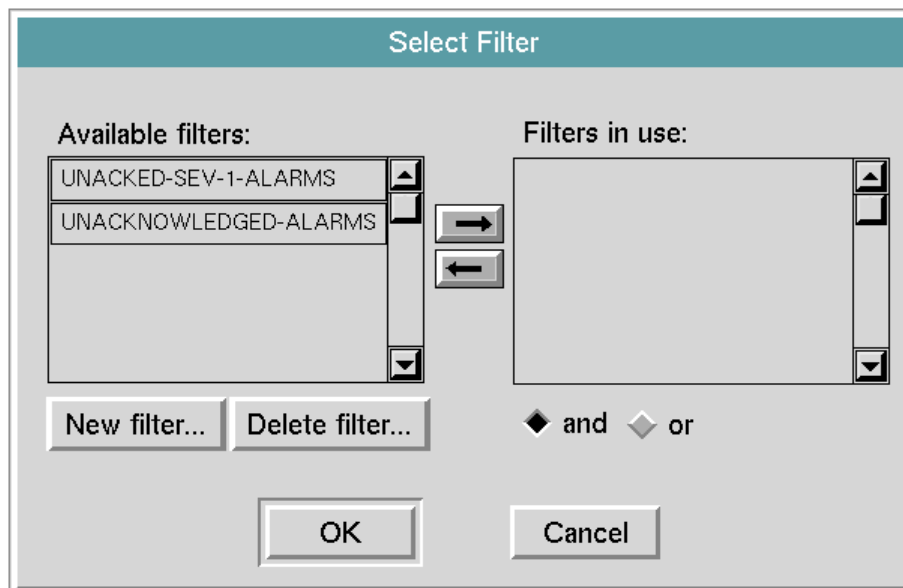
If you specify more than one filter criterion, select a logical operator (AND or OR) to apply by clicking the left-most field to toggle between AND and OR operators.



Here is a filter that causes the Alarm Queue to display unacknowledged alarm entries of severity 1:



- Click OK to create the filter. The name of the newly defined filter appears in the Available Filters list of the Select Filter dialog:

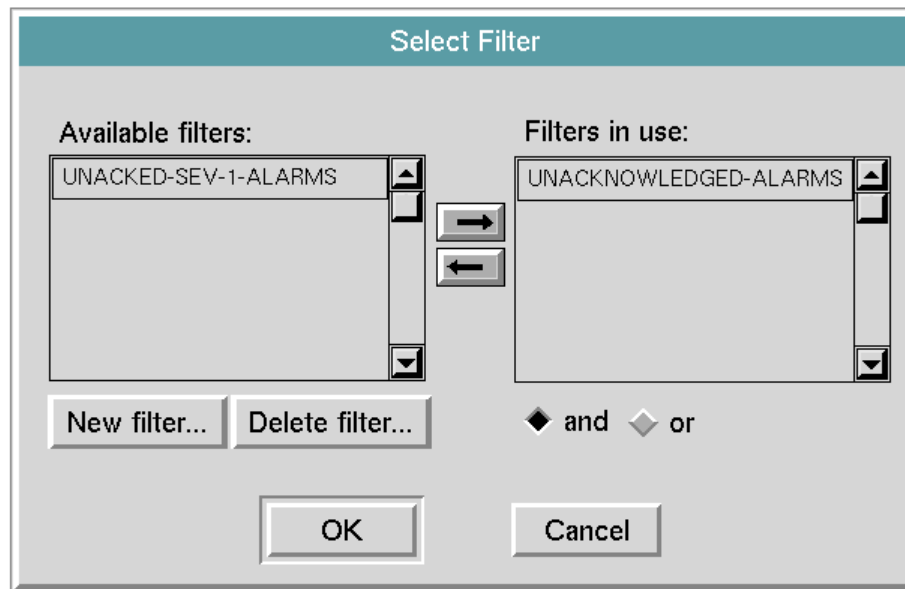


Filters that you create by using the Select Filter button are temporary. You can make temporary filters permanent (see [Making a Temporary Filter](#)

[Permanent](#)), or you can define permanent filters that appear automatically when you use the Select Filter button (see [Creating a Permanent Filter](#)).

- 9 To select the new filter, click the filter name, then click the right arrow button to move it to the Filters in Use list.

This dialog specifies that the unacknowledged-alarms filter is in use:



- 10 If you create additional filters, specify the logic for combining them by clicking the and or or radio buttons under the Filters in use list.

### **Making a Temporary Filter Permanent**

Filters created using the Select Filter dialog box are temporary filters. This section describes how to make these filters permanent.

#### **To make a temporary filter permanent:**

- 1 From the KB Workspace menu, choose **New Button**, then **action-button**, and put the button on a workspace.
- 2 Pause G2, then click the button to display its menu.
- 3 If desired, define a name and/or label.
- 4 Define the button's action as:

in order transfer filter-name to this workspace and make filter-name permanent  
where filter-name is the name of your temporary filter.

- 5 Resume G2.
- 6 Click the button to execute its action, then find and move the filter. Its name appears when you choose **name** from its menu, then press Enter.

### Applying a Filter

Once you have defined a filter, you must move that filter to the Filters in use list on the Select Filter dialog box, then apply the filter to the current queue view to begin filtering queue entries.

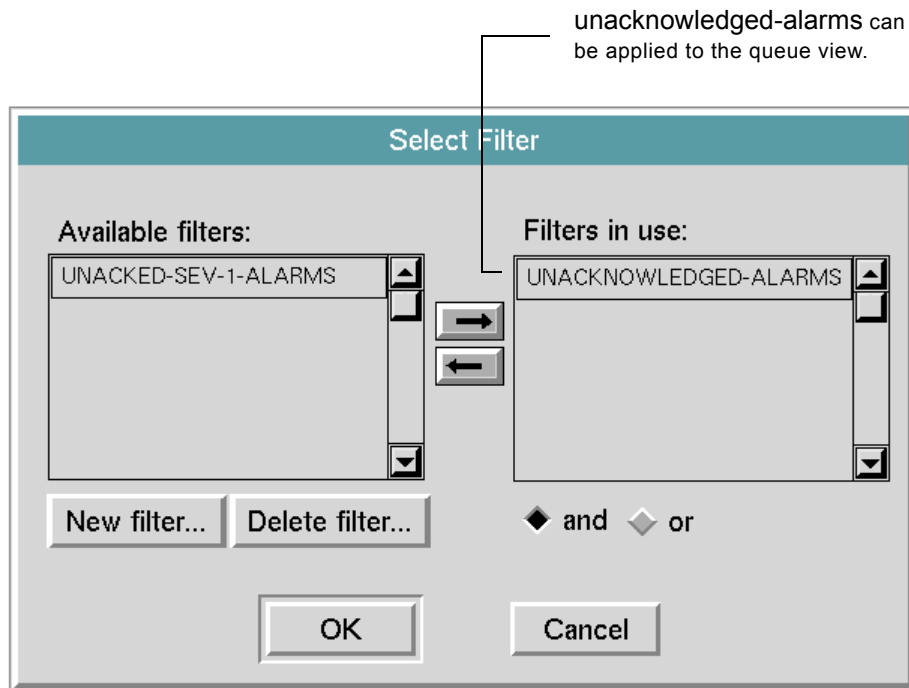
Only queue entries that meet the filter criteria appear on the queue view. Entries that do not meet the filter criteria are considered hidden. The view entry counter indicates the number of entries that appear on the queue view. The total entry counter indicates the number of entries in the queue. The total entry counter appears only when the queue contains entries that are not displayed in the view.

#### To apply a filter to a queue view:



- 1 Click the Select Filter button to display the Select Filter dialog box.
- 2 Select a filter you want to apply and click the right arrow button between the two lists to move the filter to the Filters in use list.
- 3 Click OK when you have moved all the desired filters and close the dialog box.
- 4 Click the Apply Filters button, the second button from the left.

Whenever you create a new filter by clicking the Select Filter button in a queue, the permanent filter appears in the list of available filters:



The button remains depressed to indicate that filtering is in effect. Queue entries that do not satisfy the filter criteria do not appear on the queue view. If any entries are not displayed, the total entry counter appears, indicating the number of entries in the queue.

### Deleting a Filter

You can delete any defined filter, whether it is permanent or temporary.

#### To delete a filter:

- 1 Display the Select Filters dialog box.
- 2 Select the filter you want to delete.
- 3 If the filter is in the Filters in Use list, move the filter from that list to the Available Filters list by clicking the left arrow button. You can only delete filters in the Available Filters list.
- 4 Click the Delete Filter button.
- 5 Click Yes to confirm the deletion.

If the filter is permanent, you can also delete it by selecting it and choosing **delete** from its menu.

## Sorting Entries

By default, queues sort entries in this way:

- The Alarm Queue sorts entries by severity in ascending order, with the highest severity entries at the top.
- The Error, Explanation, and Message Queues sort entries by age in descending order, with the newest entries at the top.

You can sort entries based on any of the displayed queue entry attributes, in ascending or descending order.

You can also specify the column on which to sort initially and the initial sort order. For more information, see [Configuring Sorting Attributes](#).

### To change the sort column:

- ➔ Click the column header you want to use as the sort column. An asterisk (\*) appears to the right of the column header text to indicate that sorting is based on this attribute.

### To change the sort order of queue entries:

- 1 Click the column header you want to use as the sort column. An asterisk (\*) appears to the right of the column header text to indicate that sorting is based on this attribute.
- 2 Click the Sort Entries button to toggle the sort order between ascending and descending.



## Sending Entries to Another Queue

You can send queue entries from one queue to another. The entry that you send must be compatible with the destination queue: the entry must be of the same class as or of a subclass of the entry class of the destination queue.

You can send entries of any type to a Message Queue. You can send entries to other queues only if the entry is of the same class as, or of a subclass of the entry class specified for the destination queue. If no customized entry classes are defined for an application, you can send an entry from an Alarm Queue to another Alarm Queue; from an Error Queue to another Error Queue; and from an Explanation Queue to another Explanation Queue.

You can preserve queue entries by saving or logging them to a file. For more information, see [Saving a Queue Entry](#) and [Logging Queue Entries](#).

### To send an entry to another queue:

- 1 Select the queue entry or entries you want to send to another queue.

---

**Tip** To select more than one contiguous entry, hold down the Shift key and select additional entries. You cannot select non-contiguous entries.

---



- 2 Click the Send Entry button to display a dialog for specifying the name of the destination queue.
- 3 In the Destination Queue field, enter the name of the destination queue. The named queue must exist; otherwise, an error is generated.

The selected queue entries are removed from the source queue.

## Viewing the Details of an Entry

Queue entries contain additional information that you can view by displaying a detail view of the entry.

Error, explanation, and message entries contain user-entered comments that are viewable on a detail view.

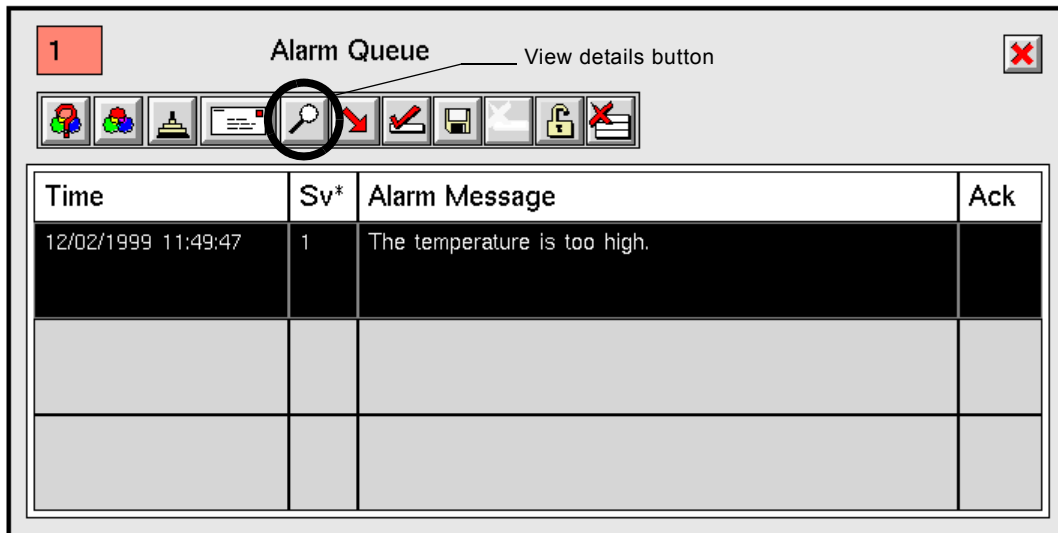
Alarm entries contain all of the following details, viewable on a detail view:

- **Comments** – Text that the end user enters while the entry is in the queue.
- **Advice** – Text that the developer enters when configuring the Alarm Capability, Recurring Alarm Capability, or Queue Message block. Advice provides information to the end user about how to respond to an alarm or message.
- **History** – A chronological list of alarm events that indicate when the alarm condition began and when it ended.
- **Explanation** – Text that GDA generates based on the current value of the block that caused the alarm or message to be sent.

In addition to viewing details, you can perform these functions through the detail view:

- Save individual details to a file.
- Add comments to the queue entry.
- Generate the current explanation for the entry (Alarm Queues only).

This figure shows a selected alarm entry in the Alarm Queue:



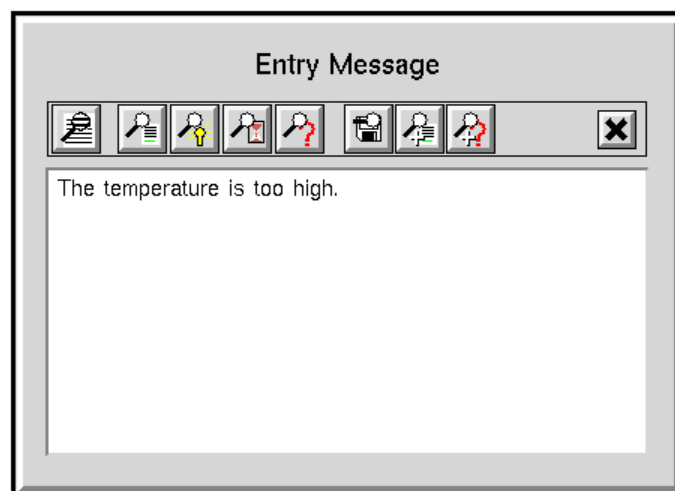
**To show the detail view:**

- 1 Select the entry whose details you want to view.
- 2 Click the View Details button to display the detail view.

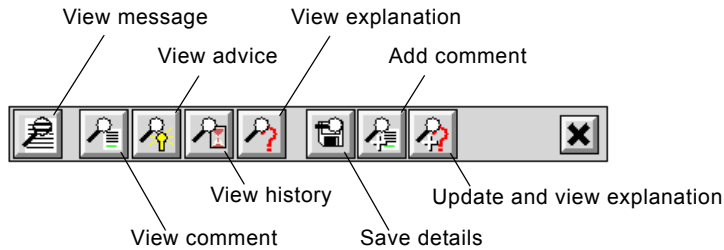


If the message displayed in the queue view has been truncated to fit into the available space, the full message will appear in the detail view, with a scroll bar added if needed.

When the detail view first appears, it displays the text for the queue entry, as this figure shows:



You access the additional information and interact with the detail view by clicking the various buttons. This figure shows the labeled toolbar buttons (other queues do not provide all these buttons):



## Displaying the Entry Message Text

At any time when viewing details, you can display the original entry message text.

### To display the entry message text:



- ➔ Click the View Message button.

## Adding and Viewing Comments

The application user can add comments to the message while viewing the details. New comments are appended to any existing comments.

### To add a comment:



- ➔ Click the Add Comment button and enter a comment in the dialog that appears. Press Enter after entering the comment. Comments are added to the entry and all views of the entry are updated to include the comments.

### To view existing comments:



- ➔ Click the View Comment button.

### To save comments to a file:



- ➔ Click the Save Comment button and specify a file name. If the file exists and is not empty, GDA overwrites the contents of the file.



## Providing and Viewing Advice for Alarms

You might want to provide advice to the application user about what to do if a particular alarm occurs. Using the example above, suppose the High Value Observation block with the Alarm Capability tests the temperature and generates an alarm when the temperature exceeds 70 degrees.

To provide advice, configure the Advice attribute of the Alarm Capability, for example, adding "Turn down the heat".

For more information on configuring advice, see the Alarm Capability in the *GDA Reference Manual*.

Advice is only available for entries in the Alarm Queue.

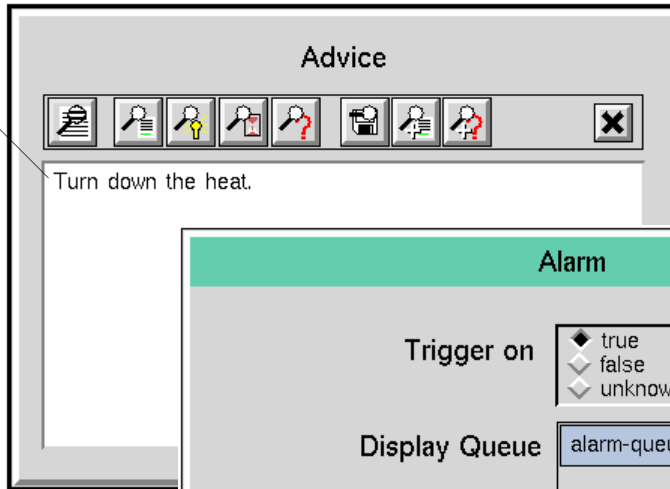
### To display advice:



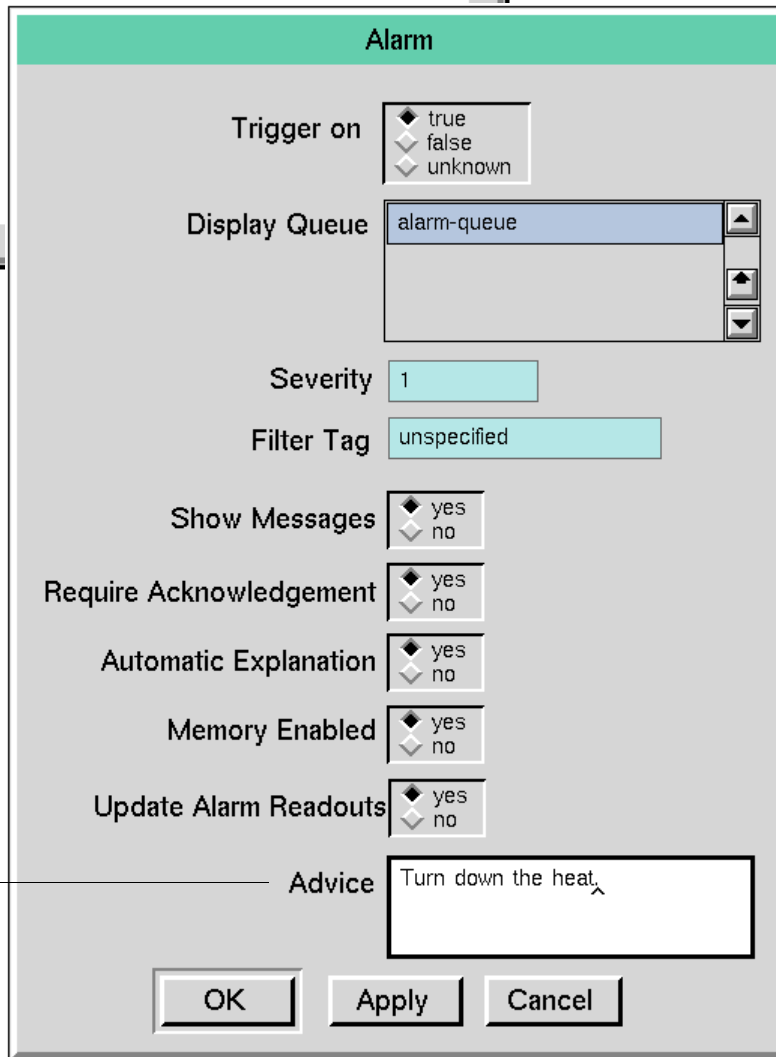
→ Click the View Advice button.

Here is the advice as it appears in the detail view and as it is specified in the Alarm Capability:

This advice...



... is generated from this attribute of the Alarm Capability.



## Viewing an Alarm Explanation

You can view an explanation associated with an alarm entry from the detail view.

To view an explanation for an alarm, both of these conditions must be satisfied:

- The Autogenerate Explanations attribute must be **true** for the Alarm Queue. You can set this attribute by configuring the queue.
- The Automatic Explanation attribute must be **true** for the Alarm Capability block. You can set this attribute by configuring the block.

For more information on configuring GDA blocks to provide explanations, see [Specifying and Generating Explanations](#).

You can also view an explanation of an inference block's output value by selecting the block and choosing the current explanation menu choice. These explanations appear on the Explanation Queue.

### To cause an Alarm Capability to generate explanations automatically:

- 1 Configure the Automatic Explanation attribute of the Alarm Capability to be **yes**.
- 2 Configure the Autogenerate Explanations attribute for the Alarm Queue to be **yes**. For more information, see [Configuring Attributes that Are Specific to Alarm Queues](#).

### To view the current explanation:



→ Click the View Explanation button.

Clicking this button shows the last explanation that has been generated, which might not be for the current alarm.

### To manually generate a new explanation and view it (when explanations are not generated automatically for an alarm message):



→ Click the Update and View Explanation button.

Clicking this button shows the current explanation when the Alarm Queue and Alarm Capability are not configured to generate explanations automatically.

If the state of the diagram changes so that the alarm is no longer active and you generate a new explanation, both the original and the current explanations will be displayed in the detail view.

## Viewing the Alarm History

You can view the history of a particular alarm entry through the detail view. An alarm history describes when an entry went into and out of alarm.

By default, the Alarm Queue reuses alarm entries that represent the same alarm condition. When the Reuse Entry attribute for the Alarm Queue is **yes**, the history

documents when an entry goes into and out of alarm. You can cause the Alarm Queue to generate a new entry each time the alarm condition occurs by setting the Reuse Entry preference of the Alarm Queue to **no**. In this case, when an alarm condition goes back into alarm, a new entry is created. The history of this alarm indicates only the time it went into alarm.

GDA purges all entries in the history when the number of entries exceeds the History Limit attribute, 50 by default. You can change this limit by setting this Alarm Queue preference. For information on configuring Alarm Queue preferences, see [Configuring a Queue](#). For more information about the alarm queue history, see [Configuring Attributes that Are Specific to Alarm Queues](#).

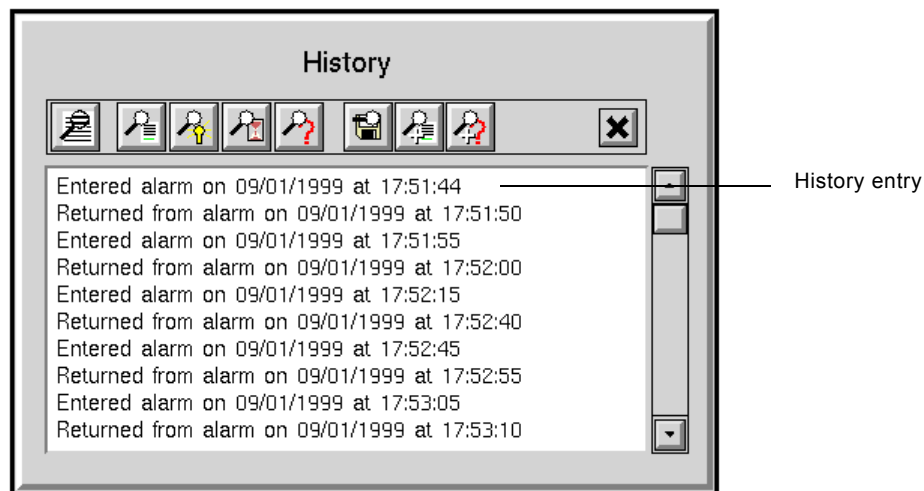
History is only available for entries in the Alarm Queue.

### To view the alarm queue entry history:



➔ Click the View History button.

The alarm history of a particular alarm entry might look like this:



### Saving Details to a File

You can save individual parts of the queue entry's details to a text file.

By default, GDA saves the queue entry details to a text file named `print-queue`, whose name appends the current date and time to the file name, for example:

*print-queue-990914-121716.text*

You can configure the format of the date and time that gets appended by configuring the File-Time Format preference, as described in [Configuring Date and Time Formatting](#).

**To save details of a queue entry to a file:**

- 1 Display the part of the queue entry details you want to save by clicking the appropriate button.

For example, to save the current explanation, click the View Explanation button.



- 2 Click the Save Details button and enter a file name or use the default.

For example, saving the explanation of an alarm entry might result in a file that looks like this:

```
Explanation
09/01/1999 17:52:34 The temperature is too high because
Temperature = 0.978 (threshold: 0)
```

**Showing the Source of an Entry**

Typically, queue entries are associated with a particular block or capability. You can also generate queue entries programmatically, using the API, as described in the *GDA API Reference*.

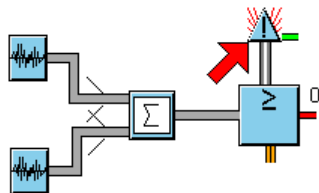
If the source of a queue entry is an item that exists on a workspace, you can show the source of the entry. You can show the source of an Alarm, Explanation, or Error Queue entry; Message Queue entries have no source.

If the source of an entry does not exist on a workspace, showing the source has no effect.

**To show the item that generated an alarm, explanation, or error entry:**

- ➔ Click the Go to Source button.

GDA opens the workspace that contains the source and displays a blinking red arrow next to the block or capability that generated the entry, as this figure shows:



## Acknowledging Alarms

Typically, when an alarm is generated and appears in the Alarm Queue, the end user acknowledges the alarm.

The Require Acknowledgement attribute of the Alarm Capability and the Acknowledge Selected Entries attribute of the Remove Entries button determine whether the user must acknowledge the alarm before it can be removed from the queue:

- If the Acknowledge Selected Entries attribute of the Remove Entries button on the queue view is **yes**, the user can remove the alarm entry without acknowledging it first.
- If the Acknowledge Selected Entries attribute is **no** and acknowledgement is required for the alarm (the default), the user must acknowledge the alarm before it can be deleted from the queue.
- If the Acknowledge Selected Entries attribute is **no** and acknowledgement is *not* required, the **Ack** column automatically displays **Yes** when the entry is posted and the user can delete the entry in one step.

The Acknowledge Selected Entries attribute is described in [Modifying Button Attributes](#).

The Reuse Entry attribute of the Alarm Queue determines whether an entry is reused when an alarm condition occurs again. This attribute affects the alarm history and log; if an alarm entry is reused, its status can be tracked in a single history display or log file. If an alarm entry is not reused, recurrence of an alarm condition causes a new entry to be created, so its history is limited to the single entry that records its creation. The Reuse Entry attribute is described in [Configuring Attributes that Are Specific to Alarm Queues](#).

The Alarm Queue can keep track of the time when an alarm is acknowledged by writing this information to a log file, as described in [Logging Queue Entries](#).

You can filter messages based on whether the alarm has been acknowledged, as described in [Filtering Queue Entries](#).

You can only acknowledge Alarm Queue messages.

### To acknowledge an alarm entry:

- 1 Select the alarm message or messages you want to acknowledge.

---

**Tip** To select more than one contiguous entry, hold down the Shift key and select additional entries.

---



- 2 Click the Acknowledge button.

The Ack (Acknowledged) column updates to indicate that the alarm has been acknowledged and its color changes. You can now remove the alarm from the queue.

This example shows the result of acknowledging an alarm and the same alarm condition occurring again, which generates a new alarm message:

Because this alarm message has already been acknowledged...

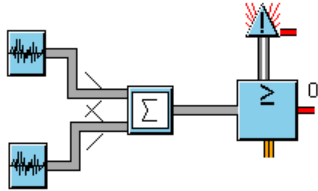
...this alarm message is generated when the same alarm condition occurs.

Acknowledgement status is unacknowledged.

Time	Sv*	Alarm Message	Ack
09/02/1999 09:06:10	2	GDL-HIGH-VALUE-OBSERVATION-XXX-121 is TRUE	
09/02/1999 09:05:50	2	GDL-HIGH-VALUE-OBSERVATION-XXX-121 is TRUE	Yes

Acknowledgement status is Yes.

This example shows an Alarm Capability that specifies Require Acknowledgement as no. When the same alarm condition occurs again, the existing acknowledged alarm highlights and the history is updated.



When the alarm occurs, it is automatically acknowledged, because the Require Acknowledgement attribute is NO.

1
**Alarm Queue**
✕

Time	Sv*	Alarm Message	Ack
09/02/1999 09:18:16	1	GDL-HIGH-VALUE-OBSERVATION-XXX-120 is TRUE	Yes

History

Entered alarm on 09/02/1999 at 09:18:16  
 Returned from alarm on 09/02/1999 at 09:18:31  
 Entered alarm on 09/02/1999 at 09:18:36

Alarm

Trigger on true  
false  
unknown

Display Queue alarm-queue

Severity 1

Filter Class start-symbol-1

Show Messages yes  
no

Require Acknowledgement yes  
no

Automatic Explanation yes  
no

Memory Enabled yes  
no

Update Alarm Readouts yes  
no

When the alarm condition occurs again, the existing alarm message history is updated.



## Saving a Queue Entry

You can save a queue entry to a text file. The text includes a unique ID for the alarm entry, as well as its severity, timestamp, acknowledgement status and time, and history.

By default, GDA saves the queue entry to a text file named `save-selected`, with the current date and time appended to the file name, for example:

```
save-selected-990914-121716.text
```

You can configure the format of the date and time that gets appended by configuring the File-Time Format preference, as described in [Configuring Date and Time Formatting](#).

You can also save queue entry details to a file, as described in [Saving Details to a File](#). You can also log alarm entries to a file, as described in [Logging Queue Entries](#).

### To save a queue entry to a file:

- 1 Select the queue entry you want to save.
- 2 Click the Save Entry button and enter a file name or use the default.



Saving an alarm entry might result in a file that looks like this:

```
***** Summary: GDA-ALARM-ENTRY-00609703e2f8-936278684.549-926 *****
Severity: 1 At: 09/02/1999 09:24:44 CT: 09/02/1999 09:24:44

Acknowledged on 09/02/1999 at 08:41:25
----- History -----
Purging history because the maximum number of elements, 50, was
exceeded at time 09/02/1999 09:32:54
Entered alarm on 09/02/1999 at 09:32:59
Returned from alarm on 09/02/1999 at 09:33:04
Entered alarm on 09/02/1999 at 09:33:14
Returned from alarm on 09/02/1999 at 09:33:19
Entered alarm on 09/02/1999 at 09:33:24
Returned from alarm on 09/02/1999 at 09:33:49
Entered alarm on 09/02/1999 at 09:34:04
Returned from alarm on 09/02/1999 at 09:34:14
Entered alarm on 09/02/1999 at 09:34:19
Returned from alarm on 09/02/1999 at 09:34:24
Entered alarm on 09/02/1999 at 09:34:29
Returned from alarm on 09/02/1999 at 09:34:54
Entered alarm on 09/02/1999 at 09:34:59
Returned from alarm on 09/02/1999 at 09:35:04
Entered alarm on 09/02/1999 at 09:35:09
Returned from alarm on 09/02/1999 at 09:35:14
Entered alarm on 09/02/1999 at 09:35:19
```

## Removing Entries from the Queue View

You can remove individual entries or all visible entries from the queue view. By default, when removing individual entries from an Alarm Queue view, the alarm must first be acknowledged. When removing all entries from a queue view, the entries need not be acknowledged because they will automatically be acknowledged before being deleted.

Any entries that are not currently visible due to filtering or locking remain in the queue.

When you remove an entry from a queue, the entry remains in any other queues in which it exists. If the entry does not exist in any other queues, the entry is permanently deleted.

By default, when you remove entries from the queue view, GDA does not confirm the deletion. You can configure the Confirm Deletions attribute of a queue to cause the queue to display a confirmation dialog before removing entries, as described in [Confirm Deletions](#).

### To remove individual entries from a queue view:

- 1 Select the entry or entries you want to remove.

---

**Tip** To select more than one contiguous entry, hold down the Shift key and select additional entries.

---

- 2 If you are removing entries from an Alarm Queue view, acknowledge the entry if it is not already acknowledged. For details, see [Acknowledging Alarms](#).



- 3 Click the Remove Entry button.

### To remove all visible entries from a queue view:



- ➔ Click the Clear View button.

To remove all entries from a queue view, both visible and hidden, you must first deactivate any current filters and unlock the queue, as needed. For details, see [Filtering Queue Entries](#) and [Locking the Alarm Queue View](#).

## Locking the Alarm Queue View

You can prevent any new entries from appearing in an Alarm Queue view by locking the view. Any entry sent to the queue while it is locked is hidden from the view. When you unlock the view, the hidden messages appear in the queue.

When new alarm entries are generated, the queue displays an additional counter that indicates the total number of messages in the queue.

Locking is only available in the Alarm Queue.

**To lock a queue view:**

→ Click the Lock View button.

The button remains depressed indicating that locking is in effect. The icon changes its appearance and the lock turns red.

This Alarm Queue is currently locked, as indicated by the depressed Lock View button. The total entry counter indicates that the queue contains 2 messages, although the second message was posted after the view was locked, so it is not visible, as the view entry counter indicates.

View entry counter

The Lock View button is clicked.

Total entry counter

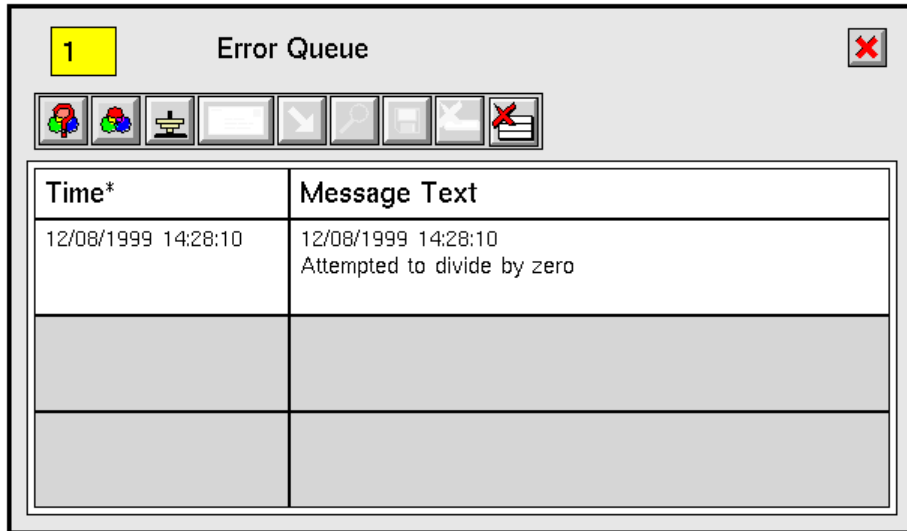
Time	Sv*	Alarm Message	Ack
09/02/1999 10:09:25	2	GDL-HIGH-VALUE-OBSERVATION-XXX-121 is TRUE	

**To unlock a queue view:**

→ Click the Lock View button when it is depressed.

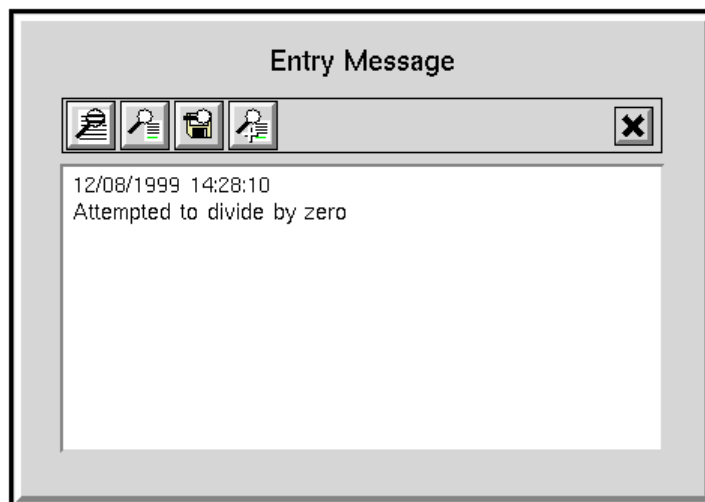
# Using the Error Queue

The Error Queue appears whenever an error occurs in a GDA diagram. For example, this error occurred when an attempt was made to divide by zero:



**To view details about the error message:**

- 1 Select the error message in the queue.
- 2 Click the View Details button. If the message displayed in the queue view has been truncated to fit into the available space, the full message will appear in the detail view, with a scroll bar added if needed. In this example, the detail view shows the same error message text shown in the queue view:



From the detail view, you can add and view comments, save the error message to a text file, and view the error message (if, for example, the detail view is displaying a comment), as described in [Viewing the Details of an Entry](#).

The error entry is also stored in the `error` attribute on the block where the error occurred. You can clear the error by removing the error from the queue view or resetting the block from the block's menu.

From the queue view, you can also perform these functions:

- Filter error messages, described in [Filtering Queue Entries](#)
- Change the sort order of the entries, described in [Sorting Entries](#)
- Send entries to another queue, described in [Sending Entries to Another Queue](#)
- Save entries to a text file, described in [Saving a Queue Entry](#)
- View the source of the error, described in [Showing the Source of an Entry](#)
- Delete entries, described in [Removing Entries from the Queue View](#)

## Using the Explanation Queue

The Explanation Queue displays explanations for the output values of inference blocks. Explanations of alarms appear on a detail view of the Alarm Queue.

Using the Explanation Queue, you can:

- Add and show comments for the explanation
- Save the explanation and comments to a text file

To perform these tasks, select the explanation and access the detail view. For information on how to access the detail view and manipulate the explanation and comments, see [Viewing the Details of an Entry](#).

From the queue view, you can also perform these functions:

- Filter explanations, described in [Filtering Queue Entries](#)
- Change the sort order of the entries, described in [Sorting Entries](#)
- Send entries to another queue, described in [Sending Entries to Another Queue](#)
- View the source of the explanation, described in [Showing the Source of an Entry](#)
- Save entries to a text file, described in [Saving a Queue Entry](#)
- Delete entries, described in [Removing Entries from the Queue View](#)

For general information on using explanations, see [Specifying and Generating Explanations](#).

# Using the Message Queue

The Message Queue appears when a message is posted to the queue. Using the Message Queue, you can:

- Add and show comments for the message
- Save the message and comments to a text file

To perform these tasks, select the message and access the detail view. For information on how to access the detail view and manipulate the message and comments, see [Viewing the Details of an Entry](#).

From the queue view, you can also perform these functions:

- Filter messages, described in [Filtering Queue Entries](#)
- Change the sort order of the entries, described in [Sorting Entries](#)
- Send entries to another queue, described in [Sending Entries to Another Queue](#)
- Save entries to a text file, described in [Saving a Queue Entry](#)
- Delete entries, described in [Removing Entries from the Queue View](#)

## Sending Messages to Queues Using the Queue Message Block

The Queue Message block enables you to send messages to any queue. The target queue handles the message as it would handle queue entries sent to the queue by other means. The block's Configuration dialog box has changed for this release, adding attributes that apply to all the queue types.

### To send a message to any queue:

- 1 Attach a Queue Message block to the output control stub of any block.
- 2 Click on the Queue Message block and select **configure** to display the Configure dialog box.

The screenshot shows the 'Queue Message' configuration dialog box. It has a title bar with the text 'Queue Message'. The dialog contains several fields and controls:

- Display Queue:** A list box with three items: 'explanation-queue', 'message-queue' (which is selected and highlighted in black), and 'error-queue'. There are up and down arrow buttons to the right of the list.
- Entry Priority:** A text input field containing the value '0'.
- Severity:** A text input field containing the value '1'.
- Require Acknowledgement:** A radio button group with two options: 'yes' (selected) and 'no'.
- Filter Tag:** A text input field containing the value 'unspecified'.
- Alarm Status:** A text input field containing the value 'in-alarm'.
- Belief Value:** A text input field containing the value '0.5'.
- Belief Status:** A radio button group with three options: 'true', 'unknown' (selected), and 'false'.
- Entry Text:** A large, empty text area.
- Advice:** A large, empty text area.

At the bottom of the dialog are three buttons: 'OK', 'Apply', and 'Cancel'.

- 3** To send a message to the Error, Explanation, or Message Queue.
  - a** Select the queue in the Display Queue list.
  - b** Enter the text of the message in the Entry Text field, then press Enter.
  - c** Press OK.
- 4** To send a message to the Alarm Queue:
  - a** Select the queue in the Display Queue list.
  - b** Enter the text of the message in the Entry Text field, enter a value for the Severity attribute, choose whether acknowledgement is required for the message, and, optionally, enter text in the Advice field, then press Enter.
  - c** Press OK.
- 5** When the Queue Message block receives a control signal, it sends the message.

The Entry Priority attribute is not used.

You should not change the value of the Alarm Status attribute.

The Filter Tag attribute is not used by GDA but is available for customizing messages and using filters. Using the attribute enables an application to distinguish between alarm sources.

The Belief Value and Belief Status attributes are described in the *GDA Reference Manual*.



# Custom Block Wizard

---

*Describes how to create subclasses of GDA custom block classes, which are available on a palette and which you can use in diagrams.*

Introduction	167
Using the Custom Class Wizard	168
Creating a New Custom Subclass	169
Customizing the Block Evaluator	180
Editing an Existing Custom Subclass	190
Deleting an Existing Custom Subclass	191
Custom Class Reference	193
General	194
Peer Input	195
Multiple Invocations	200
Single Source Encapsulation	206



## Introduction

The **Custom Class Wizard** enables you to create subclasses of GDA blocks. These new blocks can provide functionality beyond what built-in GDA blocks provide. You can access these custom subclasses from user-defined palettes and clone the blocks as needed for use in a diagram.

There are two general categories of blocks for which you can create a custom subclass, using the Custom Class Wizard:

- **Custom blocks** enable you to create new classes of GDA blocks that evaluate G2 procedures. You use the Custom Class Wizard to create subclasses of one of three different classes of GDA block:
  - **General custom blocks** perform evaluations on single or multiple inputs when the block needs to reference specific inputs by port name. The Difference block on the Arithmetic palette is a kind of General custom block because the bottom input is subtracted from the top input, which requires that the block evaluator refer to each input by name.
  - **Peer Input custom blocks** perform evaluations on single or multiple inputs when the block treats all inputs equally. The Summation block on the Arithmetic palette is a peer input block because all inputs are added together without requiring that they be uniquely identified.
  - **Multiple Invocations custom blocks** allow control over how the block processes additional inputs that the block receives while it is processing an existing input. The block processes its inputs the same way as a General custom block. The Data Delay block on the Data Control palette is a kind of multiple invocations block.
- **Encapsulation blocks** enable you to manage the complexity of a GDA diagram by placing a portion of the diagram on a subworkspace.
  - **Single Source Encapsulation (SSE)** blocks enable you to create multiple instances with the same subworkspace diagram. The SSE block enables you to make changes in all instances of the block throughout your diagram by changing the master copy of the block.
  - **Simple Encapsulation** blocks are similar to Single Source Encapsulation blocks, except that you cannot propagate changes in the definition of the encapsulation to other blocks in the diagram. For more information, see the *GDA Reference Manual*.

## Using the Custom Class Wizard

You use the Custom Class Wizard to create new subclasses of GDA blocks and to edit existing subclasses. You can create subclasses of existing GDA custom block classes or create custom block subclasses.

The following sections describe how to create a General custom block that raises its input value to a power ( $x^n$ ). The example shows how to edit the existing definition and edit the procedure for the custom class. The technique applies to all types of custom subclasses. Refer to the individual sections that follow for details specific to each type of custom subclass.

---

**Note** This is a simple example used to illustrate how to create and edit a custom subclass with a custom block evaluator; however, you could obtain the same functionality using the Arithmetic Function block on the Functions palette.

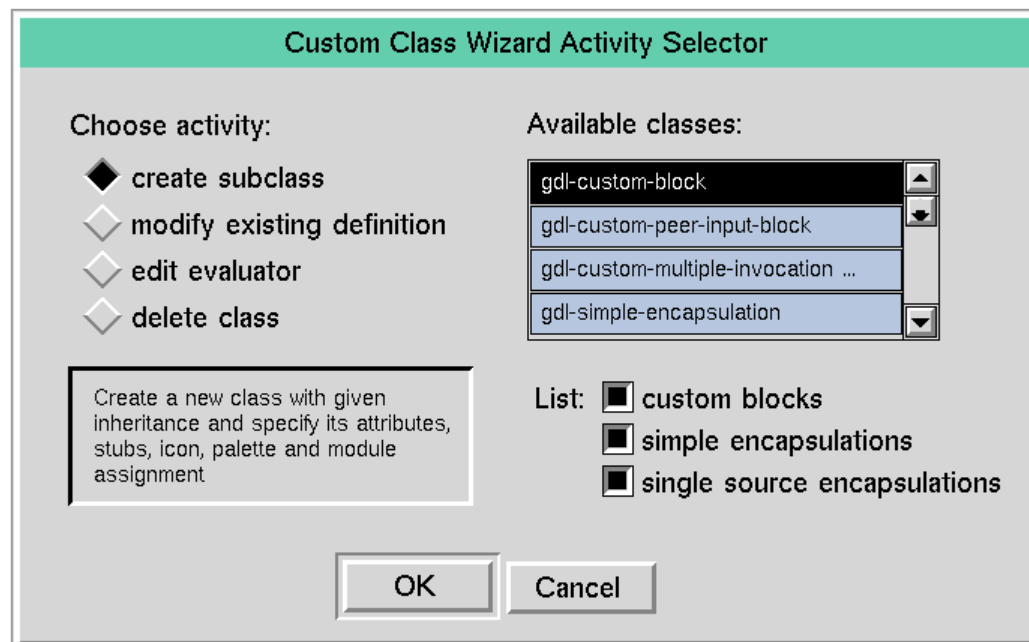
---

## Creating a New Custom Subclass

Follow these steps to create a new custom subclass.

**To create a new custom subclass:**

- 1 Choose Wizards > Custom Class Wizard from the top menu bar to display the Custom Class Wizard Activity Selector:



- 2 Choose create subclass under the Choose activity area at the left.

GDA displays all the available classes from which the new custom subclass can inherit its definition.

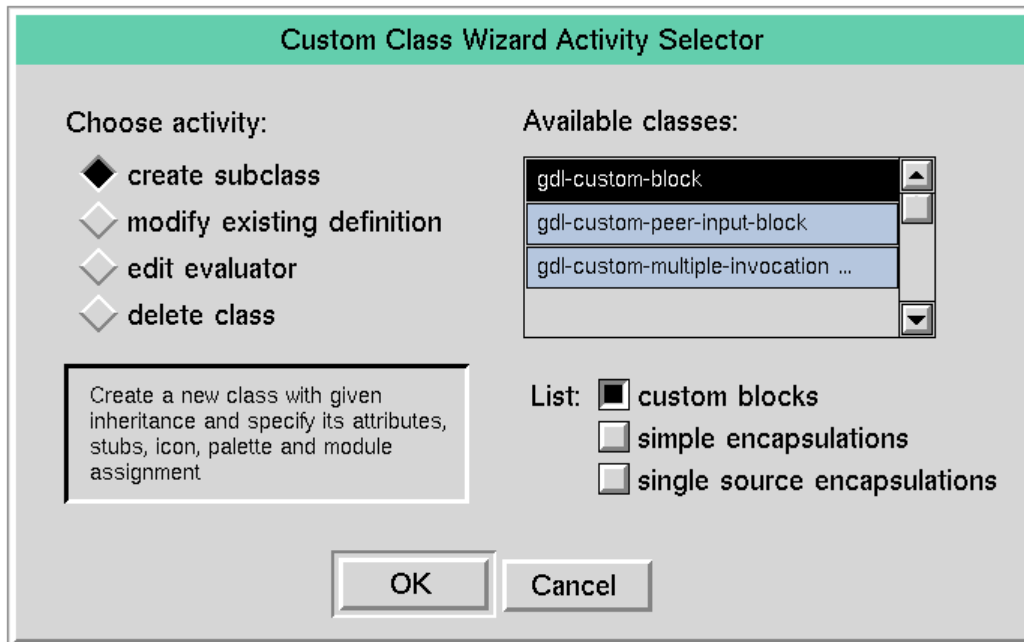
**Note** The list includes standard GDA classes, as well as custom subclasses whose definitions are based on GDA classes. Thus, you can create custom subclasses that inherit from other subclasses, thereby creating a hierarchy of custom block classes.

---

- 3 To narrow the scope of the list of available classes, select only the category of available classes that you want to see from the check boxes at the lower right of the dialog.

By default, all the available categories are selected. Click on a check box to toggle the selection off and on. GDA automatically updates the list of available classes to correspond with the selected class or classes.

For example, here is the dialog with only the three custom block classes displayed in the list of available classes:



- 4 Choose one of the available classes as the superior class for the custom subclass.

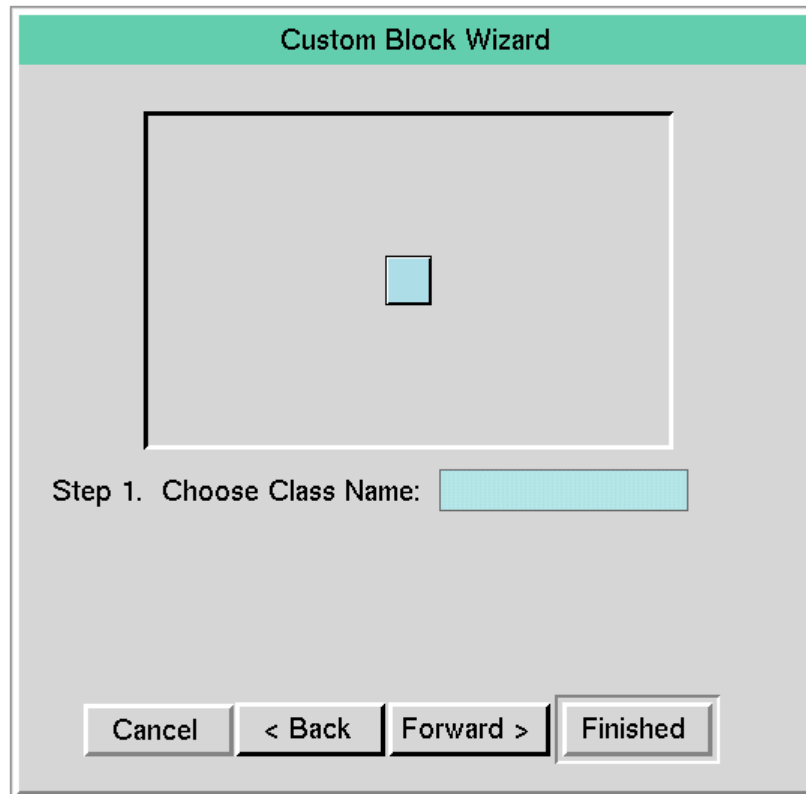
For example, to create a custom block that raises an input to a power, choose `gdl-custom-block`. The following table maps the GDA class names that appear in the Activity Selector dialog to the GDA block types:

GDA Class Name	GDA Block Type
<code>gdl-custom-block</code>	General
<code>gdl-custom-peer-input-block</code>	Peer Input
<code>gdl-custom-multiple-invocations-block</code>	Multiple Invocations
<code>gdl-simple-encapsulation</code>	Encapsulation
<code>gdl-single-source-encapsulation</code>	Single-Source Encapsulation

- 5 Click OK to accept the selections.

## Specifying the Class Name

The Custom Class Wizard displays the default icon for the class you selected in the Custom Class Wizard Activity Selector dialog, and displays the first step in the sequence:



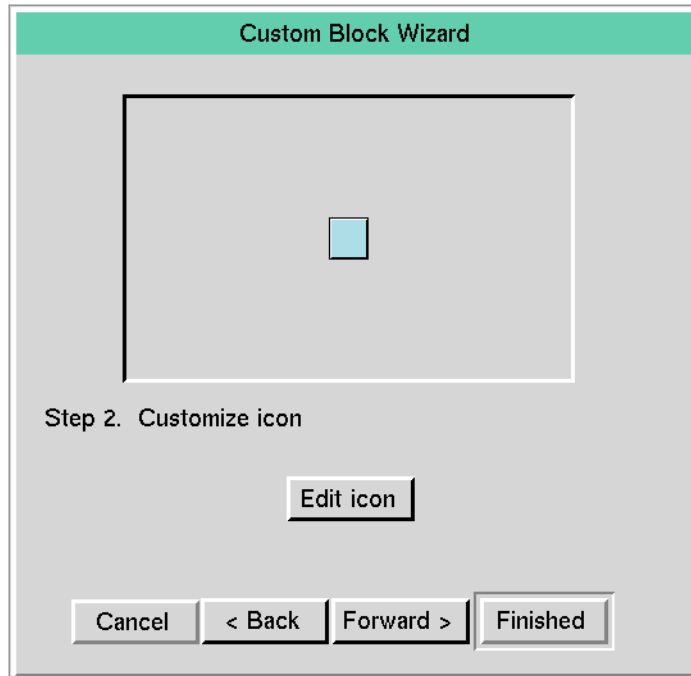
### To specify the class name:

- ➔ Enter a unique symbol for the custom class name in the dialog.

When editing an existing custom class, as described in [Editing an Existing Custom Subclass](#), existing class names appear in the list.

## Customizing the Icon

In the second step, the Custom Class Wizard prompts you to edit the icon:



### To edit the icon:

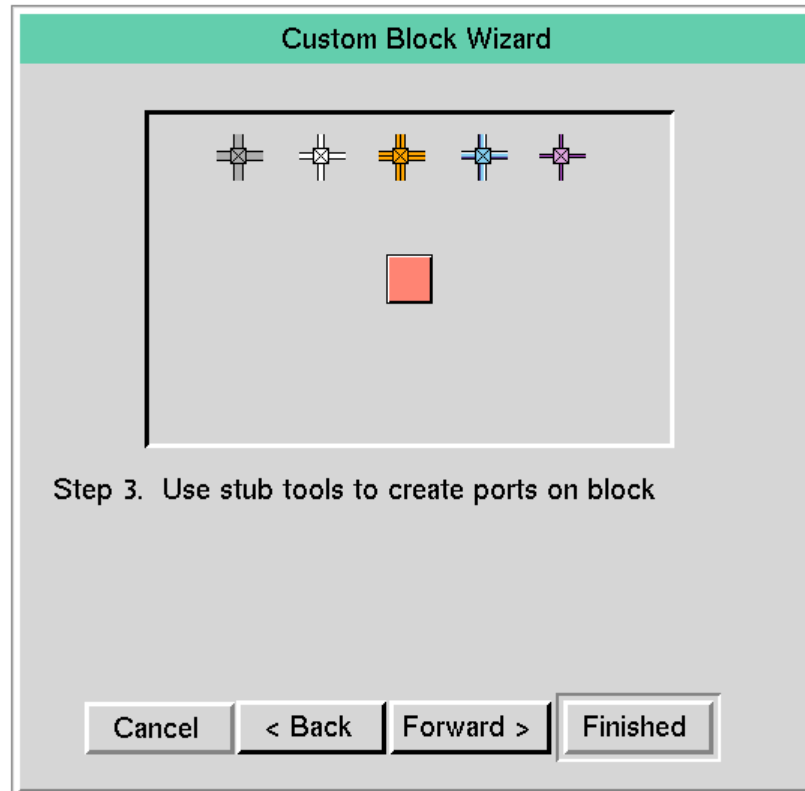
- ➔ Click on the Edit icon button to display the G2 Icon Editor. See G2 documentation for information about using the Icon Editor.

For example, here is the General custom block icon whose color has been changed:



## Customizing the Connection Stubs

In the third step, the Custom Class Wizard prompts you to create input and output ports on the block. The wizard displays Stub Tools across the top of the dialog for creating the ports:



For example, for the custom block created earlier, add a single named data input path and a single named data output path.

### To add ports to a block:

- 1 Drag a stub of one of the Stub Tools to the side of the block on which you want to add an input stub.

To create an input stub, drag the *bottom* or *right* stub on a Stub Tool into the desired side of the block, and click to place the stub; to create an output stub, drag the *top* or *left* stub on a Stub Tool into the desired side of the block, and click to place.

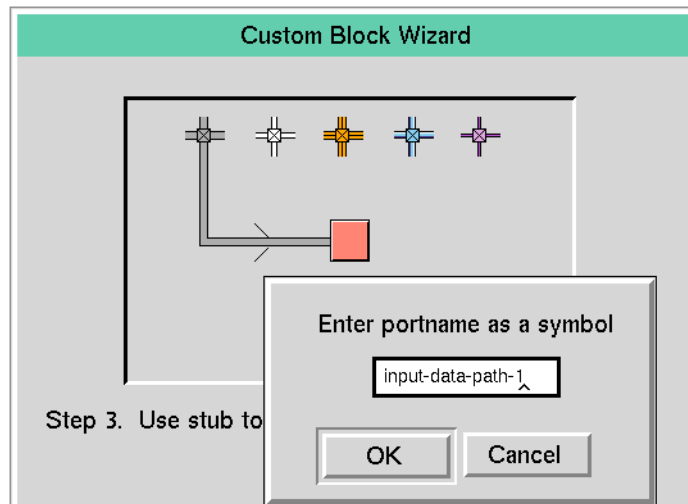
You can also drag an individual stub tool to the left or right of the block, then connect the stub to the block. Creating stubs this way might make it easier to remember which stub to connect to the block.

---

**Tip** The GDA convention for data and inference paths is to add input stubs to the left or top side of a block, and add output stubs to the right or bottom side of the block. For control paths, input stubs enter the top of the block and output stubs go out the bottom of the block.

---

The wizard displays a dialog for entering the name of the port:



- 2 Specify a new name for the input stub, or use the default name, and click OK.

The Custom Class Wizard disconnects the Stub Tool from the block, leaving a stub on the block icon. The block uses the name you specify in the block evaluator for the block. For more information, see [Customizing the Block Evaluator](#).

You can name the input and output ports, using names that are appropriate for the behavior of the block. The GDA convention for an input port is `dp-in` (or `dp-in-1`, etc., for multiple input ports) and `dp-out` for an output port.

In this example, we name the input port of the Power Block `base` to indicate that the input data value is the base value that is raised to a power.

- 3 Repeat for each input and output path on the block.

In this Power Block example, we name the output path `result`.

---

**Note** The port names should be different for each port, and they cannot be reserved words in G2, e.g., `in` and `out`. See the *GDA Reference Manual* for more information about creating stubs, including stub naming conventions.

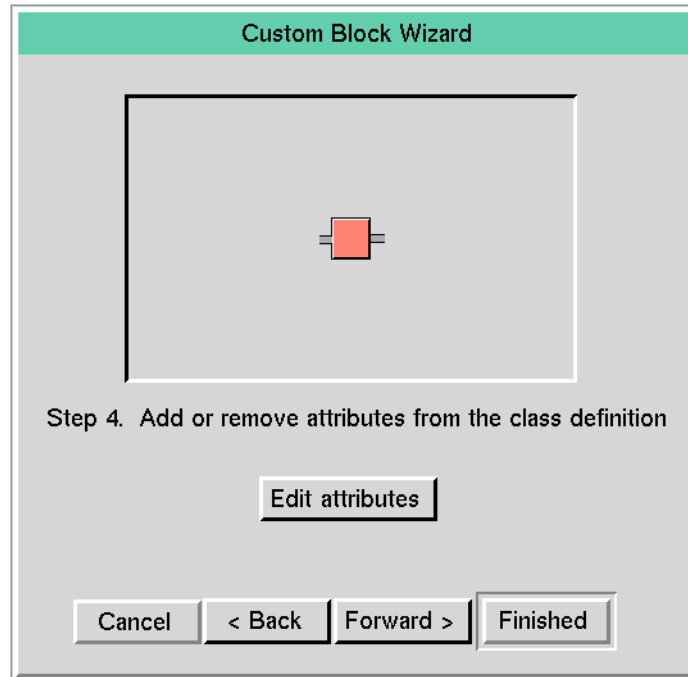
---

When you create Peer Input data blocks, you only need to name the output ports. For more information, see [Declaring Input and Output Path Local Names](#).



## Customizing the Attributes

After you add stubs to the block icon, you are prompted to edit the attributes of the block. The wizard displays this dialog with a button for editing the attributes:



For example, the custom power-block class that raises the input value by a power might raise the input data value to the value of the Exponent attribute. The Custom Class Wizard automatically generates a configuration dialog for configuring the class-specific attributes of the block.

**To add a simple attribute to the block:**

- 1 Click the Edit Attributes button to display the Attribute Editor:

**Attribute Editor**

1. Attribute Name:

2. Value type:

- float
- integer
- quantity
- symbol
- text
- truth-value
- not typed

3. Class of attribute:

- simple value
- parameter
- variable
- other

4. Additional properties:

Initial value:

History age:

History points:

5. Apply or cancel attribute edits

Current class-specific attributes

Current inherited attributes

The Attribute Editor enables you to add new attributes to the definition or delete existing attributes. For simple attributes, you can specify the name, data type, and initial value. You can also add attributes that are variables or parameters, in which case you can specify the variable or parameter class, and you can specify how the attribute keeps its history. GDA provides default variable and parameter class names in the Class of Attribute field.

- 2 Enter a unique symbol in the Attribute Name field and press the Return key.  
In the Power Block example, enter **exponent** as the name of the attribute.
- 3 Select a data type from the Value type list.  
In the Power Block example, click **integer** as the data type for the exponent.

GDA will verify the user entry to ensure that it matches the data type you select. Select **not typed** to allow the user to enter any type of data as the value of the attribute.

---

**Note** To cause GDA to run more efficiently, use the most specific Value type you can.

---

If you select **symbol** as the Value type, the **Symbol enumeration** button is enabled. Select this button to specify the symbolic values for the attribute.

In the Power Block example, the Class of Attribute is **simple value**.


- 4 Enter the default value for the attribute in the Initial value field.  
In the Power Block example, the initial value for the Exponent attribute is 1.
- 5 Click Apply to add the attribute to the list of current class-specific attributes.
- 6 Repeat these steps to add as many attributes as desired.
- 7 When you are finished, click OK.


## Specifying the Palette and Module

Next, the Custom Block Wizard prompts you to enter the name of the palette on which an instance of the custom class will appear, and the name of the module in which the class definition is stored:

Custom Block Wizard

Step 5. Select a palette for the block, or enter a new palette name and its module assignment:

Palette Name:  

Module:  

The palette is associated with the module. When you create a new palette, you assign the palette to a particular module. If you choose an existing palette, GDA fills in the module for you automatically.

**To specify the palette on which the block is to appear:**

➔ Enter the name of a custom palette in the Palette Name field, or scroll through the list of palettes and choose an existing palette.

**To specify the module in which the custom class definition is stored:**

➔ Enter the name of an existing module in the Module field.

The default module is `gdaapps`, which is the name of the top-level module in the GDA hierarchy. If you specify another module, be sure that the module exists before you create a custom class to be stored in this module. A better choice of module names for a sample custom block would be `test` or `example`.

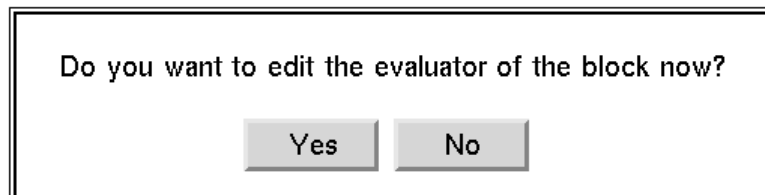
## Applying the New Class Definition

When you have completely specified the custom class, the last step is to commit your changes.

**To apply the new class to the menus:**

1 Click the Finished button.

If you have created a subclass of one of the three types of custom blocks, as opposed to one of the encapsulation blocks, GDA displays a dialog asking if you want to edit the block evaluator now.

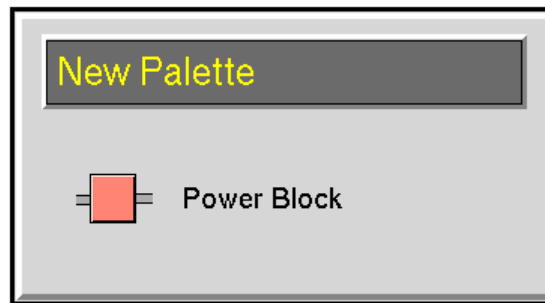
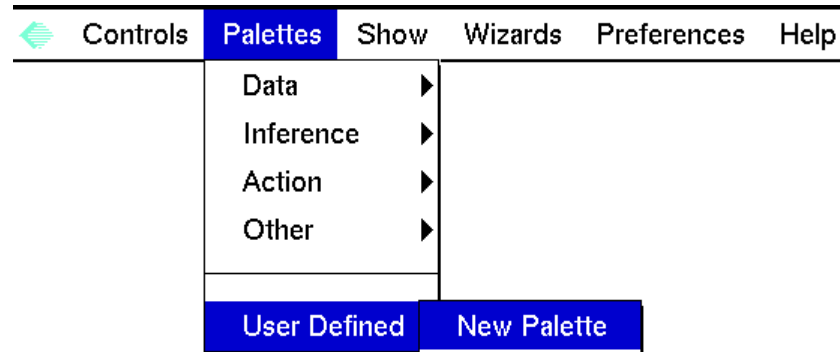


2 Click No to add the custom class to the palettes first.

You can go back and edit the block evaluator later, as [Customizing the Block Evaluator](#) describes.

## Cloning and Configuring the Custom Block

You find the user-defined palettes that contain instances of custom classes in the Palettes menu in the User Defined submenu. For example, if you specify a palette named New Palette, you would see the following menu item under the User Defined menu, which would display the following palette:

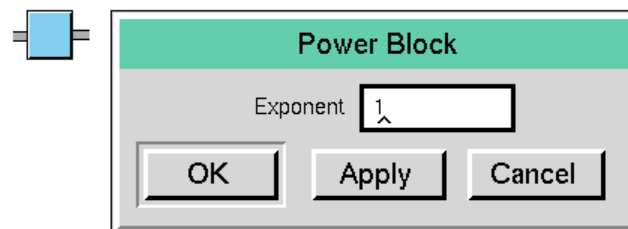


### To configure the class-specific attributes of the custom class:

➔ Clone the custom block from the palette, and choose configure.

GDA displays the automatically generated configuration dialog for the block.

Here is an instance of the power-block class with its configuration dialog, which contains the Exponent attribute:



If you create a custom class with an attribute that contains a variable or parameter, the variable or parameter appears in the custom block's table.

# Customizing the Block Evaluator

When creating custom subclasses of one of the three types of custom blocks (a General, Peer Input, or Multiple Invocations custom block), you must also create a procedure that the custom block executes when it receives a value on its input path(s). This procedure is called the **block evaluator**.

See the *G2 Reference Manual* for general information on G2 procedures.

The Custom Class Wizard automatically generates a block evaluator template for any custom block that you create. The evaluator template provides all the code needed to obtain the value of all the input paths and set the value of all the output paths that the block defines. You edit the body of the procedure to create the custom procedure that the block executes.

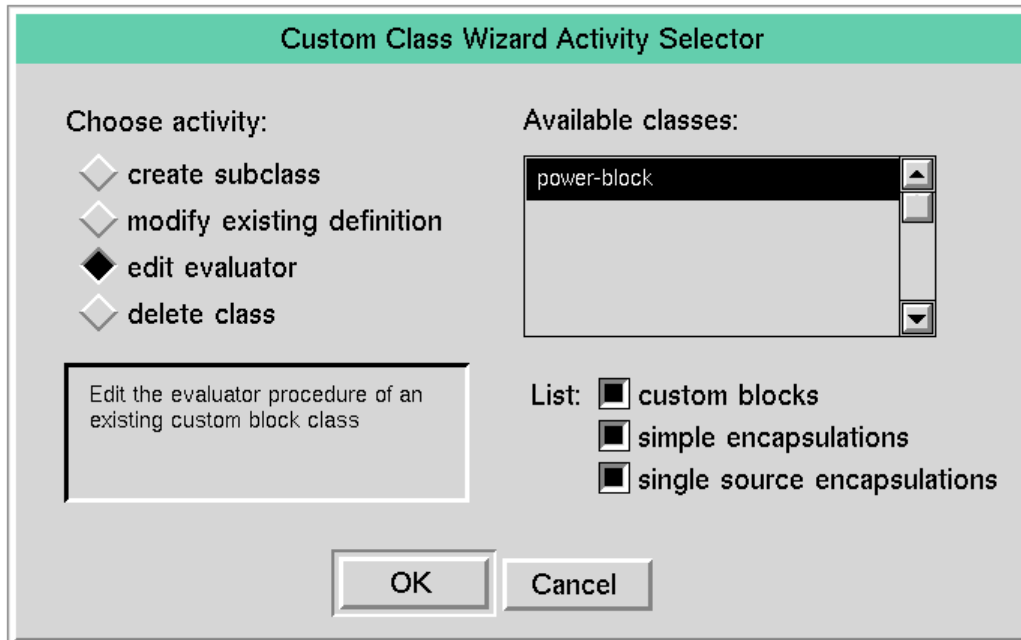
## Displaying the Block Evaluator

You can edit the evaluator for a custom block class when you initially specify the custom class, or you can create the custom class first and go back and edit the procedure later.

For example, in [Creating a New Custom Subclass](#), you created a new custom block named `power-block`, which is a subclass of `gdl-custom-block`. In [Applying the New Class Definition](#), you added the custom block directly to the new palette without editing the procedure.

**To display the block evaluator of an existing custom block class:**

- 1 Select Wizards > Custom Class Wizard to display the Custom Class Wizard Activity Selector dialog.



- 2 Select **edit evaluator** as the activity.

The wizard automatically displays all existing custom subclasses in the list of available classes.

---

**Tip** If you have created many custom classes, you can narrow the scope of the list of available classes to edit by selecting the desired category of classes whose subclasses you want to display.

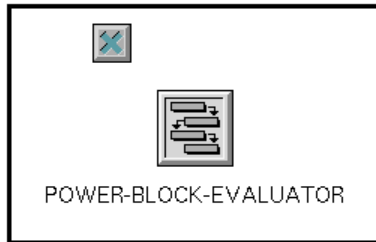
---

- 3 Select the existing custom block class whose evaluator you want to edit.

To edit the evaluator for the Power Block example, select the **power-block** class.

- 4 Click OK to edit the evaluator.

The Custom Class Wizard removes the dialog and displays a workspace that contains the default custom block evaluator:



The procedure that appears is a G2 procedure object whose name appends “-evaluator” to the associated custom block class name.

---

**Note** If you want to change the name of the block evaluator, you must also change the name of the Evaluator attribute in the table of the class definition to correspond with the new evaluator name.

---

**To display the block evaluator of a new custom block class:**

- ➔ Follow each set of steps under the headings in [Creating a New Custom Subclass](#), except that after you click Finished in step 6, click Yes to edit the block evaluator immediately.

## Types of Custom Block Evaluators

There are three basic types of block evaluators, depending on the type of custom block. The following table outlines how each block evaluator processes its input path values, and how this affects the block’s evaluation:

The block evaluator for a...	Declares input path values using...	Which means...
General custom block	Separate local names that the block evaluator refers to by name	The block evaluator identifies the inputs individually
Peer Input custom block	A single local name for all paths connected at the input of the block, and looping over the inputs	The block evaluator treats all inputs equally
Multiple Invocations custom block	The same procedure as the General custom block	The block evaluator identifies the inputs individually



For a detailed description of each block evaluator, see [General](#), [Peer Input](#), [Multiple Invocations](#), and [Single Source Encapsulation](#).

---

**Note** The default block evaluators provide you with the basic structure for obtaining data from the input path(s) and placing data onto the output path(s). It is up to you to customize how the block processes the data. You are free to edit any of the generated code, as well, if it does not meet your needs.

---

## Editing the Block Evaluator Procedure

You use the G2 text editor to edit the custom portion of the block evaluator procedure. You use the named input and output ports in the procedure.

### To edit the block evaluator:

- 1 Click on the procedure, and choose **edit** from the menu.  
The default G2 block evaluator for the particular type of custom block appears in the G2 Text Editor.
- 2 Edit the portion of the code below the comment `{** Your code goes here **}`.
- 3 Verify that the procedure you created is valid by checking the **Notes** attribute in the procedure's table. Make any necessary corrections.

In the Power Block example, the following figure shows the block evaluator. The block raises its input value by the value of the attribute **exponent**.

The figure labels each portion of the block evaluator. The table following the figure provides a brief explanation of each portion of the block evaluator. The following headings provide more detailed explanations.

```

A — power-block-evaluator(blk: class power-block, mode: symbol, client: class
    ui-client-item)
B   out-quality: symbol = mode; {the output quality} out-expiration: value = the
    symbol none; {the output expiration time} out-collection:float = 0.0; {the
    output collection-time}
    base-value:value; base-collection, base-timestamp:float; base-quality:symbol;
    base-expiration: value; {for data port base}
    result-value:value; {for data port result}
begin

  {Get input from data port base}
C   base-value, base-collection, base-quality, base-expiration, base-
    timestamp = call gdl-get-data-path-value(blk,the symbol base);
D   out-quality = resolve-gdl-quality(base-quality,out-quality);
    out-expiration = resolve-gdl-expiration(base-expiration,out-expiration);
    out-collection = max(base-collection,out-collection);

    {** Your code goes here **}
E   result-value = exp(the exponent of blk * ln...(base-value));

  {Set value for output data port result}
F   call gdl-propagate-data-path-value(blk,the symbol result, result-value, out-
    collection, out-quality, out-expiration);
end

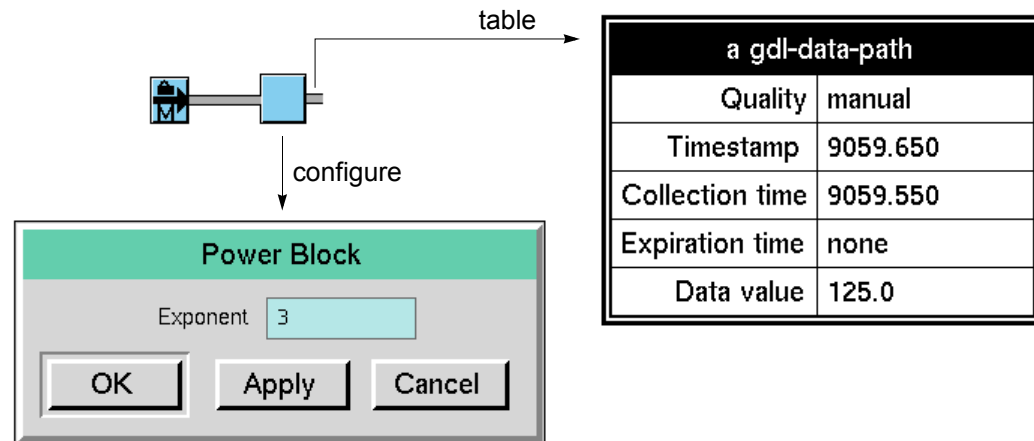
```

**The  
portion  
labeled...**

<b>Does the following...</b>	<b>For more information, see...</b>
A Declares the procedure name and arguments	<a href="#">Declaring the Procedure Name and Arguments</a>
B Declares the input and output local names	<a href="#">Declaring Input and Output Path Local Names</a>
C Obtains the input path values	<a href="#">Obtaining Input Path Values</a>
D Determines the output path attribute values	<a href="#">Determining Output Path Attributes for Custom Blocks</a>
E Customizes the output path value	<a href="#">Editing the Custom Portion of the Block Evaluator</a>
F Sets the output path values	<a href="#">Setting Output Path Values</a>

## Example Using the power-block Custom Block

The following figure illustrates how you use the power-block custom block to raise the input value of a numeric entry point to the third power. The block's configuration dialog shows the value of the Exponent attribute as 3. The data input value is 5. The output path's table shows the output Data-value, which is 125.



## Declaring the Procedure Name and Arguments

The block evaluator's procedure name is the name of the block with the word “-evaluator” appended to it, for example, power-block-evaluator.

The following table describes the procedure arguments. You use these arguments within the body of the evaluator as arguments to other procedures.

The argument...	Declares...
blk: class <i>block-class</i>	The current block being evaluated, where <i>block-class</i> is the subclass name, e.g., power-block
mode: symbol	Whether the inputs are being supplied manually using override mode ( <b>manual</b> ) or from the data-driven evaluation of the block ( <b>automatic</b> )
client: class <i>ui-client-item</i>	The client in which the block is evaluating, e.g., g2-window, if available, or gfr-default-window

## Declaring Input and Output Path Local Names

In general, the block evaluator reads values from the input path(s), performs a calculation using the inputs, and sets the resulting value(s) on the output path(s).

To do this, the block evaluator automatically declares local names for the input and output path values that the block defines. The format of the local names depend on:

- **The type of custom block.** For General and Multiple Invocations custom blocks, the block evaluator specifies separate local names for each input path, which appends “-value” to the port name. For Peer Input custom blocks, the block evaluator creates a single local name for each class of input path.
- **The type of connection path.** For data paths, the port names refer to the data values; for inference paths, the port names refer to the status and belief values; for item paths, the port names refer to the item; there are no local names for control paths.

For example, a General custom block with two data input ports named `input-data-path-1` and `input-data-path-2`, and two data output ports named `output-data-path-1` and `output-data-path-2` have the following local names:

`input-data-path-1-value`, `input-data-path-2-value`  
`output-data-path-1-value`, `output-data-path-2-value`

Compare this with the local names for a Peer Input custom block with two data input ports and two data output ports with the same names. Notice that there is only a single local name for the input ports, regardless of their names or number.

`in-value`  
`output-data-path-1-value`, `output-data-path-2-value`

### Local Names for the General and Multiple Invocations Custom Blocks

The following table summarizes the default local names for General and Multiple Invocations custom blocks that correspond to each type of input and output connection path. The table shows the local names associated with the first input and output ports created. Subsequent port names of the same type append numbers sequentially. Note that if you have renamed the input or output ports for the block, the local name includes the port name you specified, rather than the default port name.

For...	The input local names have the form...	The output local names have the form...
Data paths	<code>input-data-path-1-value</code>	<code>output-data-path-1-value</code>
Inference paths	<code>input-inference-path-1-status</code> <code>input-inference-path-1-belief</code>	<code>output-inference-path-1-status</code> <code>output-inference-path-1-belief</code>

For...	The input local names have the form...	The output local names have the form...
Item paths	input-item-path-1-item	output-item-path-1-item
Control paths	N/A	The user needs to manually encode the local variables and processing for a control output path.

### Local Names for the Peer Input Custom Blocks

The following table summarizes the default local names for Peer Input custom blocks that correspond to each type of input and output connection path. The table shows the local names associated with the first output port created. Subsequent output port names of the same type append numbers sequentially.

For...	The input local names have the form...	The output local names have the form...
Data paths	in-value	output-data-path-1-value
Inference paths	in-status in-belief	output-inference-path-1-status output-inference-path-1-belief
Item paths	in-item	output-item-path-1-item
Control paths	N/A	N/A

In addition, the local name declaration portion of the custom block evaluator declares local names for the input and output path attributes, such as **Quality** and **Expiration-time**. The local name declarations also initialize values for these path attributes.

### Obtaining Input Path Values

The first part of the body of the block evaluator gets values from input paths. To do this, the block evaluator uses one of the following API (Application Programmer's Interface) procedures that obtains data from a path. For more information, see the *GDA API Reference*.

The API procedure...	Retrieves data from...
gdl-get-data-path-value	Any data path
gdl-get-inference-path-value	Any inference path

The API procedure...	Retrieves data from...
gdl-get-control-path-value	Any control path
gdl-get-resident-path-item	Any item path and leaves the item on the path
gdl-get-path-item	Any item path and removes the item from the path

The API procedures return all relevant input path values: **Data-value** for data paths, **Status-value** and **Belief-value** for inference paths, **Quality**, **Expiration-time**, **Collection-time**, and **Timestamp** for data and inference paths, and **Quality** and **Timestamp** from a control path.

---

**Note** Typically, these API procedures obtain data from an input path; however, you can also use them to obtain data from an output path as necessary.

---

**Caution** Always use an API procedure to obtain data from a path. Do not refer to path attributes by name.

---

## Determining Output Path Attributes for Custom Blocks

The body of the block evaluator must resolve the output path **Quality**, **Expiration-time**, and **Collection-time** for custom blocks with multiple inputs using the following two API procedures. Note that the block evaluator does not determine the **Timestamp** attribute. These procedures are described in detail in the *GDA API Reference Manual*.

The API function...	Resolves...
resolve-gdl-quality	The <b>Quality</b> attribute of an output path using the lowest quality of its two quality arguments. For a description of the quality hierarchy, see <a href="#">The Quality Attribute</a> .
resolve-gdl-expiration	The <b>Expiration-time</b> attribute of an output path using the earliest of its two expiration time arguments.

The block evaluator resolves the output **Collection-time** using the maximum collection time of the input collection times.

## Editing the Custom Portion of the Block Evaluator

The custom portion of the block evaluator follows the comment `{** Your code goes here **}`. This portion of the block evaluator defines the output path value for the block, which is declared using a local name.

For the `power`-block, the value of the output path named `result` is the exponent (`exp`) of the product of the `exponent` attribute defined by the block and the logarithm (`ln`) of the value of the input path named `base`, which raises the base value to the `exponent` power.

```
result-value = exp(the exponent of blk * ln(base-value))
```

For information on how to specify a procedure, see the *G2 Reference Manual*.

## Setting Output Path Values

The last part of the block evaluator sets values onto output paths. To do this, the block evaluator uses one of the following API procedures that set a value onto an output path and causes the evaluation of downstream blocks. For more information on these procedures, as well as procedures that do not cause downstream evaluation, see the *GDA API Reference Manual*.

The API procedure...	Sets data onto...
<code>gdl-propagate-data-path-value</code>	A data path
<code>gdl-propagate-inference-path-value</code>	An inference path
<code>gdl-propagate-control-path-value</code>	A control path
<code>gdl-propagate-resident-path-item</code>	An item path and removes, but does not delete, the other item on the path
<code>gdl-propagate-path-item</code>	An output item path and deletes the other item on the path

---

**Caution** Within an evaluator, do not set a block's inputs, otherwise an infinite loop may occur.

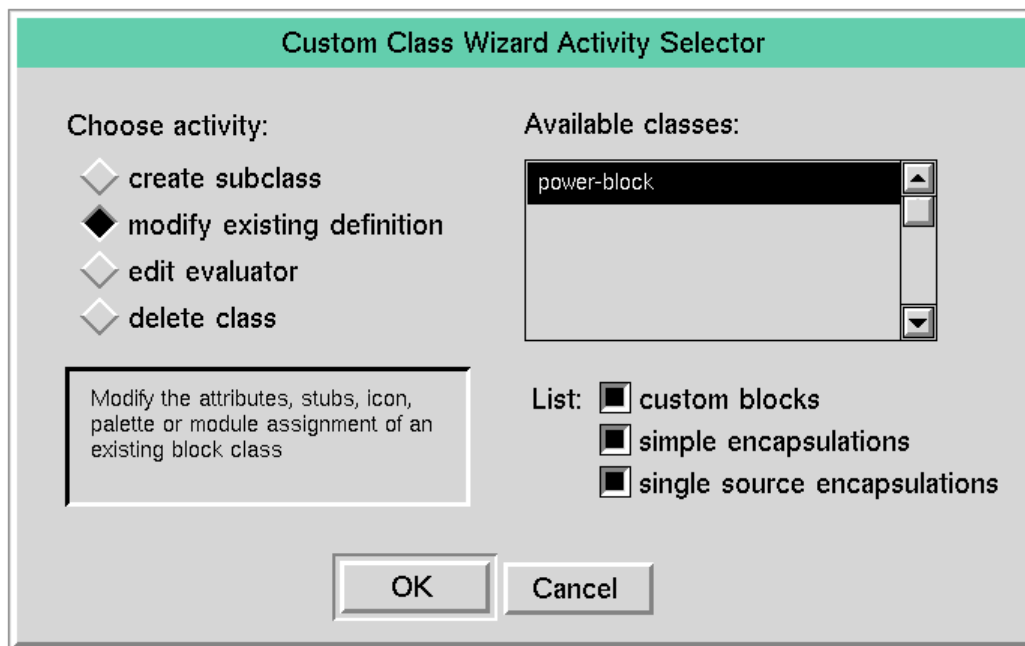
---

## Editing an Existing Custom Subclass

Once you have created a new custom subclass, you can edit the definition of the subclass: its icon, its ports, its attributes, and its evaluator. When you do this, GDA creates a new custom block evaluator and backs up the old evaluator to a different name.

### To edit the class definition of an existing custom subclass:

- 1 Choose Wizards > Custom Class Wizard from the top menu bar.
- 2 Choose modify existing definition under Choose activity. GDA displays all the existing custom subclasses that you can edit. For example:



- 3 To narrow the scope of the list of existing subclasses, under List, select only the category of available custom subclasses that you want to edit.

By default, all the available categories are selected. Click on a selected check box to toggle the selection off and on. GDA automatically updates the list of existing subclasses to correspond with the selection.

- 4 Click OK to accept the selections.
- 5 Follow the Custom Class Wizard's prompts to edit the block.



---

**Note** GDA creates a new block evaluator based on the new custom class and renames the existing evaluator to *old-classname-evaluator*. If you have customized the block evaluator of the existing class, you can copy code from the old evaluator into the new evaluator.

---

The prompts are the same as when creating a new custom subclass, as explained in [Creating a New Custom Subclass](#).

**To edit the evaluator of an existing custom subclass:**

---

**Note** If you try to edit the evaluator of a custom class created before GDA Version 3.0, you get an error message that the procedure has been moved. You should first edit the old custom class with the Wizard, which will put the evaluator procedure in the proper place. You can then edit the evaluator.

---

- 1 Choose Wizards > Custom Class Wizard from the top menu bar.
- 2 Choose edit evaluator under Choose activity.  
GDA displays all the existing custom subclasses whose evaluators you can edit.
- 3 To narrow the scope of the list of existing subclasses, under List, select only the category of available custom subclasses that you want to edit.  
By default, all the available categories are selected. Click on a selected check box to toggle the selection off and on. GDA automatically updates the list of existing subclasses to correspond with the selection.
- 4 Click OK to accept the selections. GDA displays the block evaluator for the custom subclass.
- 5 Edit the block evaluator as desired.

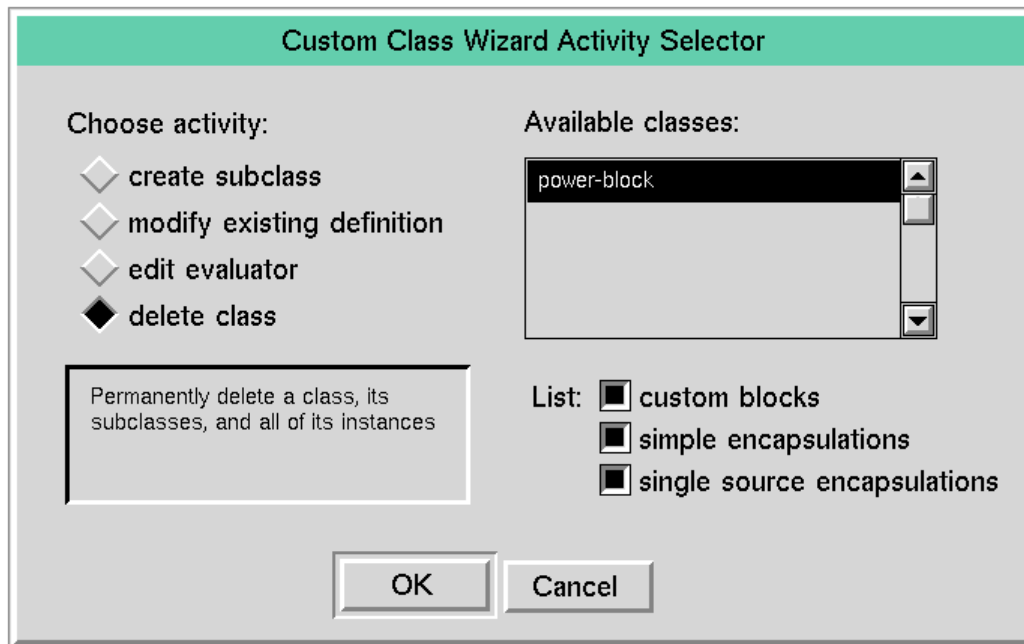
## Deleting an Existing Custom Subclass

You can delete a custom subclass and remove it from the custom palette. As a result of deleting a subclass, you might want to reconfigure the layout of the custom palette or delete the palette.

**To delete an existing custom subclass:**

- 1 Choose Wizards > Custom Class Wizard from the top menu bar.
- 2 Choose delete class under Choose activity.

GDA displays all the existing custom subclasses that you can delete. For example:



- 3 To narrow the scope of the list of existing subclasses, under List, select only the category of available custom subclasses that you want to delete.

By default, all the available categories are selected. Click on a selected check box to toggle the selection off and on. GDA automatically updates the list of existing subclasses to correspond with the selection.

- 4 Click OK to delete the subclass.
- 5 Click Yes in the confirmation dialog to delete the custom subclass.

---

**Note** The wizard deletes the custom subclass itself, any subclasses of the custom class, as well as all instances of the custom subclass, including the instance on the custom palette.

---

If you delete a class, you can reconfigure the layout of the custom palette. For example, if you deleted the first of several blocks on a palette, you might want to move the existing custom blocks up on the palette.

**To reconfigure the layout of the custom palette after deleting a class:**

- ➔ Go into Administrator mode, and move the blocks and associated labels to the desired location on the palette. You can move a block by dragging it. You can move a label by using operate on area.

If you have deleted all custom blocks on a palette, you might want to delete the custom palette itself.

**To delete a custom palette:**

- ➔ Go into Administrator mode, and select the **delete palette** menu item on the palette workspace.

## Custom Class Reference

This section describes each of the GDA classes from which you can create a custom subclass. These are the classes that are available:

- General
- Peer Input
- Multiple Invocations
- Single Source Encapsulation

## General

The General custom block specifies the basic set of attributes necessary for creating a custom GDA block. The General custom block performs evaluations on single or multiple inputs when the block needs to reference specific inputs by port name. The Difference block on the Arithmetic palette is similar to a General custom block because the bottom input is subtracted from the top input, which requires that the block evaluator refer to each input by name.

---

**Note** To create a custom block where the block evaluator does not identify the inputs by name, use the Peer Input block, described on . To create a general-purpose custom block that handles multiple simultaneous values, use the Multiple Invocations block, described on .

---

For general information on how to specify the block and its associated procedure, see [Creating a New Custom Subclass](#) and [Customizing the Block Evaluator](#).

## Peer Input

The Peer Input custom block performs evaluations on single or multiple inputs when the block treats all inputs equally. The Summation block on the Arithmetic palette is similar to a Peer Input block because all inputs are added together without requiring that they be identified separately. A Peer Input custom block specifies additional attributes that determine which inputs the block uses in its calculation based on the input quality.

For general information on peer input blocks, see [Connecting to Peer Input Blocks](#).

---

**Note** To create a custom block where the block evaluator identifies each input by its port name, use the General block, described on [General Block](#), or the Multiple Invocations block, described on [Multiple Invocations Block](#).

---

### Creating Connections for Custom Peer Input Blocks

You use the Custom Class Wizard to customize the connection ports of subclasses of the `gdl-custom-peer-input-block` class. When adding input ports to a Peer Input custom block, notice that, by default, the wizard does not name the port. This is because a Peer Input custom block treats all inputs equally, and the block evaluator loops over all the inputs to determine their values. The wizard does, however, name all output ports of a Peer Input custom block.

---

**Tip** Typically, you create a single input port on a Peer Input custom block, and you drag additional input paths into an instance of the block on a workspace to create additional inputs. When creating the subclass, however, always create at least one input port to generate the correct block evaluator.

---

### Customizing the Peer Input Custom Block Evaluator

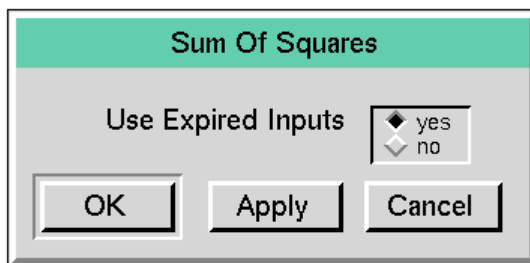
The block evaluator for a Peer Input custom block with input data paths defines a single local name for all input paths connected to the block, and loops over the inputs to obtain their values. The block evaluator for a Peer Input custom block with input inference paths defines a single local name for the status value and a single local name for the belief value of all input paths connected to the block. This is in contrast to a General custom block, which defines separate local names for each input connection whose values are obtained independently. See the next example for details.

Because of the way the default Peer Input block evaluator is designed, several things follow:

- The block evaluator for a Peer Input custom block cannot control the order of evaluation of the inputs.
- The custom portion of the block evaluator cannot differentiate among individual input path values for paths of the same type, e.g., data paths.
- You can drag input paths into the block on a workspace after you customize the subclass.
- You do not need to name input paths when adding connections to the block using Stub Tools or dragging connections into the block on a workspace.

## Configuring

This is the configuration panel for the subclass of a Peer Input custom block created in the example below:



Attribute	Description
Use Expired Inputs	For information on this attribute, see <a href="#">Determining Whether a Block Uses Expired Inputs</a> .

## Example

Suppose you want to create a custom block that calculates the sum of the squares of its inputs. To do this, use the Custom Class Wizard to create a Peer Input custom block that sets the output path value to the square of its input values. The default block evaluator loops over all of the inputs, which processes all of the input paths, regardless of their names or number.

The following figure shows the Peer Input custom block, its evaluator, and its table. Notice that the block only has one input path by default. You can edit the value of the **Use-expired-inputs** attribute in the table.



SUM-OF-SQUARES-EVALUATOR

a sum-of-squares	
Notes	SUM-OF-SQUARES-XXX-183: OK
Error	""
Comments	none
Use expired inputs	yes

The figure below shows the block evaluator for this custom class, with important parts labeled. The table that follows the figure provides a brief explanation of each labeled portion of the block evaluator.

```

sum-of-squares-evaluator(blk: class sum-of-squares, mode: symbol, client:
  class ui-client-item)
  use-exps: symbol = the use-expired-inputs of BLK;
  out-quality: symbol = mode; {the output quality}
  out-expiration: value = the symbol none; {the output expiration time}
  out-collection: float = 0.0; {the output collection-time}
  in-timestamp, in-collection: float;
  in-quality: symbol;
  in-expiration: value;
  p: class gdl-path;
  in-value: value; {for data inputs} ————— A
  output-data-path-1-value: value; {for data port output-data-path-1}
  begin
    output-data-path-1-value = 0; {initial value} ————— B
    for p = each gdl-data-path connected at an input of blk do ————— C
      in-value, in-collection, in-quality, in-expiration, in-timestamp = call gdl-
        get-data-path-value(blk,p);
      if in-quality /= the symbol no-value and (use-exps = the symbol yes — D
        or in-quality /= the symbol expired) then begin
        out-quality = resolve-gdl-quality(in-quality,out-quality);
        out-expiration = resolve-gdl-expiration(in-expiration,out-expiration);
        out-collection = max(in-collection,out-collection);

        {***Your code goes here***}
        output-data-path-1-value = output-data-path-1-value + (in-value * in-
          value); ————— E
        end;
      end;
    end;

  {Now set outputs}

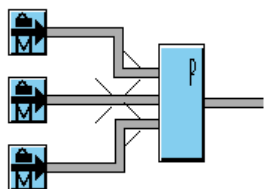
  {Set value for output data port output-data-path-1}
  call gdl-propagate-data-path-value(blk,the symbol output-data-path-1,
    output-data-path-1-value, out-collection, out-quality, out-expiration);
  end

```



The portion labeled...	Does the following...
A	Declares a single local name for all input paths
B	Initializes the output path
C	“For” loop, which loops over input paths to obtain their value
D	Determines output path quality
E	Custom portion of procedure, which sets the output value to the sum of the squares of each input value

The following figure illustrates how you use this block to sum the squares of a custom block with three input paths. You drag the input paths into the block on a workspace after you create the custom block. The input values are 2, 3, and 4, and the output path Data-value is 29.



a gdl-data-path	
Quality	manual
Timestamp	1428.000
Collection time	1427.900
Expiration time	none
Data value	29

# Multiple Invocations

The Multiple Invocations custom block processes simultaneous signals. The block processes its inputs the same way as a General custom block. The Data Delay block on the Data Control palette is similar to a Multiple Invocations block. A Multiple Invocations custom block specifies additional attributes that determine how the block handles simultaneous signals.

For information on the options available for the Multiple Invocations attribute of a block that handles multiple signals, see [Specifying How to Handle Multiple Values](#).

---

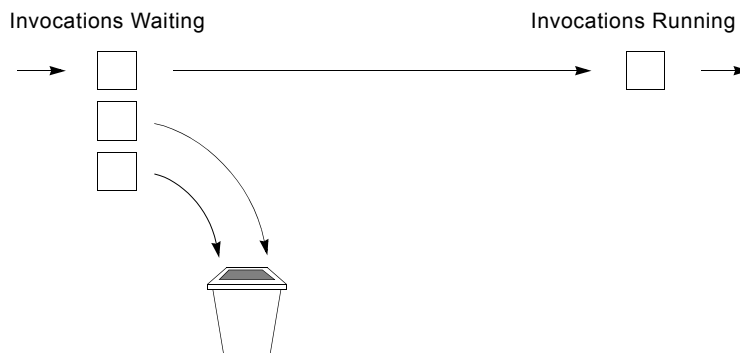
**Note** To create a custom block that does not process simultaneous signals, use the General custom block or the Peer Input custom block.

---

## Determining How the Block Handles Multiple Control Signals

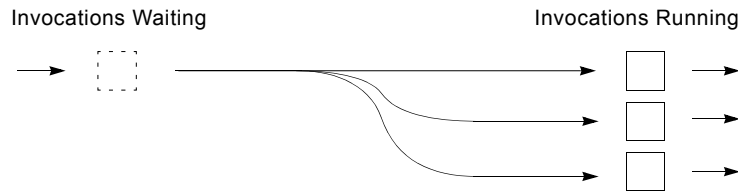
The Multiple Invocations custom block contains two configurable attributes that determine what happens when the block receives multiple values: Multiple Invocations and Asynchronous Evaluation. The block also has two read-only attributes in its table that indicate the current status of the multiple invocations of the block: Invocations-waiting and Invocations-running.

The default value for Multiple Invocations is `ignore`, and the default value for Asynchronous Evaluation is `false`. The result is that the block processes single values one time and ignores extra values, as shown in the following figure. The boxes indicate values or signals received and processed by the block. The line at the left of the diagram indicates an incoming value to the block, and the line to the right of the diagram illustrates outgoing values.

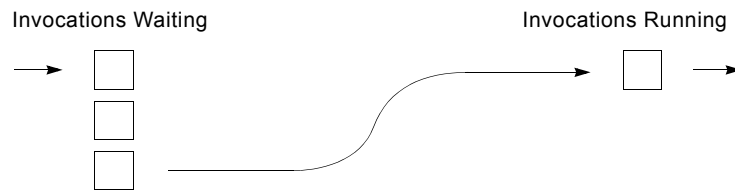


When Multiple Invocations is `ok`, Asynchronous Evaluation should be `true` in order to allow multiple processing. In the following diagram, the dashed item under the heading Invocations Waiting indicates values that are automatically

advanced to the Invocations Running category without waiting. The items under the heading Invocations Running indicate values currently being processed.



When Multiple Invocations is `queue`, the block only processes a single value at a time. If `Asynchronous Evaluation` is `true`, the following block evaluates immediately after the current block is advanced to the Invocations Running. If `Asynchronous Evaluation` is `false`, GDA waits until the current block is finished evaluating before evaluating the next block. In the following diagram, the block receives a value, which goes into the queue. The block processes the oldest value in the queue first.




---

**Note** If the block takes a long time to evaluate, you may want to set `Asynchronous Evaluation` to `true` to allow other processing in the overall GDA diagram while the block evaluates.

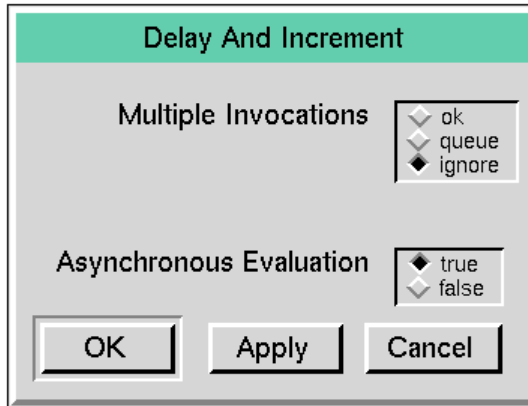
---

## Customizing the Multiple Invocations Custom Block Evaluator

The Multiple Invocations custom block evaluator is identical to the General custom block evaluator. You customize the block evaluator as you would for any custom block.

## Configuring

This is the configuration dialog for the subclass of a Multiple Invocations custom block created in the example below:



Attribute	Description
Multiple Invocations	Specifies whether the block ignores ( <b>ignore</b> ), queues ( <b>queue</b> ), or processes ( <b>ok</b> ) simultaneous signals when the block is already evaluating
Asynchronous Evaluation	Specifies whether GDA waits for the evaluation of the block to finish before beginning the next evaluation ( <b>false</b> ), or begins the evaluation of the next block immediately ( <b>true</b> )

A subclass of a Multiple Invocations custom block also defines these two read-only attributes in its table:

The attribute...	Is a...
Invocations Waiting	Read-only attribute that indicates the number of signals in the queue when Multiple Invocations is <b>queue</b>
Invocations Running	Read-only attribute that indicates the number of simultaneous evaluations when Multiple Invocations is <b>ok</b>

For more information about these attributes, see [Specifying How to Handle Multiple Values](#).

## Example

Suppose you want to create a block that delays processing its input signal by three seconds and adds one to the delayed input. To do this, create a Multiple Invocations custom block with a single input data path and a single output data path. To allow processing of multiple simultaneous inputs, specify Multiple Invocations as `ok`. To allow other processing during the three second delay, specify `Asynchronous Evaluation` as `true`.

The following figure shows the block, the evaluator, and the associated table:



DELAY-AND-INCREMENT-EVALUATOR

a delay-and-increment	
Notes	DELAY-AND-INCREMENT-XXX-249: OK
Error	""
Comments	none
Invocations waiting	0
Invocations running	0

The following figure shows the Multiple Invocations custom block evaluator, which labels the custom portion of the code, which delays processing for 3 seconds, then increments the output value:

```

_delay-and-increment-evaluator(blk: class delay-and-increment, mode: symbol,
 client: class ui-client-item)
  out-quality: symbol = mode; {the output quality} out-expiration: value = the
    symbol none; {the output expiration time} out-collection:float = 0.0; {the
    output collection-time}
  input-data-path-1-value:value; input-data-path-1-collection, input-data-path-1-
    timestamp:float; input-data-path-1-quality:symbol; input-data-path-1-
    expiration: value; {for data port input-data-path-1}
  output-data-path-1-value:value; {for data port output-data-path-1}
  begin

  {Get input from data port input-data-path-1}
    input-data-path-1-value, input-data-path-1-collection, input-data-path-1-
      quality, input-data-path-1-expiration, input-data-path-1-timestamp = call
      gdl-get-data-path-value(blk,the symbol input-data-path-1);
    out-quality = resolve-gdl-quality(input-data-path-1-quality,out-quality);
    out-expiration = resolve-gdl-expiration(input-data-path-1-expiration,out-
      expiration);
    out-collection = max(input-data-path-1-collection,out-collection);

    {** Your code goes here **}
    wait for 3 seconds;
    output-data-path-1-value = input-data-path-value + 1;

  {Set value for output data port output-data-path-1}
    call gdl-propagate-data-path-value(blk,the symbol output-data-path-1,
      output-data-path-1-value, out-collection, out-quality, out-expiration);
  end

```

— Custom portion of procedure

The following table shows sample input and output values for this block. The input and output values update once a second. The output values increment the input value associated with  $t_{n-3}$  by one.

At time...	The input value is...	And the output value is...
0 seconds	0.0	no-value
1 second	1.0	no-value
2 seconds	2.0	no-value
3 seconds	3.0	1.0

<b>At time...</b>	<b>The input value is...</b>	<b>And the output value is...</b>
4 seconds	4.0	2.0
5 seconds	5.0	3.0
6 seconds	6.0	4.0
7 seconds	7.0	5.0
8 seconds	8.0	6.0
9 seconds	9.0	7.0

# Single Source Encapsulation

The Single Source Encapsulation (SSE) block enables you to manage complexity in a GDA model. Using SSEs enables you to:

- Create a specialized block that is associated with a diagram, in which blocks are configured with default attribute values
- Clone the SSE multiple times within a model
- Modify attribute values of blocks within a “local” copy of the SSE
- Manage the “master” copy of the SSE, propagating modifications to local copies of the SSE

Using SSEs enables you to maintain consistency among cloned diagrams encapsulated on a subworkspace.

You use the Custom Class Wizard to create a subclass of the `gdl-single-source-encapsulation` class. The new subclass has a customized icon, connections, and attributes. Each instance of a particular subclass inherits the block’s definition, including its subworkspace.

You can also create subclasses of the `gdl-simple-encapsulation` class. For more information on simple encapsulation blocks, see the *GDA Reference Manual*.

## Creating a New Subclass of SSE Block

When you create a new subclass of a Single Source Encapsulation block, you have two choices:

- Use the Custom Class Wizard to create a subclass of the `gdl-single-source-encapsulation` class, and edit the subclass.
- Convert a Simple Encapsulation block to a Single Source Encapsulation block by selecting `convert to sse` from its menu. This creates an instance of the new subclass of SSE and places it on the workspace.

---

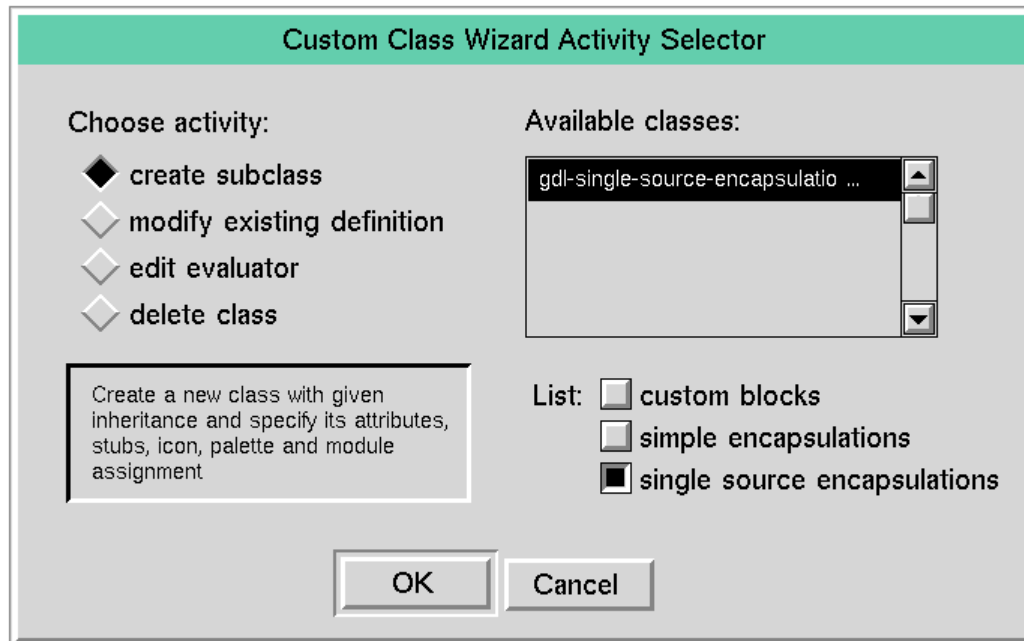
**Note** You cannot create subclasses of SSE blocks that are based on an existing custom subclass. This behavior is unlike General, Peer Input, and Multiple Invocations custom blocks, where you can create subclasses of subclasses of these three types of custom blocks.

---



**To create a new subclass of SSE block:**

- 1 On the Custom Class Wizard Activity Selector dialog, deselect the custom blocks and simple encapsulations choices in the List area, make sure the `gdl-single-source-encapsulatio` class is highlighted, then choose OK:



- 2 The next steps are the same ones used to create any custom block, described in [Creating a New Custom Subclass](#).

On the dialog that appears for the next step, enter a class name for the new SSE class, then choose Forward.

- 3 Next, use the Stub Tools to add input and output stubs to the block, then choose Forward.

---

**Note** When you create a subclass of Single Source Encapsulation block, there is no need to name the ports explicitly. Be sure to create as many input and output ports as you will need when you are creating the subclass, because you cannot interactively add paths to the block on a workspace.

---

- 4 Next, specify any attributes for the SSE block and choose Forward.
- 5 Next, specify the palette on which the custom block is to reside, specify the module, then choose Forward.

## The Master Diagram and the Local Diagram

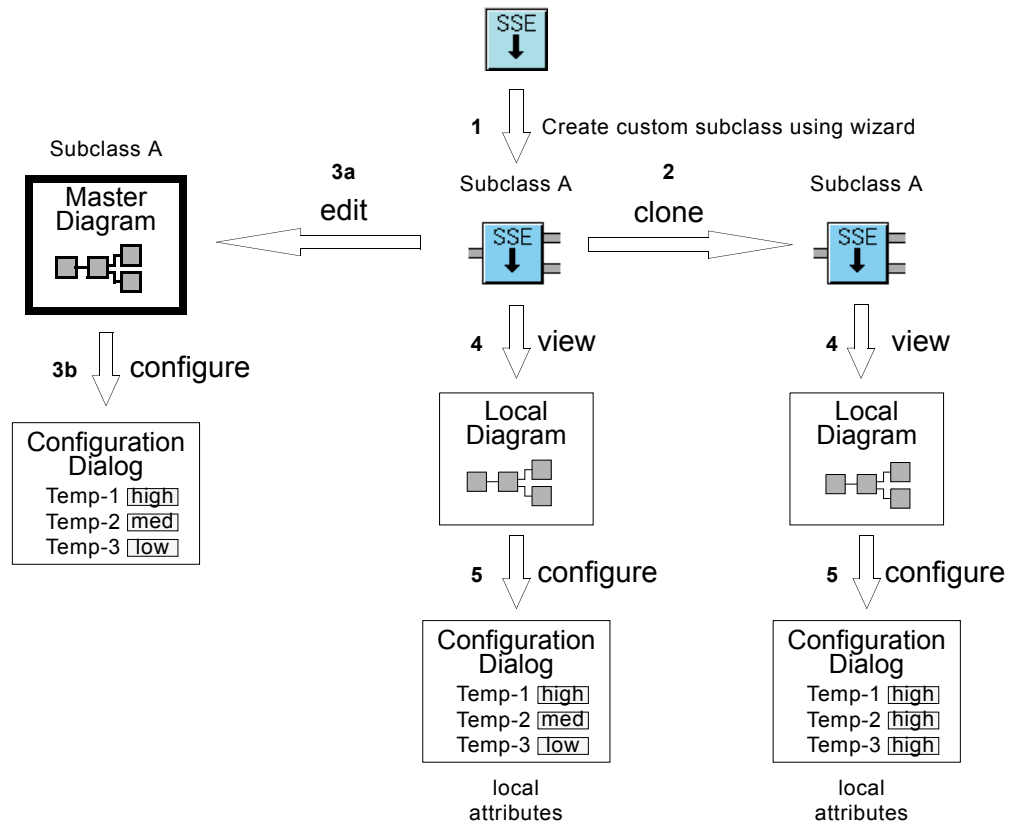
An SSE is associated with two diagrams:

- A **Master Diagram**, which defines the encapsulation for all instances of the SSE. The master diagram is the workspace on which the blocks that make up the SSE reside.
- A **Local Diagram**, which defines the local version of the SSE. You can configure blocks in the local diagram differently than they are configured in the Master Diagram.

You interact with these diagrams in the following ways:

- You create the SSE and define its structure by creating its Master Diagram. You configure blocks in the diagram to provide their default values.
- You can modify the structure of the Master Diagram, which updates the structure of all Local Diagrams of the same subclass; local attribute values remain unchanged. See [Editing the Master Diagram](#) for details.
- You can modify the local attributes of a particular instance on the Local Diagram without affecting the local attributes of other instances of the same subclass. See [Viewing the Local Diagram](#) for details.
- You can modify attribute values for all instances of a particular subclass using the **broadcast** facility, which overwrites individual local attribute values. See [Changing the Value of an Attribute](#) for details.

The following diagram illustrates some of the ways of interacting with this block:



- 1 Create Subclass A, which is a subclass of the `gdl-single-source-encapsulation` class, by using the Custom Class Wizard.
- 2 Create an instance of subclass A by cloning the customized block from the custom palette.
- 3 Edit the Master Diagram of subclass A (3a), configure the default attributes for the blocks (3b), and save the diagram to update the Local Diagrams of all the instances of subclass A.
- 4 View the Local Diagram of any instance.
- 5 Configure the local attributes of any blocks within the diagram of any instance, as needed.

## Editing the Master Diagram

After creating a new subclass of Single Source Encapsulation block, edit the Master Diagram to:

- Create the initial encapsulation diagram.
- Edit an existing diagram and propagate the changes to other blocks of the same subclass.

### Displaying the Master Diagram

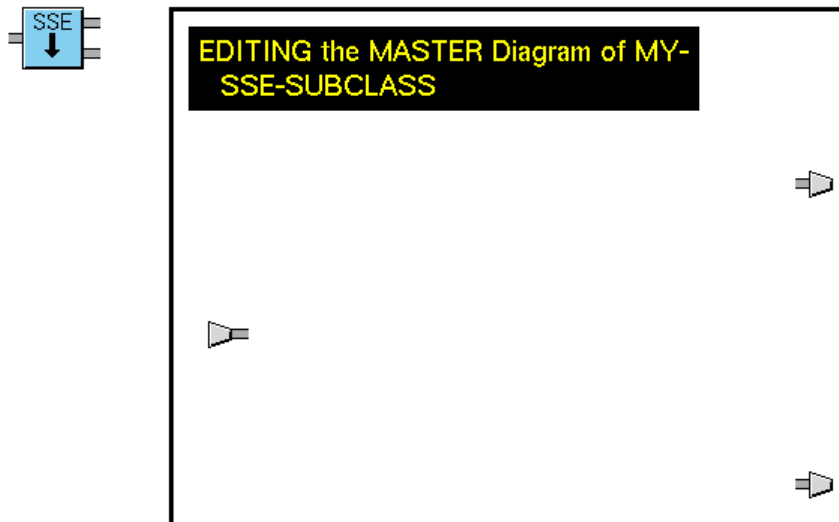
**To display the Master Diagram:**

- ➔ Select edit master diagram from the menu of an instance of a Single Source Encapsulation block.

When you create a new subclass, the Master Diagram is initially empty, except for the connection posts associated with the input and output ports defined for the block.

The input(s) to the Single Source Encapsulation block is passed to the block(s) connected to the input connection post(s) on the diagram. This value is then passed through all the blocks in the diagram until it reaches the output connection post(s). The output(s) of the Single Source Encapsulation block is the value(s) that is passed to the output connection post(s) on the diagram.

The following figure illustrates the Master Diagram and associated Single Source Encapsulation block with one data input port and two data output ports:



Note that the Master Diagram of a Single Source Encapsulation block does not contain a close button or a go to superior workspace button like the simple Encapsulation block. Instead, you must close the diagram using **save diagram** or **cancel diagram editing**, as described in [Saving the Master Diagram](#).

---

**Note** You cannot edit the Master Diagram from two instances of the same class simultaneously.

---

### Editing the Blocks on the Master Diagram

You edit the Master Diagram by adding, deleting, or moving blocks or connections. Editing the structure of the Master Diagram has these restrictions:

- You configure the attributes of blocks on the Master Diagram to define their default values; you configure blocks on the Local Diagram to provide values for each instance.
- You cannot create connection posts manually on the Master Diagram; connection posts on the Master Diagram must connect to the superior workspace. To change the number of connection posts on the Master Diagram, you must modify the existing definition of the subclass.
- You cannot place G2 objects on the Master Diagram except rules and free text.
- You cannot create scanned rules on an SSE, either by specifying a scan interval for the rule. Rules are only invoked when the block evaluates.
- To include a subclass of G2 variable on an SSE, you must add an attribute named `sse-id` to the variable's definition.
- You cannot change the number of states of a Multi-state block contained on a Local Diagram, only on the Master Diagram.
- You cannot evaluate blocks on the Master Diagram.
- When configuring a Chart Capability on the Master Diagram, you should not specify the Name of Chart attribute; GDA does not preserve the value. You should specify this attribute on the Local Diagram instead.
- When configuring a Numeric Entry Point, do not configure the Name of Sensor attribute.

### Using Rule Terminals on Single Source Encapsulation Blocks

You include Rule Terminals on a Single Source Encapsulation block to execute rules when the block evaluates. When an instance of an SSE block receives a value, the block invokes any rule located on its Master Diagram that refers to that value.

For more information on including rules on an SSE, as well as some important restrictions when using rules on SSEs, see the *GDA Reference Manual*.

## Saving the Master Diagram

When you are done editing the Master Diagram, save the diagram, cancel editing the diagram, or save the Master Diagram locally.

### Saving the Master Diagram

**To save the diagram:**

- Choose **save diagram** from the Master Diagram's workspace menu.
- or
- Click on the title bar of the Master Diagram, then choose **save diagram** from the menu that appears.

Saving the diagram updates all existing instances of the same subclass to use the new structure. Depending on the number of existing instances, GDA displays a clock in the upper-left corner of the window. Saving the Master Diagram also closes the diagram.

Saving the Master Diagram does not replace local attribute values for previously defined instances of the subclass. To change local attribute values, use **broadcast attribute** on the Local Diagram for any block in the encapsulation, as explained in [Changing the Value of an Attribute](#). You can also delete the block from the Master Diagram, then clone the block and define its default attribute values.

---

**Note** When saving the Master Diagram, GDA does not always preserve object names.

---

---

**Note** Be sure to save or close the Master Diagram of an SSE before saving the KB; otherwise, errors may occur when you reload the KB.

---

### Cancelling Editing the Master Diagram

**To cancel editing the diagram without propagating the changes:**

- Choose **cancel diagram editing** from the Master Diagram menu, which also closes the diagram.
- or
- Click on the title bar, then choose **cancel diagram editing** from the menu that appears.

## Saving the Master Diagram Locally

If you want to test the encapsulation block locally before applying the changes to all other instances, save the Master Diagram locally.

### To save locally:

- ➔ Choose apply changes locally.
- or
- ➔ Click on the title bar, then choose apply changes locally from the menu that appears.

Saving the diagram locally saves the changes to the current instance. GDA displays the Local Diagram for the instance and leaves the Master Diagram open. Do not forget to edit the Master Diagram again and choose **save diagram** when you are sure the changes are valid.

## Creating Instances of a Single Source Encapsulation

To create multiple instances of a Single Source Encapsulation block, clone the SSE after customizing and saving its Master Diagram. Each instance of the subclass contains the same diagram and attribute values as the subclass.

You customize local attribute values for individual instances as described in [Changing the Value of an Attribute](#).

You can modify attribute values on the Master Diagram and cause all instances of the SSE to have the same attribute values by broadcasting each attribute value, as described in [Changing the Value of an Attribute](#).

## Viewing the Local Diagram

Each instance of a Single Source Encapsulation block contains a Local Diagram, whose structure is identical to that of the Master Diagram. You use the Local Diagram for:

- Viewing the details of the encapsulation without editing the structure; you cannot add, delete, or move any blocks or connections on the Local Diagram.
- Configuring the attributes of blocks for individual instances.

### To display the Local Diagram:

- ➔ Choose view local diagram from the Single Source Encapsulation block's menu.

The title indicates that you are editing the local diagram.

**To hide the local diagram:**

➔ Click on the title bar of the diagram.

or

➔ Choose hide diagram from the Local Diagram's menu.

## **Changing the Value of an Attribute**

You change the value of attributes defined by blocks on the Local Diagram of a Single Source Encapsulation block in one of two ways:

- Locally, for a single instance
- Globally, for all instances of a subclass

**To change attribute values locally:**

- 1 Display the Local Diagram for a particular instance by using the **view local diagram** menu choice.
- 2 Edit the configuration dialog of any block.
- 3 Remove the diagram using **hide diagram**.

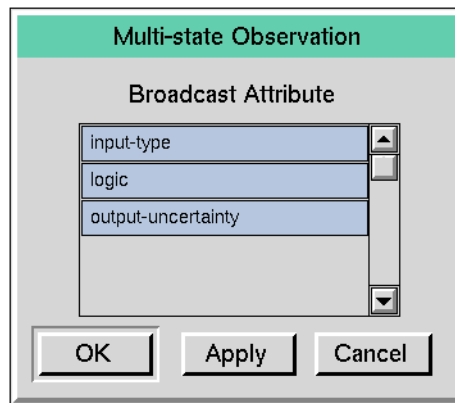
Any subsequent changes to the Master Diagram or the Local Diagram of any other instance of the same subclass do not affect the locally defined attribute values. However, locally defined attributes can be overridden by broadcasting attribute changes; see the next sequence of steps.

**To change attribute values globally:**

- 1 Display the Local Diagram for a particular instance by using the **view local diagram** menu choice.
- 2 Edit the configuration dialog of any block.
- 3 Choose **broadcast attribute** from the block's menu to display the Broadcast Attribute dialog.



The following figure shows this dialog for a Multi-state block:



- 4 Select the name of a single attribute whose new value is to be broadcast.
- 5 Click Apply.
- 6 Repeat for as many attributes as necessary.
- 7 Select OK when done.

Changing attributes globally explicitly overrides local attribute values for the specified attributes in all instances of the subclass.

---

**Note** Broadcasting attributes does not update the attribute values defined in the Master Diagram.

---

### Configuring Attributes in the Master Diagram

You generally only configure attributes in the Master Diagram when you initially create the subclass to provide default values, then configure attributes in the Local Diagram of an instance of a class of SSE to provide the values needed by that instance..

When you create a subclass of SSE, you can specify attributes of blocks in the Master Diagram; these become the default attribute values, which apply to all instances of the subclass. To do this, be sure to configure the attributes before you create multiple instances of the SSE by cloning; otherwise, not all instances will have the default values.

---

**Note** You cannot broadcast attribute values from the Master Diagram.

---

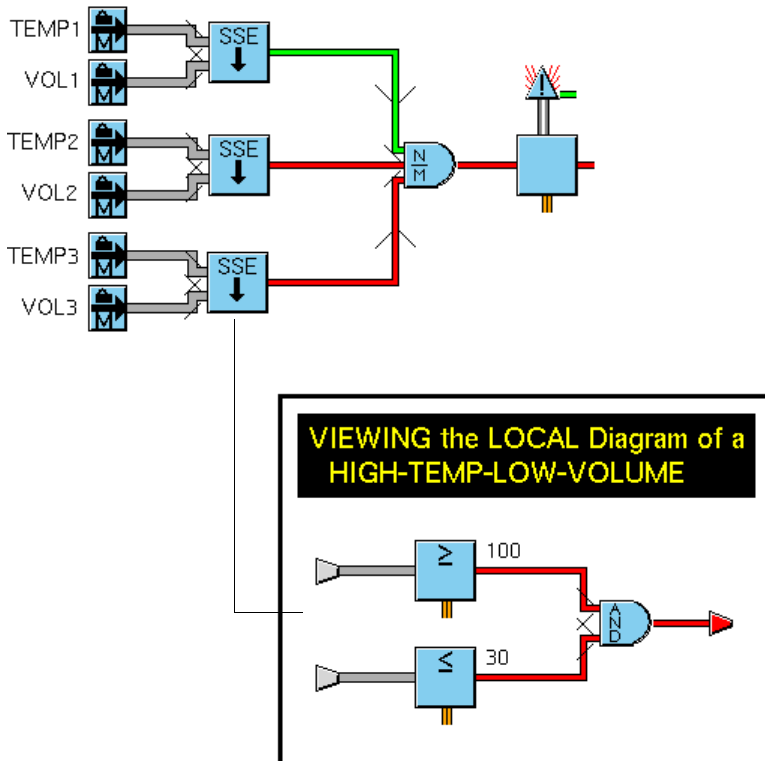
**Caution** Do not configure the Chart Name for a Chart Capability on the Master Diagram; otherwise, errors may occur. Similarly, do not configure the Sensor name for an Entry Point.

## Configuring

The Single Source Encapsulation block has no configurable attributes.

## Example

The following example illustrates a diagram that uses three Single Source Encapsulation blocks that test two input values and generate one output value. The figure shows the Local Diagram for the bottom SSE block.



# Creating and Configuring Queues

---

*Describes how to create new queues and configure their attributes to modify queue behavior and appearance.*

Introduction	217
Attributes of Queues You Can Modify	218
Creating a New Queue	219
Configuring a Queue	221
Using a New Queue	229
Logging Queue Entries	229



## Introduction

GDA support four types of queues: Alarm Queue, Error Queue, Explanation Queue, and Message Queue, and provides a built-in queue of each type, named alarm-queue, error-queue, explanation-queue, and message-queue.

You can use features of the queue facility in your applications in these ways:

- You can use the built-in queues without changes to take advantage of the features GDA provides. For more information about the features of the built-in queues, see [Attributes of Queues You Can Modify](#).
- You can modify the built-in queues to change some of these features.
- You can create new queues to change queue features or to provide multiple queues of a particular type. You can also create a new queue class and modifying an instance of the class. Creating a new queue class enables a developer to extend the behavior or queues.

This table lists the queue types, indicating the source of the entries that get sent to the queues:

<b>This queue type...</b>	<b>Posts entries from this source...</b>
Alarm Queue	Alarm Capability block Recurring Alarm Capability block
Error Queue	Errors in a GDA diagram
Explanation Queue	Explanations of output values of inference blocks
Message Queue	Queue Message block (which can send entries to all queue types)

This chapter describes how to create and specify the behavior of new queues.

For information about changing the appearance of queues, see [Creating Queue Views](#).

## Attributes of Queues You Can Modify

You can modify numerous aspects of queues. The Error Queue, Explanation Queue, and Message Queue have the same attributes. The Alarm Queue has several additional attributes.

The queue attributes and procedures you can modify fall into these categories:

- Attributes that handle new queue entries
  - Whether to display a view of the queue when an entry is generated
  - Whether to beep when a new entry is generated
  - The entry class of new entries posted to a queue
  - A procedure to execute when a new entry is posted to a queue

- Attributes that handle deletions of and changes to entries
  - Whether to confirm deletion
  - A procedure to execute when an entry is removed or deleted
  - A procedure to execute when an entry is changed
- Attributes that manage queue capacity
  - How long an entry exists
  - How many entries a queue can contain
- Attributes that manage Alarm Queues
  - Whether to log Alarm Queue entries and the format of those entries
  - How many entries to maintain in the history
  - Whether to reuse Alarm Queue entries or generate a new entry at each new occurrence of an alarm condition
- Other attributes
  - Whether to display the traceback for errors and alarms

## Creating a New Queue

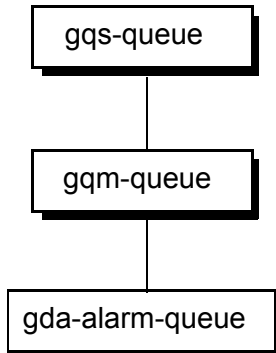
Creating a new queue enables your application to make use of multiple queues of a particular queue type. For example, you might want to use different alarm queues to receive alarms for different parts of an application.

To create a custom queue, you can create a new queue instance from a built-in queue. The new queue is an instance of the same class as the built-in queue.

The built-in queues are instances of these classes and exist in these modules:

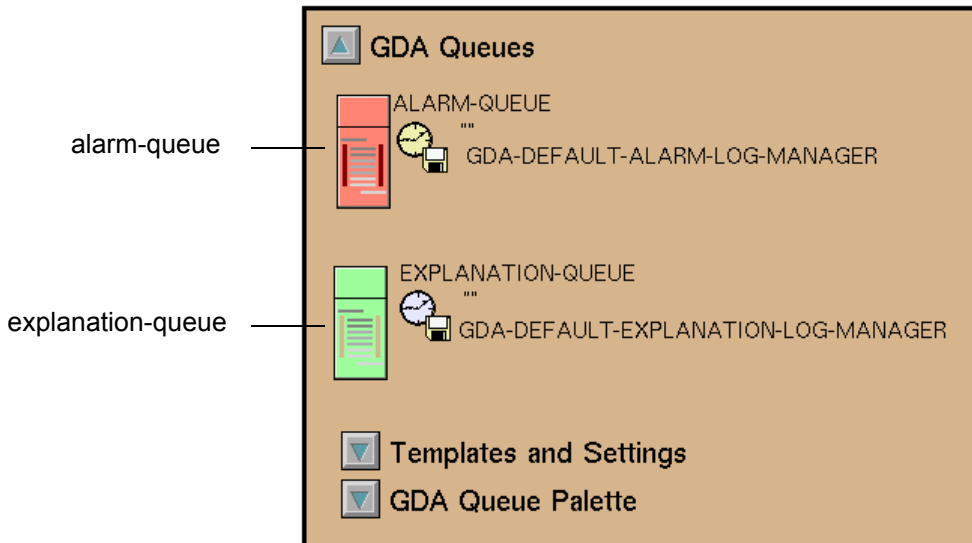
<b>This built-in queue...</b>	<b>Is an instance of this class...</b>	<b>And exists in this module...</b>
alarm-queue	gda-alarm-queue	gda
explanation-queue	gqm-queue	gda
message-queue	gqm-queue	gdabasic
error-queue	gqm-queue	gdabasic

These queue classes inherit from a common superior class, as this class hierarchy shows:

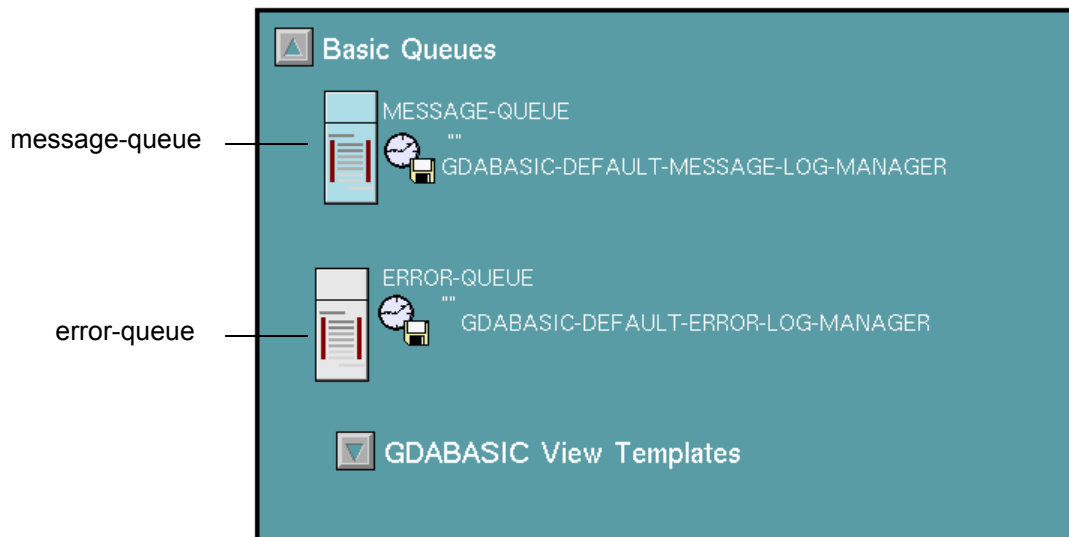


**To create a new queue:**

- 1 Select Inspect from the Main Menu.
- 2 Type go to, then enter the name of the queue you want to clone (alarm-queue, error-queue, explanation-queue, or message-queue).
- 3 If you entered alarm-queue or explanation-queue, this workspace appears:



If you entered `error-queue` or `message-queue`, this workspace appears:



- 4 Click the queue you want to create a new instance of and choose **create new queue instance** from its menu. Click in a workspace to place the item on the workspace.
- 5 Click the new queue instance and choose **name** to specify the name of the new queue.

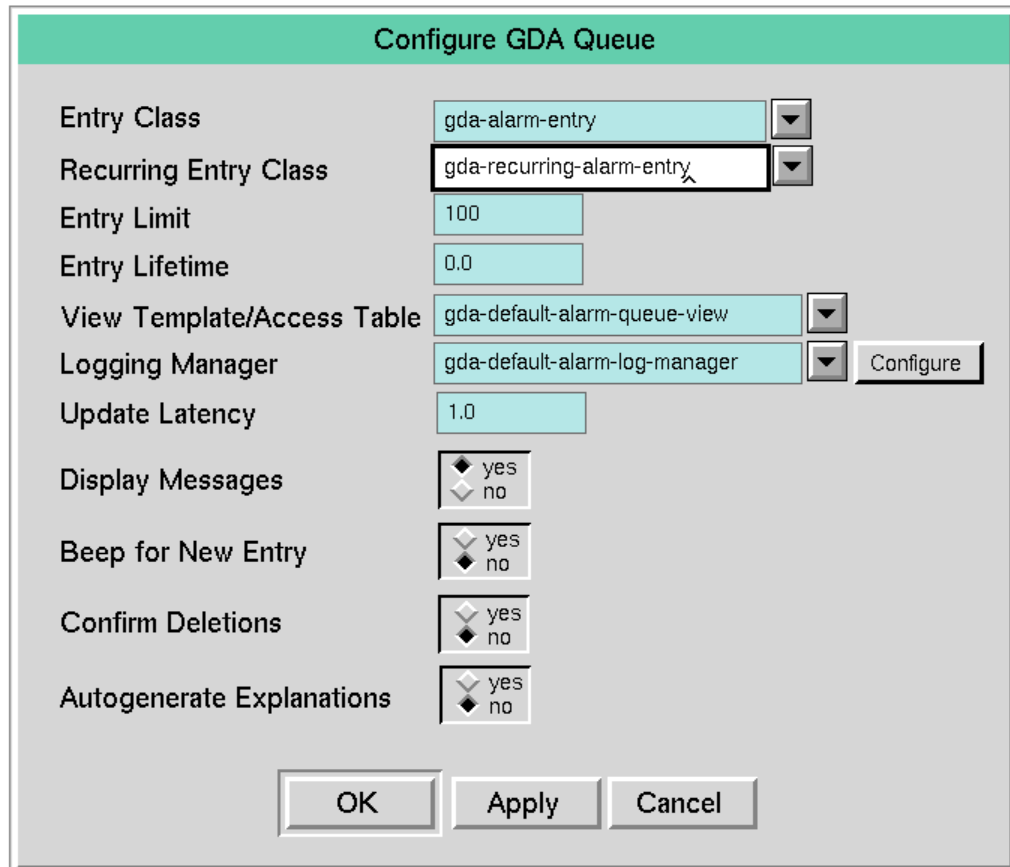
## Configuring a Queue

Once you have created a new queue, you configure it to modify its appearance or behavior. You can also configure the built-in queues without creating new queue instances.

For a queue class, you can configure characteristics of all queues in the class, or you can configure characteristics of individual queues. You can then have an application that provides multiple queues of a particular class, with some characteristics shared by all the queues, and others that make each queue unique. The characteristics that are shared by all queues of a queue type are called *preferences*.

**To configure attributes of a queue:**

- ➔ Select the queue whose attributes you want to configure, then choose **configure** from its menu. For example, this dialog box appears when you configure an Alarm Queue:



The image shows a dialog box titled "Configure GDA Queue". It contains several configuration options:

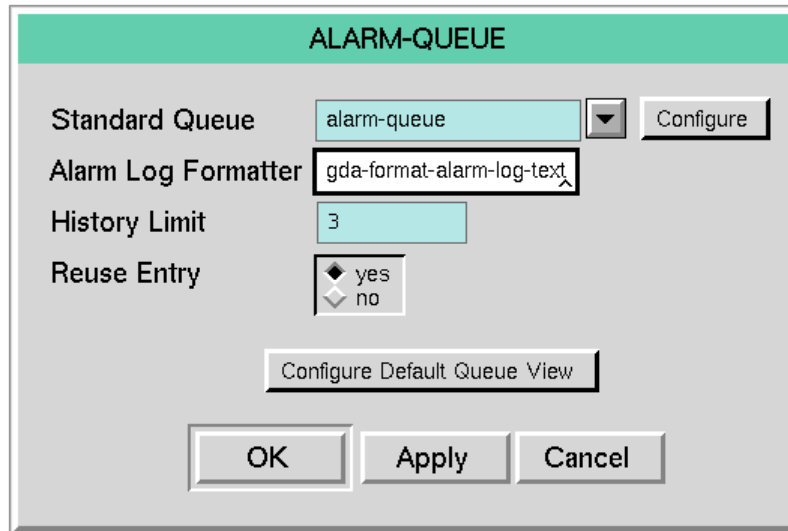
- Entry Class:** gda-alarm-entry (dropdown menu)
- Recurring Entry Class:** gda-recurring-alarm-entry (dropdown menu)
- Entry Limit:** 100 (text input)
- Entry Lifetime:** 0.0 (text input)
- View Template/Access Table:** gda-default-alarm-queue-view (dropdown menu)
- Logging Manager:** gda-default-alarm-log-manager (dropdown menu) with a **Configure** button next to it.
- Update Latency:** 1.0 (text input)
- Display Messages:** yes (radio button selected)
- Beep for New Entry:** no (radio button selected)
- Confirm Deletions:** no (radio button selected)
- Autogenerate Explanations:** no (radio button selected)

At the bottom of the dialog box are three buttons: **OK**, **Apply**, and **Cancel**.



**To configure preferences of a queue class:**

- From the Preferences menu, select Queues, then select the type of queue whose preferences you want to define. For example, this dialog box appears when you set the preferences for Alarm Queues:

**Configuring Attributes that Handle New Entries**

This table lists the attributes that handle new queue entries. Each attribute is described in more detail in the sections that follow.

Attribute	Description
<a href="#">Beep for New Entry</a>	Whether or not to beep when a new entry is generated. The default value is <code>no</code> .
<a href="#">Default Priority</a>	This attribute is not used by GDA. It can be used by the GDA developer as a sorting or filtering attribute. The value of this attribute sets the value of the <code>Gqm-priority</code> attribute on each new entry (this attribute is not available for Alarm Queues).
<a href="#">Display Messages</a>	Whether or not to display a queue view automatically when a new entry is generated. The default value is <code>yes</code> .

Attribute	Description
<a href="#">Entry Class</a>	The class of the entry created when posting to the queue. The default entry class depends on the queue type.
<a href="#">Item Addition Callback</a>	The procedure (callback) that executes when a new entry is posted to a queue. The default is unspecified.

### Beep for New Entry

This attribute determines whether the computer beeps when a new entry is generated. The default is `no`.

If this attribute is set to `yes`, a beep occurs when a new entry is posted to the queue only if the Display Messages attribute is `yes`.

### Default Priority

This attribute is not used by GDA but provides the GDA developer an attribute to use for sorting or filtering.

### Display Messages

This attribute determines whether the queue is displayed when a new entry is posted to it. If you set this attribute to `no`, you can display the queue by choosing Show > Queues and choosing the queue type.

### Entry Class

This attribute indicates the queue entry class that the queue will display. Normally, the entry class should be the default. If you have defined an entry class for queue entries, specify it here. The entry class must be compatible with the queue; it must either be the default class or a subclass of the default class.

The entry constructor for alarm and recurring alarm entries expects that the queue to which the entry is posted is an alarm queue (or a subclass).

In general, it is not necessary to subclass the queue entry classes because most of the attributes of an entry, such as creation time, comments, severity, and acknowledged, are filled in when the entry is created. However, you can create a new queue entry subclass to customize or add attributes. For information on creating a custom queue entry subclass, see the *GDA API Reference*.

### Item Addition Callback

The `Gqs-item-addition-callback` attribute specifies the name of a procedure to execute when an entry is posted to the queue. This enables an application to have

access to an entry as it is posted to a queue through a single procedure. By default, no procedure is associated with the queue types (the value is *unspecified*).

Application developers can monitor new queue entries in these ways:

- Notification of item addition; use the view manager and the subscription feature
- Subclassing the queue; use a post method
- Subclassing the queue entry; use a constructor method

For more information about this callback, see the *GDA API Reference*.

## Configuring Attributes that Handle Changes and Deletions to Entries

This table lists the attributes that handle changes to and removals or deletions of queue entries. Each attribute is described in more detail in the sections that follow.

Attribute	Description
<a href="#">Attribute Update Callback</a>	The procedure (callback) that executes when an attribute of an entry is changed. The default is <i>unspecified</i> .
<a href="#">Confirm Deletions</a>	Whether or not to display a confirmation dialog when the user clicks the Remove Entries or Clear View button to clear entries in the queue. The default value is <i>no</i> .
<a href="#">Item Removal Callback</a>	The procedure (callback) that executes before an entry is removed from one queue but continues to exist in another. The default is <i>unspecified</i> .

### Attribute Update Callback

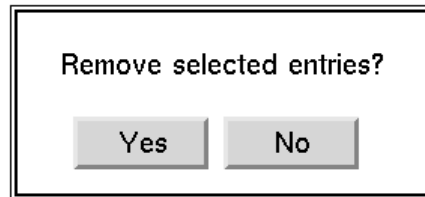
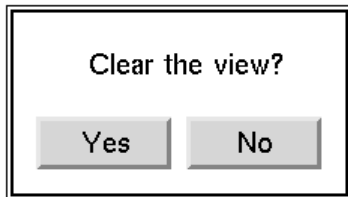
You use the `Gqs-attribute-update-callback` attribute to define the behavior that occurs when an attribute of a queue entry is updated. By default, no procedure is associated with the queue types (the value is *unspecified*).

For more information about this callback, see the *GDA API Reference*.

### Confirm Deletions

This attribute causes a confirmation dialog to be displayed that requests that the user confirm an attempt to either clear entries in a queue by pressing the Clear View button or remove entries from a queue by pressing the Remove Entries button.

The confirmation dialogs look like this:



### Item Removal Callback

A queue entry can exist in more than one queue. If you click the Remove Entries or Clear View button when this is the case, the entry gets removed only from the current queue, but remains in the other queues. If the entry is removed from the last queue, it is deleted.

---

**Note** The Monitor Deletion Events attribute corresponds to the `gqsv-monitor-deletion-events-on-visible-items` attribute of a `gqmv-tabular-view-template`, such as `gda-default-alarm-queue-view`, the default queue view template for the alarm-queue.

---

For more information about this callback, see the *GDA API Reference*.

## Configuring Attributes that Handle Queue Capacity

This table lists the attributes that handle the capacity of queues. Each attribute is described in more detail in the sections that follow.

Attribute	Description
<a href="#">Entry Lifetime</a>	The length of time until the queue entry expires, in seconds. The default value is 0.0, which indicates that the entry never expires.
<a href="#">Entry Limit</a>	The maximum number of entries allowable in the queue. The default value is 100.

### Entry Lifetime

This attribute specifies how long an entry is retained in the queue, in seconds. The default value is 0.0, which indicates that the entry never expires. When an entry expires, it is removed from all queues on which it appears.

---

**Tip** An expired alarm entry is not removed if the entry has not been acknowledged. The alarm becomes inactive when it expires. Removing the entry from the queue view requires that it first be acknowledged. Expired alarm entries are not reused.

---

This attribute applies only to Alarm and Explanation Queue entries, even though the attribute appears on the Error and Message Queue configuration dialogs.

### Entry Limit

This attribute specifies the maximum number of entries allowable in the queue. The default value is 100.

If the maximum is exceeded, the oldest entries are removed until the entry limit is satisfied.

## Configuring Attributes that Are Specific to Alarm Queues

This table lists the attributes that affect only the Alarm Queue. Each attribute is described in more detail in the sections that follow.

Attribute	Description
<a href="#">Alarm Log Formatter</a>	The name of the procedure that determines how GDA formats alarm log text. The default is <code>gda-format-alarm-log-text</code> .
<a href="#">Autogenerate Explanations</a>	Whether or not to generate explanations automatically when an alarm is generated. The default is <code>no</code> .
<a href="#">History Limit</a>	The maximum number of entries allowed in the alarm history. The default value is 50.
<a href="#">Recurring Entry Class</a>	The queue entry class that the Alarm Queue uses to display alarms that originate from a Recurring Alarm Capability. The default is <code>gda-recurring-alarm-entry</code> , which is a direct subclass of <code>gda-alarm-entry</code> . The class must either be the default or a subclass of the default class.
<a href="#">Reuse Entry</a>	Whether the Alarm Queue should reuse existing alarm entries when the same alarm condition occurs and the alarm has not been acknowledged. The default is <code>yes</code> .

## Alarm Log Formatter

This attribute specifies the name of the procedure that determines how GDA formats alarm log text. The default is `gda-format-alarm-log-text`. For more information, see the *GDA API Reference*.

For information on alarm logging, see [Logging Queue Entries](#).

## Autogenerate Explanations

This attribute determines whether to capture the explanation of the conditions that caused an alarm when the alarm occurs and when an alarm changes its state. For this attribute to cause explanations to be generated, the Automatic Explanation attribute of the Alarm Capability must be **yes**. The detail view for the alarm contains the first explanation for an alarm entry and the explanation for the most recent state of the alarm.

The default value for this attribute is **no**, which requires that you click the Update and View Explanations button to generate and view the current explanation. Setting this attribute to **yes** causes the explanation of the condition when the alarm went off to be retained even if the condition that caused the alarm to go off changes.

---

**Note** To generate explanations automatically, you must also set the Automatic Explanation attribute of individual Alarm Capabilities to **yes**.

---

## History Limit

This attribute specifies the maximum number of notations allowed in the alarm history. When the limit is exceeded, older entries are deleted. The default value is 50.

You view the alarm history by viewing the details of an alarm entry. For more information, see [Viewing the Alarm History](#).

## Recurring Entry Class

This attribute specifies the queue entry class that the Alarm Queue uses to generate alarms that originate from a Recurring Alarm Capability. The default value is `gda-recurring-alarm-entry`, which is a direct subclass of `gda-alarm-entry`.

You can provide your own subclass to customize this type of alarm entry. For details, see the *GDA API Reference*.

Alarm history notations are also written to the alarm log, if alarm logging is turned on. For more information, see [Logging Queue Entries](#).

## Reuse Entry

This attribute determines whether the Alarm Queue should use an existing entry when the same alarm condition occurs but the alarm has not yet been acknowledged. A value of **yes** causes GDA to reuse the alarm and store the information in the alarm history; a value of **no** causes GDA to generate a new alarm entry.

## Using Tracebacks for Alarms and Errors

When an error or alarm is generated, GDA can provide traceback information to help determine the cause and location of the condition or problem. You can specify that the traceback information be displayed on the queue detail view by selecting Show Tracebacks on the Configure dialog for these queues. The default value is **yes**, which causes the traceback information to be displayed. This attribute is not available on the Explanation and Message Queues.

## Using a New Queue

Once you have created and named a new queue, you can use the queue in the same way as the corresponding built-in queue.

### To use a new queue in place of a built-in queue:

- 1 Configure the target queue of the appropriate block or capability to be the new queue.

For example, to specify `my-alarm-queue` as the target queue for an Alarm Capability block, display the configuration dialog for the block, then select that queue from the list of queues for the Display Queue attribute.

- 2 Specify the new queue as the Standard Queue attribute for the particular queue type. Doing this enables you to access the queue when you select the queue type from the Show menu.

Select Queues from the Preferences menu, then select the queue type of the new queue. Specify the name of the queue for the Standard Queue attribute.

- 3 Configure the attributes of the new queue, as appropriate. For details, see [Configuring a Queue](#).

## Logging Queue Entries

All queue types support the ability to write queue entries to a log file. You can use this feature to record an entry's activity. The entries are written to the file as they are created. You can also selectively write queue entries to a file using the Save Entries button on the queue view; see [Saving a Queue Entry](#).

This table lists the default log managers for the four queue classes:

<b>Queue Type</b>	<b>Default Log Manager</b>
Alarm Queue	<code>gda-default-alarm-log-manager</code>
Error Queue	<code>gdabasic-default-error-log-manager</code>
Explanation Queue	<code>gda-default-explanation-queue</code>
Message Queue	<code>gdabasic-default-error-log-manager</code>

## Enabling and Disabling Alarm Logging

By default, logging is initially turned off.

### To turn on logging:

- 1 Using Inspect, go to the log manager.
- 2 Display the menu for the log manager and select turn on logging. GDA displays the log file name above the log manager name in the workspace.

### To turn off logging:

- 1 Access the log manager.
- 2 Display its menu and select turn off logging.

## Providing a Name and Location for the Log File

By default, the log file is stored in your working directory and is named `logz_YYYYMMDDHHMMSS.txt`, where

z indicates the queue type: a for Alarm Queue, e for Error Queue, x for Explanation Queue, and m for Message Queue.

YYYYMMDD is the year, month, and day when the file is created.

HHMMSS is the hour, minutes, and seconds when the file is created.

### To specify a name and location for the log file:

- 1 Access the `gda-default-alarm-log-manager` and display its table.
- 2 Set the `Glif-log-directory` attribute to the directory where the log file is to reside. By default, the log file is stored in the same directory from which G2 is loaded.
- 3 The `Glif-log-file-name-template` attribute specifies the pattern of the log file name. When creating a file name, GDA substitutes the date and time the file is created at the location of the \* character.



To specify a different log file name, change this attribute value. You cannot specify more than one \* in the log file name template.

## When GDA Creates a New Log File

GDA creates a new log file whenever:

- Logging is enabled after having been disabled.
- Logging is enabled and G2 is started or restarted.
- When a log file is open and the number of seconds specified by the log manager's `Glf-time-interval-to-open-new-file` attribute has elapsed from the time the log file was created. The default is 86400 seconds, equal to 24 hours.
- When the size of the current log file exceeds the number of bytes specified by the log manager's `Glf-maximum-file-size-in-bytes` attribute. The default is 100,000 bytes.

When a new log file is created and the most recently created log file is empty, that file is deleted if the log manager's `Glf-automatically-delete-empty-log-files` attribute is true.

## Determining the Log File Header and Message Contents

The log file contains:

- The log file name
- The log file header
- Log entries

### The Log File Name

The log file name is centered on the top line of the log file and does not include the `.txt` extension.

### The Log File Header

The log file header, which appears below the log file name, consists of the name of the log manager and the time the log file was opened, preceded and followed by a row of equal signs. By default, the log file header is generated by the procedure named in the log manager's `Glf-default-log-file-header-writer` attribute.

Here is an example of a log file header for alarm entries:

```
=====
Log file for GDA-DEFAULT-ALARM-LOG-MANAGER, opened at time 10 Aug
1999 11:53:12 a.m.
=====
```

## Alarm Log Entries

Each alarm log entry consists of a header, severity and time information, and the description for the alarm. By default, these entry components are formatted as follows:

- The header includes the UUID of the entry on a line that starts and ends with five asterisks.
- The next line indicates the severity of the alarm, the time the alarm was raised, and the collection time for the alarm.
- The entry also includes the description for the alarm the first time the alarm is entered. The description is defined by selecting the Description button on the configuration dialog box for the block connected to the Alarm block.

For example, the next four lines make up a sample initial alarm log entry:

```
***** GDA-ALARM-ENTRY-006008C5F93C-934302445.274-8690 *****
Severity: 2 At: 12:27:25 CT: 12:27:24
```

Tank is hot.

When an alarm that requires acknowledgement is acknowledged, GDA writes summary and history information about the alarm to the log. The log contains additional information depending on the value of attributes of the log manager:

- The advice, if the Gda-log-advice attribute is true
- Comments entered by the operator, if the Gqm-log-comments attribute is true
- The explanation for the alarm, if the Gda-log-explanations attribute is true

For example, this excerpt is written to the log when an alarm is acknowledged and explanations, advice, and comments are written to the log:

```
***** Summary: GDA-ALARM-ENTRY-006008C5F93C-934302445.274-8690 *****
Severity: 2 At: 12:27:25 CT: 12:27:24
```

Acknowledged on 08/10/1999 at 12:28:09

----- History -----

Entered alarm on 08/10/1999 at 12:27:25

Returned from alarm on 08/10/1999 at 12:27:57

Entered alarm on 08/10/1999 at 12:28:00

Returned from alarm on 08/10/1999 at 12:28:02

----- Explanation -----

08/10/1999 12:27:42

Tank is hot. because the input value of GDL-HIGH-VALUE-OBSERVATION-XXX-171 = 100 (threshold: 90)

----- Advice -----

This is the advice defined for the tank temperature observation.

----- Comments -----

This is a comment entered by the operator from the entry in the alarm queue

## Error Log Entries

Each error log entry consists of a header and the description for the error. By default, these entry components are formatted as follows:

- The header includes the date and time of the error and its priority.
- The next lines include the date and time of the error, then the description for the error.

## Explanation Log Entries

Each explanation log entry consists of a header, a date and time stamp, and the description for the explanation. By default, these entry components are formatted as follows:

- The header includes the UUID of the entry on a line that starts and ends with five asterisks.
- The next line indicates the date and time the entry was generated.
- The lines that follow contain the explanation.

## Message Log Entries

Each message log entry consists of a date and time stamp followed by the message.

## Customizing the Entry

You can modify the format of entries written to an Alarm Queue log. For more information, see the description of the `Alarm-log-formatter` attribute in the *GDA API Reference*.

## Incremental Logging of Alarm Entries

If the log manager's `Gda-log-events-incrementally` attribute is true, a log entry is written each time a condition goes into or out of alarm.

If the attribute is **false** (the default), a log entry is written only the first time the alarm is raised. When the alarm is acknowledged, the alarm history provides the incremental information for the alarm.

## The Time Format for Alarm Entries

You can modify the format of the times posted for each alarm entry. The log manager's `Gqm-time-format` attribute controls the format of the time. By default, the format is determined by the `datetime` format. You can also define this attribute value to be `time`, `date`, or `filetime` format. These formats are defined on the General Queue Settings dialog box, accessible by choosing the **Preferences > Queues > General Settings** menu item. For a list of formats, see [Configuring Date and Time Formatting](#).

# Creating Queue Views

---

*Describes how to customize aspects of the queue views.*

Introduction **235**

Characteristics of the Built-in Queue Views **236**

Creating a New Queue View Template **240**

Configuring a Queue View Template **242**

Configuring the Detail View **266**

Creating and Configuring the Access Manager **273**



## Introduction

All queue entries posted to a queue are held by the queue but are not necessarily displayed to the user. The user sees entries on a **queue view**, which is a formatted workspace containing information about the queue, buttons for manipulating entries, and a table of queue entries.

The appearance, contents, and behavior of the queue view are determined by these factors:

- View templates associated with the queue
- The access manager associated with the queue
- Filters applied to the queue

A **view template** defines a queue view, controlling the physical appearance and behavior of the queue. GDA supplies a default view template for each queue type. You can use this template, modify it, or create and configure a new template.

If you want to display queue entries for a particular queue type on just one view, but you want that view's appearance or behavior to differ from the built-in view, you can either modify the built-in queue view template or create and configure a new one.

If you want to display queue entries for a particular queue type on more than one view, you need to create and configure new templates. If you do this, you also need to create an access manager for the queue. The **access manager** specifies the view template that determines the particular view seen by a specific user when an application provides more than one view for a queue. If you need to use an access manager, you must create one. For more information, see these sections:

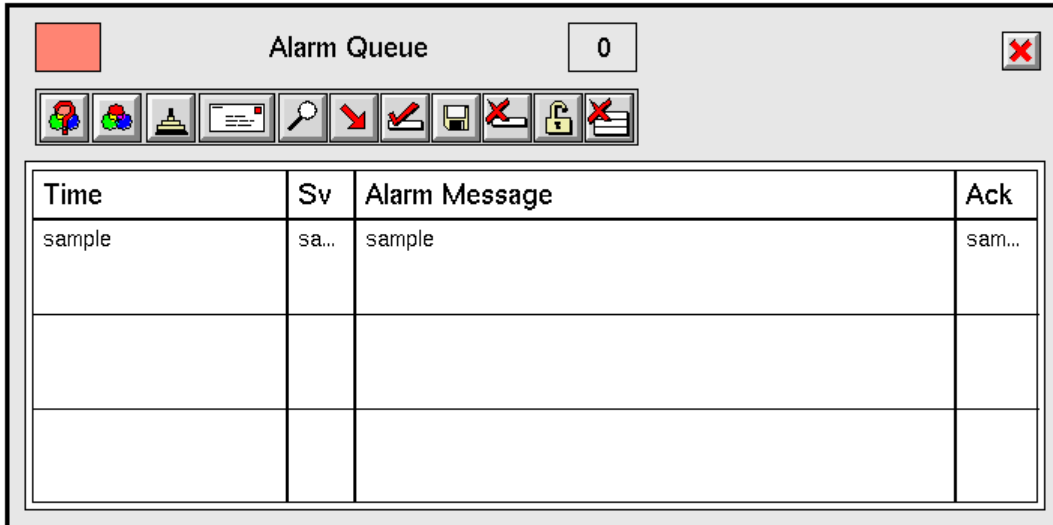
- [Creating a New Queue View Template](#)
- [Configuring a Queue View Template](#)
- [Creating and Configuring the Access Manager](#)

A **filter** determines which queue entries are to be displayed on a queue view by applying specified criteria to each entry. Only entries that satisfy the criteria are displayed. For more information, see [Filtering Queue Entries](#).

## Characteristics of the Built-in Queue Views

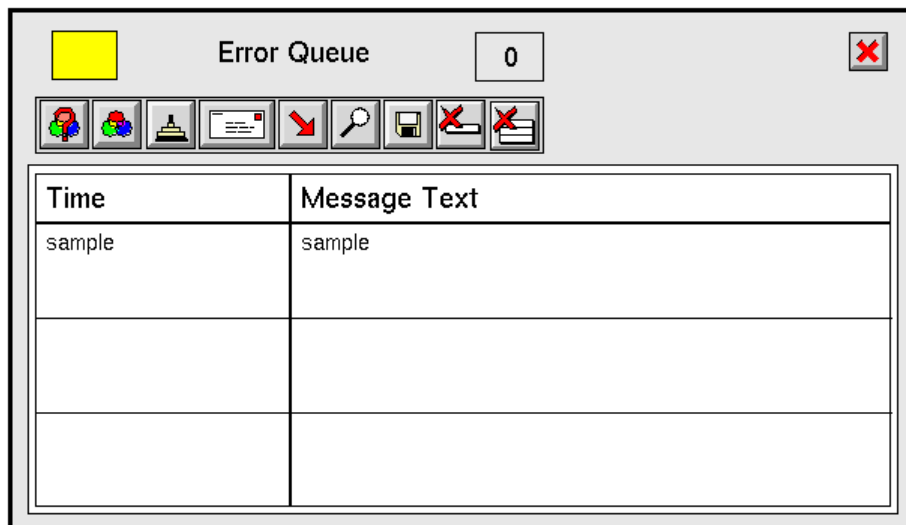
GDA provides a queue view template that defines the appearance and behavior of the built-in queue view for each queue type. This section shows what each built-in queue view looks like and lists queue view characteristics.

The built-in Alarm Queue view template generates an Alarm Queue view that looks like this:



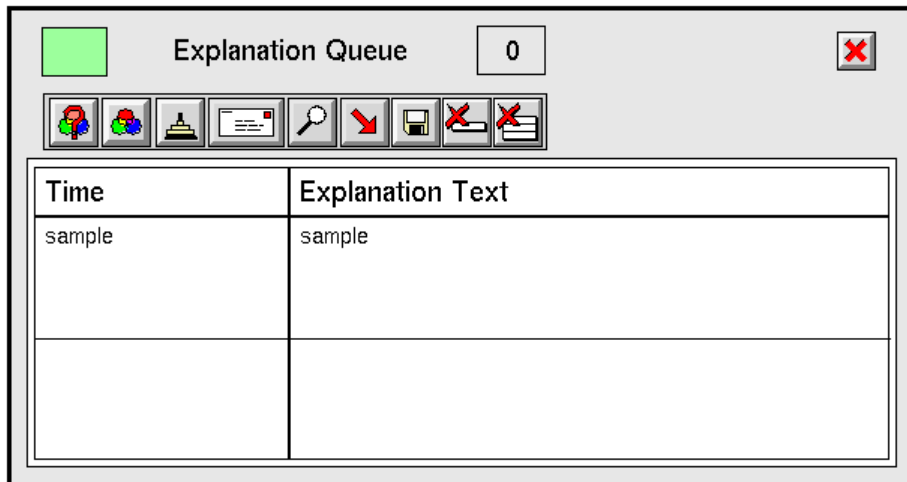
The Alarm Queue has these characteristics: a capacity of 100 rows, displaying 3 rows at a time; each row uses 3 lines. Text is displayed in small font. The history has a capacity of 50 events. The entries do not expire; they do not beep when generated; do not confirm deletion; are sorted by severity in ascending order; and new entries are sorted. No callbacks are specified. Explanations are not automatically generated. When the entry limit is reached, entries are removed according to age.

The built-in Error Queue view template generates an Error Queue view that looks like this:



The Error Queue has these characteristics: a capacity of 100 rows, displaying 3 rows at a time; each row uses 3 lines. Text is displayed in small font. The entries do not expire; they do not beep when generated; do not confirm deletion; are sorted by age in descending order; and new entries are sorted. No callbacks are specified. Error tracebacks are displayed. When the entry limit is reached, entries are removed according to age.

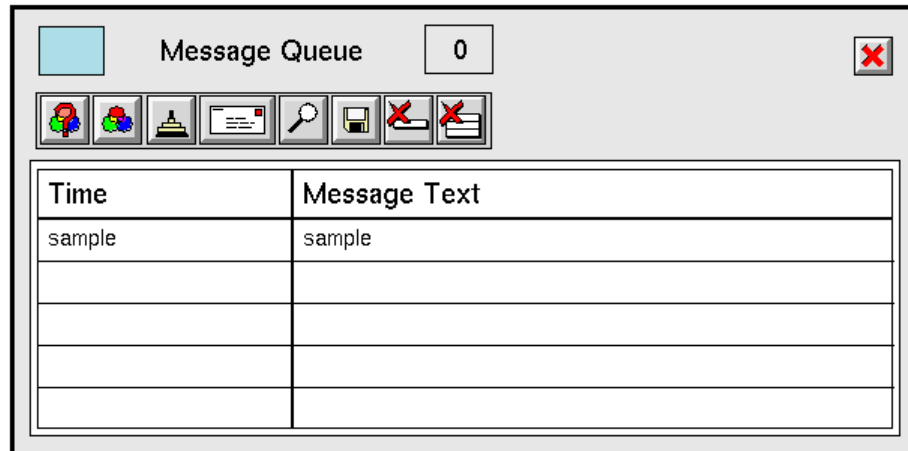
The built-in Explanation Queue view template generates an Explanation Queue view that looks like this:



The Explanation Queue has these characteristics: a capacity of 100 rows, displaying 2 rows at a time; each row uses 4 lines. Text is displayed in small font. The entries do not expire; they do not beep when generated; do not confirm deletion; are sorted by age in descending order; and new entries are sorted. No callbacks are specified. When the entry limit is reached, entries are removed according to age.



The built-in Message Queue view template generates a Message Queue view that looks like this:



The Message Queue has these characteristics: a capacity of 100 rows, displaying 5 rows at a time; each row uses 1 line. Text is displayed in small font. The entries do not expire; they do not beep when generated; do not confirm deletion; are sorted by age in descending order; and new entries are sorted. No callbacks are specified. When the entry limit is reached, entries are removed according to age.

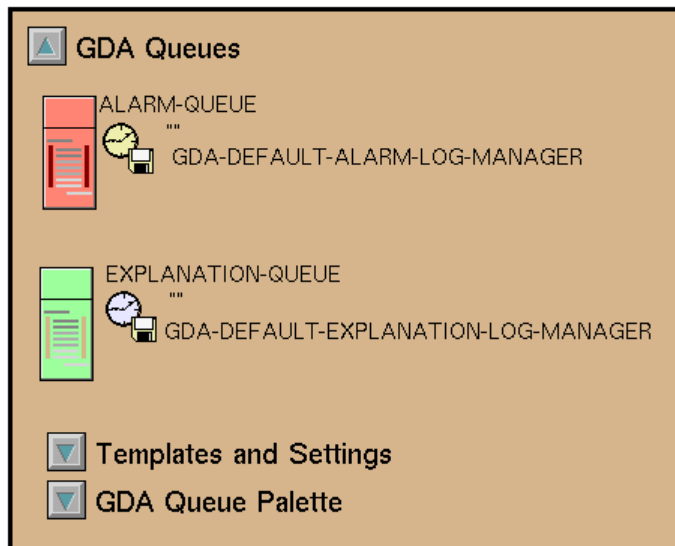
# Creating a New Queue View Template

To create a new queue view, you first access the view templates, then clone a template to your workspace.

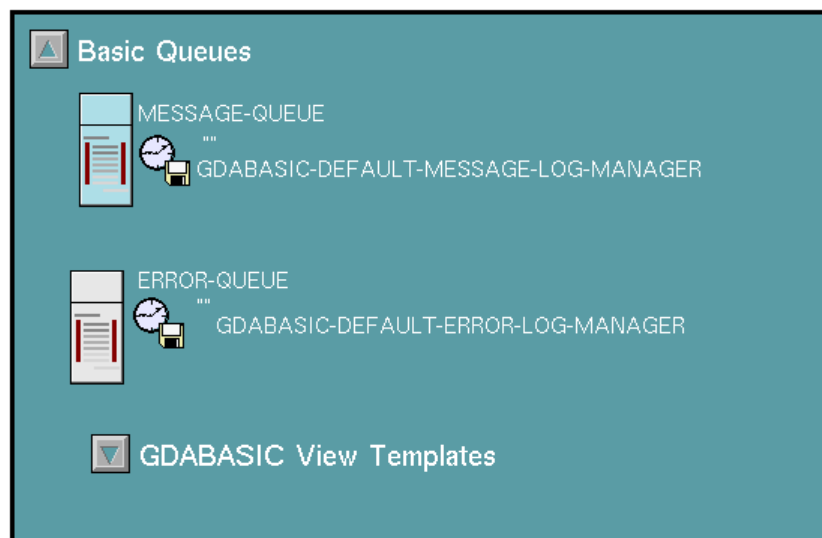
**To access the templates for the queues:**

- 1 Using Inspect, enter the go to command followed by the name of the queue.

If you accessed the Alarm Queue or Explanation Queue, this workspace appears:

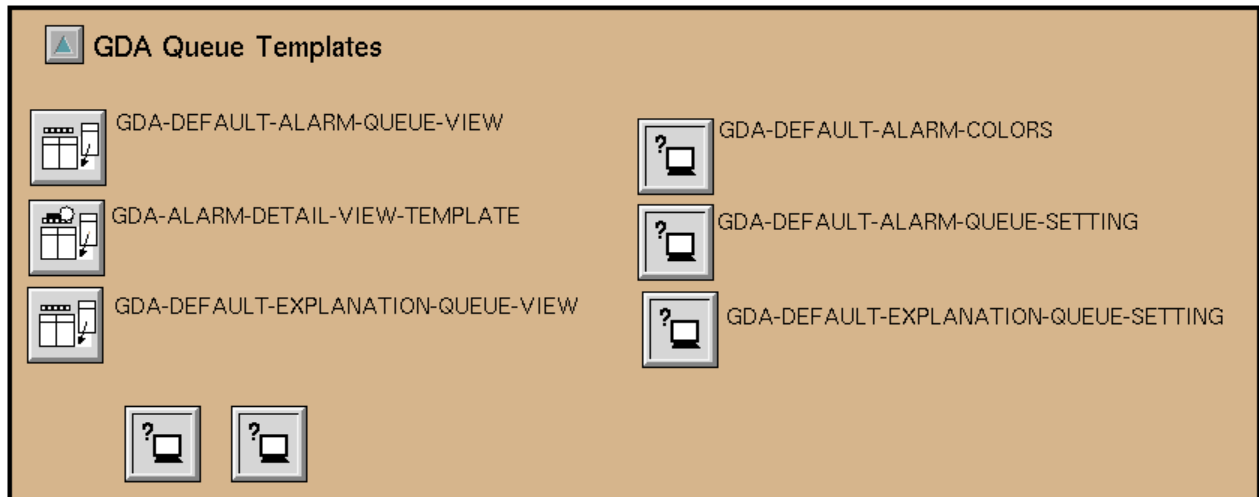


If you accessed the Error Queue or Message Queue, this workspace appears:



- Then, to access the view templates, click the Templates and Settings button on the GDA Queues workspace, or the GDABASIC View Templates button on the Basic Queues workspace.

If you are accessing templates for the Alarm Queue or Explanation Queue, this workspace appears:



If you are accessing templates for the Error Queue or Message Queue, this workspace appears:



#### To create a new queue view template:

- Click the queue view template item on the workspace you displayed in the previous step (for the Alarm Queue, click `GDA-default-alarm-queue-view`; for the Explanation Queue, click `GDA-default-explanation-queue-view`) and choose `clone` from its menu.
- Move the template item into your workspace.
- Choose `name` and provide a unique name for the new template.

# Configuring a Queue View Template

You can configure these elements of the queue view for each type of queue:

- The number of lines for each entry
- The font size for entry text
- Sorting options, including the column on which to sort, the sort order, whether new entries are to be sorted with existing entries, and whether clicking on a column header causes entries to be sorted
- The queue view label, including the text and colors
- The queue view colors
- The toolbar buttons, including the location, attributes, and location
- The columns that contain values for the attributes of the entries, and their headers
- The queue entry counters, including the location and colors

For example, you might want the default template to display more than the default number of rows, or you might want to include additional columns to display other queue entry attributes.

You can cancel changes you make to the table (columns) in the layout. You cannot cancel other changes you make to the layout, such as buttons or counters, although you can change them back to their original state.

The changes you make to the default queue view layout take effect the next time you display a queue; currently visible queue views are not affected.

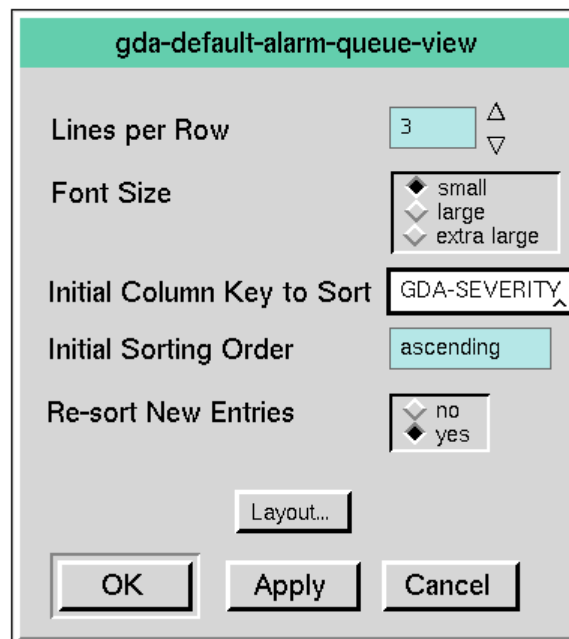
The steps you follow to configure a queue view depend on whether you are configuring a built-in queue view template or a new queue view template. Both follow.

## Configuring a Built-in Queue View

To modify the built-in queue view associated with a particular queue, follow these steps.

### To configure a built-in queue view:

- 1 Choose Preferences > Queues, then choose the type of queue whose default view you want to configure.
- 2 Click the Configure Default Queue View button. For the Alarm Queue, this dialog box appears:



Some queue view characteristics are accessible on this dialog. For these, see [Modifying Queue View Attributes on the Configure Dialog](#). Other attributes, which you access by clicking the Layout button, are described in [Configuring the Layout of a Queue View](#).

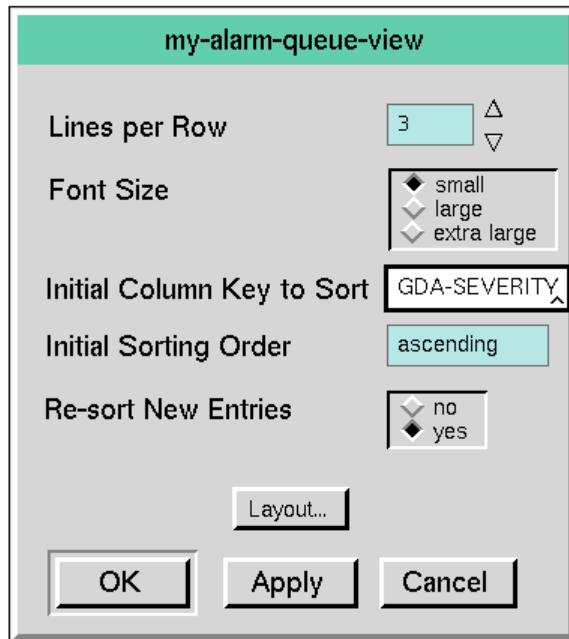
## Configuring a New Queue View

To modify a new queue view, follow these steps.

### To configure a new queue view:

- ➔ Click the queue view template item, then choose **configure** from its menu. This is the dialog box that appears for the Alarm Queue view template (named my-

alarm-queue); the configuration dialog boxes for other view templates are similar.



## Modifying Queue View Attributes on the Configure Dialog

You can configure the lines per row, font size, and sort attributes directly on the Configure Queue View dialog box.

To apply modifications to the Font attribute and the sorting attributes, click the Apply button to save the changes while keeping the dialog box open. Click the OK button to save the changes and close the dialog box. Click Cancel to close the dialog box without making changes. If you make changes and click Apply, then click Cancel, only the changes made since you clicked Apply are not saved.

### Configuring the Lines per Row Attribute

The Lines per Row attribute determines the number of lines used to display each queue entry. The default value depends on the queue type: the Alarm Queue is 3; the Explanation Queue is 4; the Error Queue is 3; and the Message Queue is 1. Click the arrows to increase or decrease the number of lines per row.

When you modify the lines per row attribute, GDA displays the view template so you can see the effects of the change. To accept the change, click on the background of the template and select the **Finished Configuration** menu item. To return to the dialog box without making the change, select the **Cancel Configuration** menu item.

## Configuring the Font Size Attribute

The Font Size attribute sets the size of the font to use for queue entry text. The options are `small`, `large`, and `extra-large`. The default is `small` for all queue types.

## Configuring Sorting Attributes

The Initial Column Key to Sort, Initial Sorting Order, and Re-sort New Entries attributes control the sorting of queue entries.

The Initial Column Key to Sort attribute specifies the default column used to sort entries. The sort key is a symbol that names a queue entry attribute, or a key symbol that refers to a derived function. For information on the attributes you can use for sorting, see the tables in [Entry Attributes Used with Filters](#).

- The default for the Alarm Queue is `gda-severity`, which sorts entries based on severity.
- The default for the Error, Explanation, and Message Queues is `gqm-creation-time`, which sorts entries based on age.

The Initial Sorting Order attribute specifies whether to sort entries in ascending or descending order.

- The default for the Alarm Queue is `ascending`, which displays entries with the lowest severity at the top.
- The default for the Error, Explanation, and Message Queues is `descending`, which displays the newest entries at the top.

The application user can override these settings. For details, see [Sorting Entries](#).

The Re-sort New Entries attribute specifies whether to re-sort queue entries when a new entry arrives. The default is `yes`. Setting the value to `no` causes new entries to be added to the end of the entry list.

You can specify whether users can sort entries by clicking on a column header. For more information, see [Specifying Whether Clicking on a Column Header Sorts Entries](#).

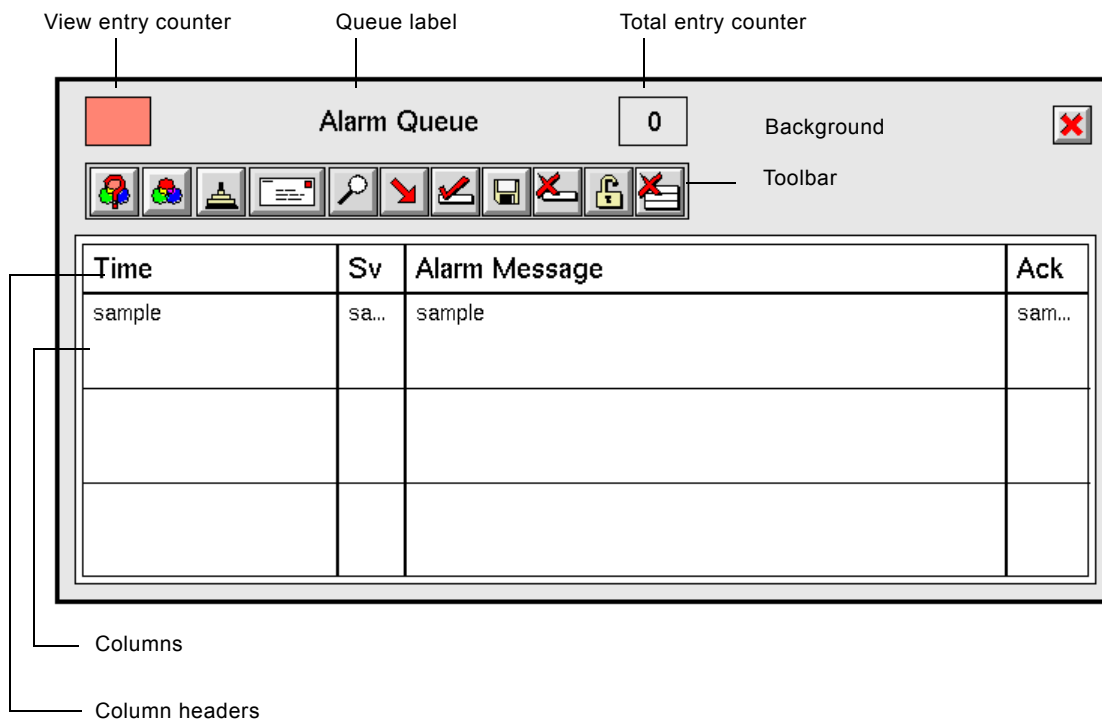
## Configuring the Layout of a Queue View

To modify the layout associated with a particular queue view, follow these steps.

### To configure the layout of a built-in queue view template:

- 1 From the Preferences menu, choose Queues, then choose the queue type whose template you want to configure.
- 2 Click the Configure Default Queue View button, then click the Layout button.

Here is the built-in queue view template for the Alarm Queue:



### To configure the layout of a queue view template:

- ➔ Click the template item and choose **configure tabular view**. The layout workspace that appears is similar to the one shown above (or the same, if you are configuring a view template for an Alarm Queue).

## Modifying the Queue View Label

The queue view label is the text centered above the toolbar buttons. The label consists of the text, the border around the text (by default, transparent), and the background. The default label text is the queue type. You can modify the text, position, and colors of the label.

### To modify the queue view label:

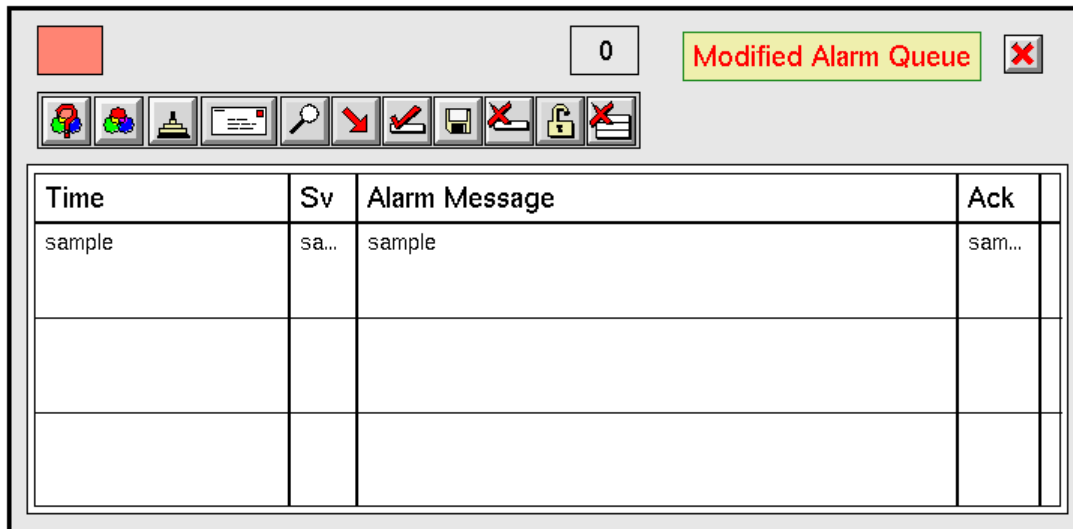
- 1 To modify the queue view label text, click the text to display the editor and enter a new string.
- 2 To modify the label position, drag the label to a new location on the template.
- 3 To modify the color of the text, background, or border:
  - a Click the border around the label to display the gqmv view label menu. Because the border is not displayed, you need to click a short distance



from the text to select the border instead of the text or the template background.

- b** Choose color.
- c** Choose background-color, foreground-color, or text-color.
- d** Choose the desired color from the palette.

This figure shows the Alarm Queue view with a queue label whose position, text, and colors have been modified:



## Modifying Queue View Colors

You can modify the background and foreground colors of the default queue view by configuring the workspace colors.

- The queue **background** is the gray area.
- The queue **foreground** controls the color of the column header text, the border around the column header and between each header, the border around the table, and the label text, if it was not explicitly changed.

### To configure overall view colors:

- 1** Click the grey background of the view. On the KB Workspace menu, choose Color.
- 2** To modify the queue background color, choose background-color to display a palette of colors, then select a color.
- 3** To modify the queue foreground color, choose foreground-color to display a palette of colors, then select a color.

- 4 Save the changes by choosing **Finished Configuration** from the KB Workspace menu. To cancel the change, you must manually return the colors to their original values.

## Manipulating Toolbar Buttons

You can delete, move, and add toolbar buttons to the queue view. You can also create customized buttons by subclassing them and defining their public methods.

You manipulate toolbar buttons from the queue view template. If you need directions for displaying the template, see [Configuring the Layout of a Queue View](#).

### Deleting Toolbar Buttons

You might not want certain toolbar buttons to appear on some queue views.

#### To delete a button from the view:

- 1 Click the button you want to delete and choose **delete** from its menu.
- 2 Move other buttons to a new location, as needed, as described in the next section.

If you delete buttons, they do not get restored if you choose **Cancel Configuration**.

### Moving Toolbar Buttons

#### To move a button:

- ➔ Drag the button to a new location.

---

**Tip** To adjust precisely the position of a button, scale the view until it is very large by repeatedly entering **Ctrl + b**, move the button, then return the view to full-size by entering **Ctrl + f**.

---

If you move buttons, their positions do not get reset if you choose **Cancel Configuration**.

### Adding Toolbar Buttons

You add buttons to the toolbar by cloning them from one of two palettes:

- The GDA Queue Palette, which contains buttons for acknowledging and deleting alarms.
- The GQM Views Palette, which contains all the other buttons on the queue view.

---

**Caution** You should not add detail view (subview) buttons to a queue view; these buttons only work on the detail view.

---



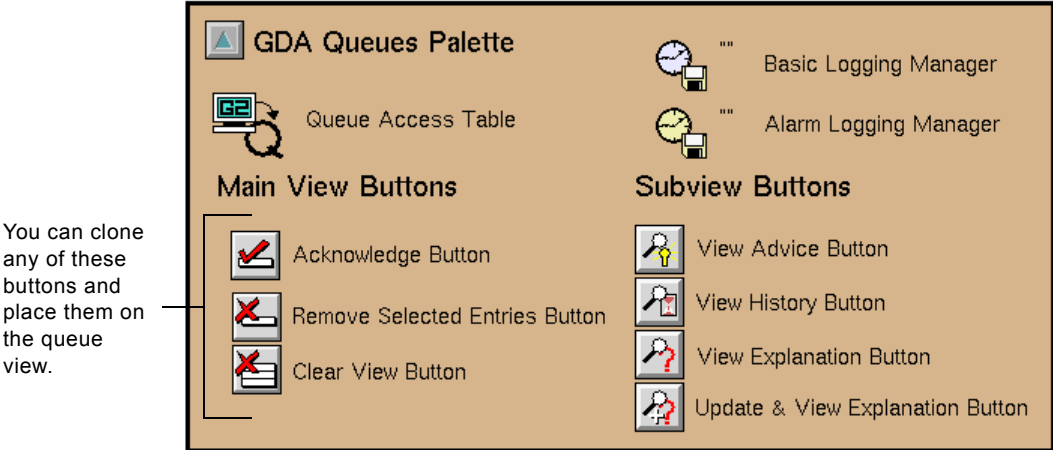
---

**Caution** You should not add the Go to Source button to a Message Queue view because a gqm-entry does not have a source. This button only works for the Alarm Queue, Error Queue, and Explanation Queue.

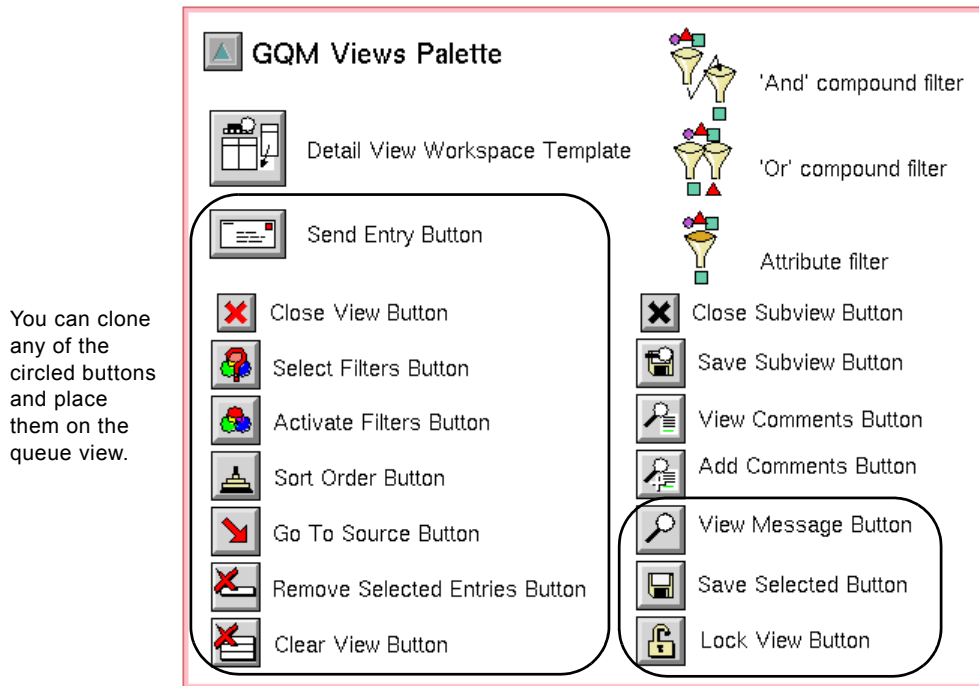
---

#### To add buttons to the toolbar:

- 1 To add buttons that apply to the Alarm Queue:
  - a Using Inspect, enter go to alarm-queue.
  - b Click the GDA Queue Palette button to display this workspace:



- 2 To add buttons that apply to other queues, display the GQM Views palette of buttons:
  - a In Administrator mode, choose Main Menu > Get Workspace > GQMV-TOP-LEVEL.
  - b Click the GQM Views Palette button to display this workspace:






- 3 Clone the desired button from one of these two palettes and place it in the toolbar area of the default queue view. For information about moving buttons, see [Moving Toolbar Buttons](#).
- 4 Move the other buttons to a new location, as needed.

When you are finished configuring the queue view, click on the background of the layout to display the KB Workspace menu, then select **Finished Configuration**, at the bottom of the menu choices.

If you add buttons, they do not get removed if you choose **Cancel Configuration**.

## Modifying Button Attributes

You can configure attributes for three buttons on the built-in queue view toolbar. The other buttons have no configurable attributes.

Button	Attribute	Description
 View Details	Detail View Template	<p>The View Details button displays information about the selected queue entry on a detail view, defined by the detail view template.</p> <p>This attribute specifies the name of the template the queue uses for the detail view. The default for the Alarm Queue is <code>gda-alarm-detail-view-template</code>. The default for the Error Queue, Explanation Queue, and Message Queue is <code>gqmv-default-detail-view-template</code>. Both are instances of the class <code>gqmv-detail-view-template</code>.</p>
 Go to Source	Highlight Procedure	<p>The name of the procedure that determines the action that occurs when the user selects an entry and clicks the Go to Source button. The default is <code>gqsv-highlight-item</code>, which displays the workspace of the source and places a flashing arrow next to the source for 10 seconds. If the source does not appear on a workspace, the procedure does nothing.</p> <p>For information on customizing this procedure, see the <i>GDA API Reference</i>.</p>
 Remove Entries (Alarm Queue only)	Acknowledge Selected Entries	<p>Whether or not to acknowledge selected Alarm Queue entries when removing them. The default is <code>no</code>, which means the user must acknowledge selected entries as a separate action by clicking the Acknowledge Entries button. Set to <code>yes</code> to acknowledge and remove entries in a single step.</p> <p>If you set this attribute to <code>yes</code>, you might consider removing the Acknowledge button from the toolbar, as described in <a href="#">Deleting Toolbar Buttons</a>.</p>

### To modify a button attribute:

- 1 Click the button whose attribute you want to configure and choose the configure button menu choice.
- 2 Edit the attribute in the dialog that appears, then press OK or Apply.
- 3 To apply the edits, click on the background of the template and choose Finished Configuration from the KB Workspace menu.

To cancel the edits, click on the background of the template and choose Cancel Configuration from the KB Workspace menu.

## Creating and Customizing Buttons

This section provides the name and class of each button on the queue view and the detail view.

To customize an existing button, override one of the public methods, as described in the *GDA API Reference*.

### Queue View Button Classes

This table indicates the class of which each detail view button is an instance:

<b>This button...</b>	<b>Is an instance of this class...</b>
Acknowledge	gqsv-acknowledge-button
Activate Filters	gqsv-activate-filters
Clear View	gda-clear-alarm-entries-button (on Alarm Queue) gqmv-clear-entries-button (on other queues)
Close View	gqmv-close-queue-view-button
Go To Source	gqmv-go-to-source-button
Lock View	gqsv-lock-view-button
Remove Selected Entries	gda-remove-alarm-entries-button (on Alarm Queue) gqsv-remove-item-button (on other queues)
Save Selected	gqmv-save-selected-button
Select Filters	gqsv-select-filters-button
Send Entries	gqsv-send-entry-button
Sort Order	gqsv-sort-order-button
View Message	gqmv-view-entry-details-button

All of the above classes are subclasses of the `gqsv-toolbar-button` class. Mouse tracking is enabled for the `gqsv-toolbar-button` subclasses. If you create a new

class from this class or a subclass of it, the new button will behave properly when the user clicks the button. If you create a button that does not inherit from this class, you will need to write your own mouse tracking procedure.

### Detail View Button Classes

This table indicates the class of which each detail view button is an instance:

This button...	Is an instance of this class...
Add Comments	gqmv-add-comment-button
Close Subview	gqmv-close-view-button
Save Subview	gqmv-save-subview-button
Update & View Explanation	gda-update-explanation-button
View Advice	gqmv-view-advice-button
View Comments	gqmv-view-comments-button
View Explanations	gda-view-explanation-button
View History	gda-view-history-button
View Message	gqmv-view-message-button

All of the above classes are subclasses of the `gqmv-action-button` class. Mouse tracking is enabled for all `gqmv-action-button` subclasses. If you create a new class from this class or a subclass of it, the new button will behave properly when the user clicks the button. If you create a button that does not inherit from this class, you will need to write your own mouse tracking procedure.

### Modifying the Queue Entry Counters

The default queue view has two queue entry counters:

- The view entry counter, which indicates the number of entries that are currently visible in the queue view. The `Gqmv-count-id` attribute determines what this counter indicates; for the view entry counter, its value is `view-count`.
- The total entry counter, which indicates the total number of entries in the queue when the number of entries in the view is different than the number in the queue. If no entries are filtered out when the filters are activated, this counter does not appear. The `Gqmv-count-id` attribute determines what this counter indicates; for the view entry counter, its value is `queue-count`.

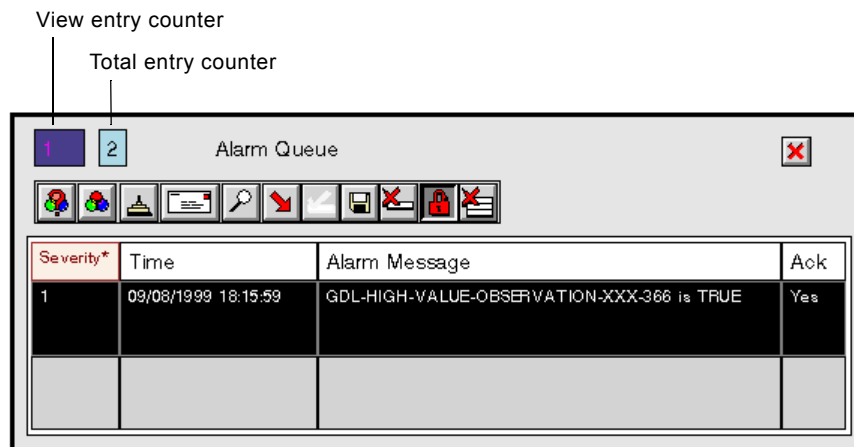
You can modify the position, size, and color of these counters.

You cannot undo changes you make to these counters by choosing **Cancel Configuration** from the queue view menu.

### To modify the queue entry counters:

- 1 To modify the position of a counter, drag the counter to a new location on the view.
- 2 To modify the size of the counters, click the border around the counter and choose **change min size**, drag the edges of the black box to define its new size, then click **Update Now** or **Yes**.
- 3 To modify the color of a counter, click the border around the counter and choose **color**, then choose **background-color**, **border-color**, or **text-color** to configure these colors.

This figure shows the result of modifying the view and total entry counters of an Alarm Queue view:



## Modifying Columns

You can modify these characteristics of columns:

- The columns that appear in the view
- The order of the columns
- The font used to display the text in the columns
- The background, text, and border colors of the column cells
- The number of rows visible in the view
- The height of each cell in a column
- The type of data that can be displayed in the column
- Whether cells can be selected or edited



- Whether clicking on a column header sorts entries
- The procedure that validates a user-entered value
- Callback procedures that execute when a value is entered in a cell or when the cell is selected
- How columns format floating point values, dates and times, and ordinal numbers

To modify the column header label and contents, as well as other column header attributes, see [Modifying Column Headers](#).

### **Specifying the Columns that Appear in the View**

You can determine which attributes are displayed in columns in the view by adding or deleting columns.

#### **To add a column to the view:**

- 1 You can insert a new column before or after an existing column. Select the column header of the column before or after which you want to add the new column header to display its menu.
- 2 Select either add new column after or add new column before.

#### **To delete a column from the view:**

- ➔ Select the column header of the column you want to delete and choose delete column.

### **Moving Columns in the View**

You change the order in which columns appear by physically moving columns.

#### **To move a column in the view:**

- 1 While the pointer is on the column header of the column you want to move, click and hold down the mouse button.
- 2 Drag the column header until it appears where you want the new location to be.
- 3 Release the mouse button.

### **Changing the Number of Rows Visible in the Queue View**

By default, the Alarm Queue and Error Queue display 3 rows, the Explanation Queue displays 2 rows, and the Message Queue displays 5 rows.

#### **To change the number of visible rows:**

- 1 Click on any column cell and choose table.
- 2 Edit the Number of visible rows attribute.

## Changing the Font Size of the Text Displayed in a Column

By default, the font size of the text in the column cells is **small**.

### To change the font size of text displayed in a column:

- 1 Click on the header of the column whose font size you want to change and choose **table**.
- 2 Edit the **Font size** attribute. You can define the font size as **small**, **large**, and **extra-large**.

## Changing the Column Color

By default, the background color of all column cells is **white**, the text color is **black**, and the border color is **black**.

### To change the colors of a column:

- 1 Click on a cell of the column whose colors you want to change and choose **table**.
- 2 Edit the **Default background color**, **Default text color**, or **Default border color** attribute by entering the name of a color. To obtain a list of available colors, click the background of the view template and choose **Color > background-color**, then specify one of these colors as a symbol, for example, **forest-green** (colors consisting of more than one word separate the words with a hyphen).

## Changing the Height and Width of Columns

You can adjust the height and width of columns (in pixels) by modifying column and column header attributes. The default column height is 63 pixels.

### To modify the height of all columns:

- 1 Click on any column cell (not column header) and choose **table**.
- 2 Edit the **Cell-height** attribute.

### To modify the width of a column:

- 1 Click on the column header for the column you want to adjust and choose **table**.
- 2 Edit the **Width** attribute.

## Specifying Whether Clicking on a Column Header Sorts Entries

You can specify whether clicking on a column header causes entries to be sorted.

**To specify whether clicking on a column header sorts entries:**

- 1 Click on the column header for the column you want to control and choose `table`.
- 2 Edit the `Allow-click-to-sort` attribute. The default is `yes`, which sorts the entries when the user clicks on the column header.

**Controlling the Type of Data Displayed in Columns**

You can specify the type of data that a new column cell can display. For more information, see the *GDA API Reference*.

**Controlling the Format of Floating Point Numbers in a Column**

All column cells format floating point numbers by providing an instance of a `GXL-FLOAT-FORMATTER` as the value of the `Float-format` attribute of the column. You can customize these aspects of floating point numbers:

- Whether default formatting is used
- The minimum number of characters in the formatted numbers
- The number of significant digits, or the number of decimal places
- How the value is represented
- Whether to display zeros to the right of the last nonzero digit

For more information about this object and its attributes, see the *G2 XL Spreadsheet User's Guide*.

**To configure the float formatter for a column:**

- 1 Click any cell in the column whose float formatter you want to configure and choose `table`.
- 2 Click on the `Float-format` attribute and choose `subtable`.
- 3 Edit the attributes of the subtable according to the table above.

By default, all cells that show floating point numbers remove the trailing zeros and show 4 decimal places.

**Determining Whether a User Can Edit Data in a Column**

You can control whether users can enter data into column cells.

**To specify whether a user can edit values in the column cells:**

- 1 Click any cell in the column you want to control and choose `table`.
- 2 Click on the `Cells are editable` attribute and modify its value. The default is `false`, which means that the end user cannot edit the values in the cells. Editing the value in a cell modifies the current queue entry.

## Controlling Whether a Column Cell Can Be Selected

You can specify whether or not the user can select a cell in the column. The default value is `true`, which means that users can select the cell.

### To specify whether a column cell can be selected:

- 1 Click the cell in the column you want to control and choose `table`.
- 2 Click on the `Cells are selectable` attribute and modify its value.

## Formatting the Contents of a Column

The `Key-value-conversion-procedure` attribute is a procedure that formats the contents of the selected column. This table describes the procedures supplied with GDA for the attribute and indicates which column headers use them as their default:

Procedure	Description
<code>gdl-get-time-attribute-value</code>	<p>This procedure is the default value for the <code>Time</code> column in all queues.</p> <p>Converts a timestamp to a date-time format using the <code>Date-time Format</code> attribute, as described in <a href="#">Configuring Date and Time Formatting</a>.</p>
<code>gqsv-get-attribute-value</code>	<p>This procedure is the default value for the <code>Severity</code> and <code>Alarm Message</code> columns in the <code>Alarm Queue</code>, and the <code>Message Text</code> column in all other queues.</p> <p>Displays the value of the <code>queue entry</code> attribute as text.</p>
<code>gda-format-acknowledged</code>	<p>This procedure is the default value for the <code>Ack (Acknowledged)</code> column in the <code>Alarm Queue</code>.</p> <p>Formats the value of the <code>Acknowledged</code> attribute of a <code>gda-alarm-entry</code>. A value of <code>false</code> causes the <code>Ack</code> column to be blank when an alarm is not acknowledged. A value of <code>true</code> displays <code>Yes</code> in the column.</p>

### To customize the conversion procedure for a column header:

- 1 Define a procedure with the desired behavior that has this signature:

```
my-conversion-procedure  
(item: class item, attribute: symbol)  
-> (value: value)
```

Argument	Description
<i>item</i>	The queue entry whose column header data is to be converted.
<i>attribute</i>	The queue entry attribute whose data is to be converted.

Return Value	Description
<u><i>value</i></u>	The queue entry attribute data as it is to appear in the queue.

- 2 Specify your procedure as the **Key-value-conversion-procedure** when configuring attributes for any column header in any default queue view.

## Configuring Date and Time Formatting

You can configure the date and time formats used to display timestamps.

By default, all queues format date and time attributes as follows:

Attribute	Format	Example
Date-Time Format	<m2>/<d2>/<y4> <h24>:<min>:<sec>	09/09/1999 16:23:50
Date Format	<m2>/<d2>/<y4>	09/09/1999
Time Format	<h24>:<min>:<sec>	16:23:50
File-Time Format	<y2><m2><d2>- <h24><min><sec>	990909-163323

GDA uses the date-time format when:

- Displaying these queue entry attributes in the queue:
  - Gqm-creation-time
  - Gda-acknowledge-time
- Writing queue entry data to a file, for example:
  - Severity: 1 At: 09/09/1999 10:23:50
  - CT: 09/09/1999 10:23:50

GDA uses the date format and time format when writing queue entries to a file. For example:

Acknowledged on 09/09/1999 at 17:25:38  
----- History -----  
Entered alarm on 09/09/1999 at 17:25:33  
Returned from alarm on 09/09/1999 at 17:25:55

GDA uses the file-time format for the default file name when saving queue entry data and detail view data to a file, for example:

save-selected-990909-173323.text

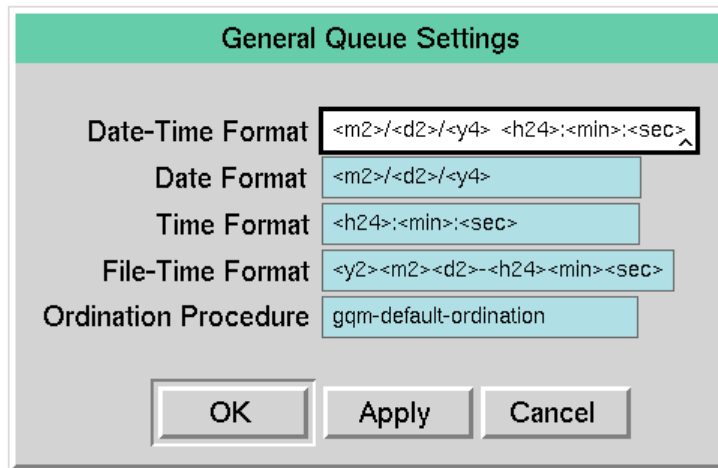
This table lists and describes all date and time formats:

<b>Format</b>	<b>Example</b>
<d>	Day
<d2>	2-digit day
<df>	Full day name
<da>	Abbreviated day name
<dor>	Ordinal day
<m>	Month
<m2>	2-digit month
<mf>	Full month name
<ma>	Abbreviated month name
<y2>	2-digit year
<y4>	4-digit year
<apm>	A.M. or P.M.
<h12>	Hour (12-hour clock)
<h24>	Hour (24-hour clock)
<min>	Minute
<sec>	Second

When specifying formats, the character that separates formats appears in the displayed date/time. For example, <m2>/<y4> might display 12/1999.

**To configure the date and time formats:**

- 1 Choose Preferences > Queues > General Settings to display this dialog:



- 2 Configure the date and time attributes as described in the tables above.

**Configuring Ordination**

By default, the queues format numbers as ordinal text by using the `gqm-default-ordination` procedure, which results in this ordinal text:

1st, 2nd, 3rd, 4th, 5th, 6th, 7th, etc.

You can configure the procedure that gets used to format ordinals, for example, to support internationalization.

**To configure the ordination:**

- ➔ Configure the Ordination Procedure attribute to name a custom procedure.

You can provide a custom procedure that gets called whenever the queue needs to format an integer as ordinal text, for example, 1st, 2nd, 3rd, 4th, 5th, etc.

**To customize the ordination procedure for the overall environment:**

- 1 Define a procedure with the desired behavior that has this signature:

**my-ordination-procedure**

(*number*: integer)

-> (*ordinal*: text)

<b>Argument</b>	<b>Description</b>
<i>number</i>	The integer to use as the base of the ordinal text.

<b>Return Value</b>	<b>Description</b>
<u><i>ordinal</i></u>	The ordinal text.

- 2 Specify your procedure as the Ordination-procedure when configuring attributes for the overall environment. Choose Preferences > Queues > General Settings to access the General Queue Settings dialog box.



## Default Alarm Queue Color Formatter Procedures

The `Dynamic-color-formatter` attribute controls the colors of particular columns in a queue view for an Alarm Queue. This table summarizes the procedures supplied with GDA for the `Dynamic-color-formatter` attribute and describes which column headers use them as their default:

Procedure	Description
<code>gda-color-entry-by-severity</code>	<p>This procedure is the default value for the Time, Severity, and Alarm Message columns in the Alarm Queue. The value of this attribute for columns in all other queues is <code>unspecified</code>.</p> <p>When an alarm entry is active, this procedure colors the cells of the column according to the severity; otherwise, it colors the cells using the default colors.</p> <p>When an alarm entry is not active, this procedure colors the cells <code>white</code>.</p> <p>You can change the color settings by choosing Preferences &gt; Colors &gt; Alarms and selecting colors on the dialog box.</p>
<code>gda-color-entry-by-acknowledgement</code>	<p>This procedure is the default value for the Acknowledged column in the Alarm Queue.</p> <p>When the alarm has not been acknowledged, this procedure colors the cells <code>cyan</code>. If the alarm has been acknowledged, it colors the cells using the default colors.</p>

### To customize the color formatter procedure for a column header:

- 1 Define a procedure with the desired behavior that has this signature:

#### **my-color-formatter-procedure**

```
(entry: class gda-alarm-entry, attribute: symbol)
-> (background-color: symbol,
    border-color: symbol,
    text-color: symbol)
```

Argument	Description
<i>entry</i>	The queue entry whose cell color is to be formatted.
<i>attribute</i>	The queue entry attribute whose cell color is to be formatted.

Return Value	Description
<u><i>background-color</i></u>	The background color of the cell.
<u><i>border-color</i></u>	The border color of the cell.
<u><i>text-color</i></u>	The text color of the cell.

- 2 Specify your procedure as the `Dynamic-color-formatter` when configuring attributes for any column header in any default queue view.

## Modifying Column Headers

You can configure various characteristics of column headers, including the header label text, font size, or colors, to modify characteristics of the column headers and of the data contained in the columns. You can also specify whether a column value is to be monitored.

### Specifying the Attribute to Display in the Column

**To specify the attribute to display in the column:**

- 1 Click on the header you want to modify and choose `table`.
- 2 Choose `Attribute-or-key` and specify the attribute name. The default value is unspecified.

For a list of available queue entry attributes, see the tables in [Creating a Temporary Filter](#).

### Modifying the Header Label Text

**To modify the text of the header label:**

- 1 Click on the header you want to modify and choose `table`.
- 2 Choose `Visible-label` and edit the label.

## Modifying the Font Size of the Label

**To modify the font size of the label:**

- 1 Click on the header you want to modify and choose **table**.
- 2 Choose **Font-size** and edit the label.

## Modifying Colors of the Label

You can modify the color of the label text, background, and borders.

**To modify the colors of the label:**

- 1 Click on the header you want to modify and choose **table**.
- 2 Choose **Default background color**, **Default text color**, or **Default border color** and edit the color by entering the name of the color.

To obtain a list of available colors, click the background of the view and choose **Color > background-color**, then specify one of these colors as a symbol, for example, **forest-green**. Colors made up of more than one word use hyphens between the words.

## Monitoring a Column Value

You can write a procedure that monitors the value in a column. Each column header has an attribute that specifies whether the value of the column is to be monitored. For new columns added to the view layout, the value of this attribute is **yes**. In general, the value should be **yes** only for columns whose values can change.

**To indicate that a column value is to be monitored:**

- 1 Click on the header you want to modify and choose **table**.
- 2 The value of the **Monitor-this-attribute** attribute should be **yes**.

For more information, including how to indicate the name of the procedure that monitors the attribute value, see the *GDA API Reference*.

## Configuring the Detail View

The detail view of a queue appears when you select a queue entry and click the Show Details button on the queue view, as described in [Viewing the Details of an Entry](#).

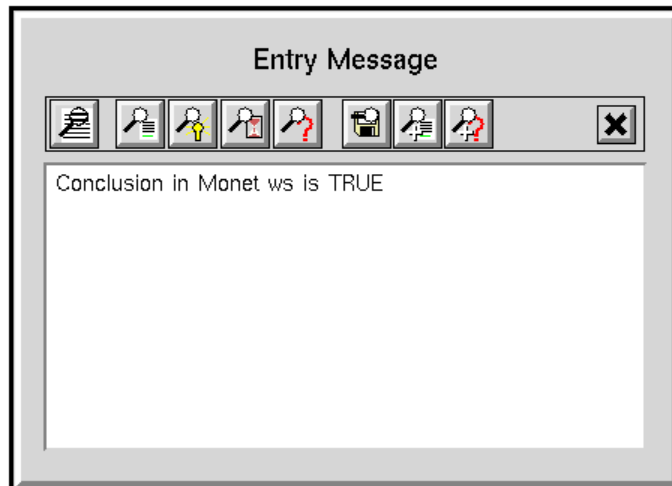
You can configure these aspects of the detail view:

- The default colors and size of the display area
- The default location and colors of the label
- The default position and scale when the detail view is launched

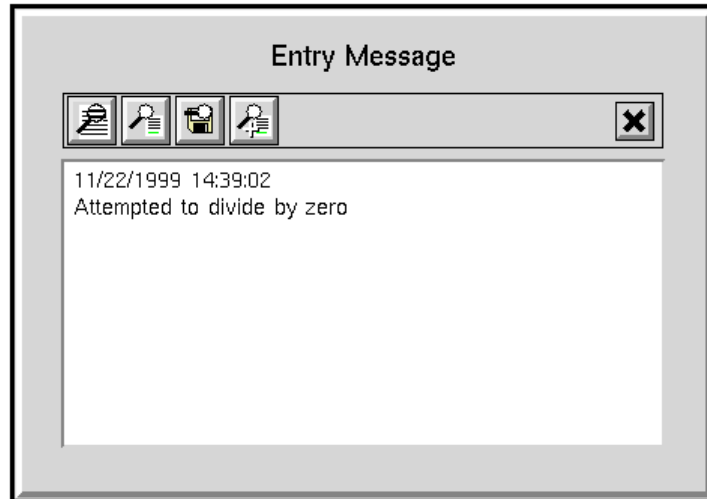
You cannot configure the buttons of the detail view.

## Configuring the Detail View Template

The default detail view template for the Alarm Queue is `gda-alarm-detail-view-template`, which generates a detail view that looks like this:



The default detail view template for the Error, Explanation, and Message Queues is `gqmv-default-detail-view-template`, which generates a detail view that looks like this (this figure shows detail for an error entry):



The detail view template is specified as the **Detail view template** attribute of the View Details button. For more information about specifying this attribute, see [Modifying Button Attributes](#).

You can configure these aspects of the default detail view templates:

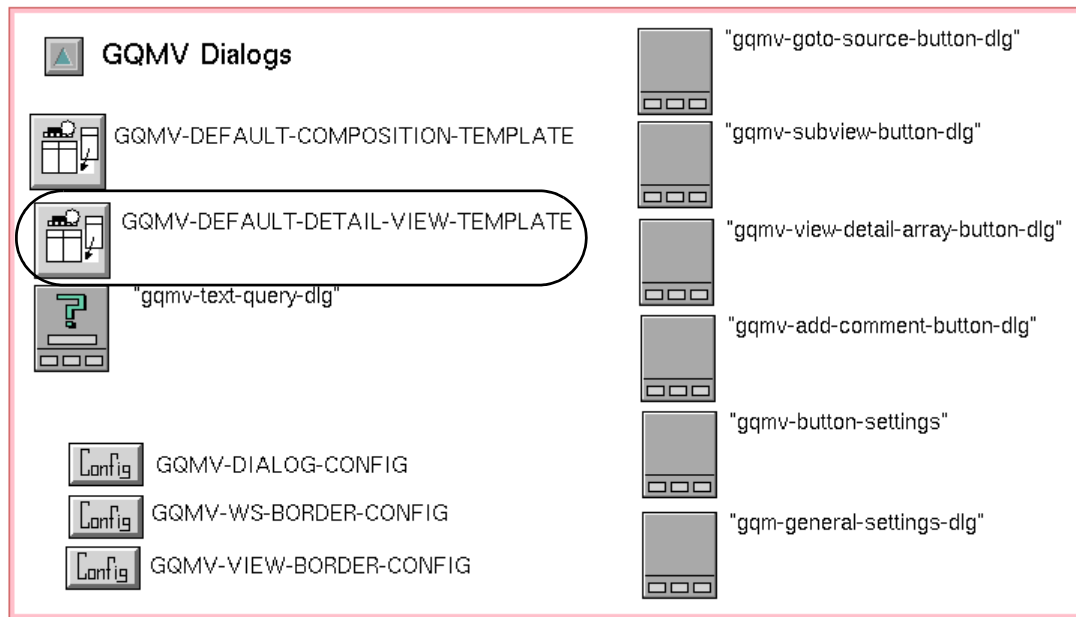
- The colors
- The size
- The label

To configure the detail view templates, you must be in Administrator mode.

## Displaying the Default Detail View Templates

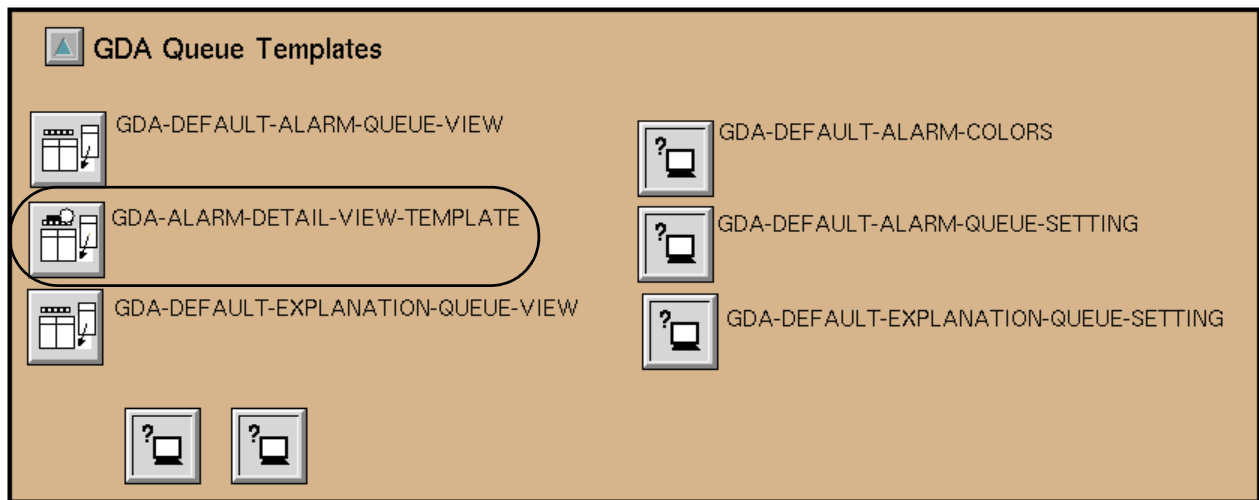
To display the default detail view templates:

- 1 Make sure you're in Administrator mode.
- 2 To display the default detail view template for the Error Queue, the Explanation Queue, or the Message Queue, display the GQM Views detail view template:
  - a Choose Main Menu > Get Workspace > GQMV-TOP-LEVEL.
  - b Click the GQM Views Dialogs button to display this workspace:



- 3 Or, to display the default detail view template for the Alarm Queue:
  - a Choose Main Menu > Get Workspace > GDA-4.0. You can also use Inspect to go to the Alarm Queue.
  - b Click the GDA Queues button and choose go to subworkspace.

- c Click the Templates and Settings button to display this workspace:



- 4 For both templates, click the detail view template object to display its menu and choose **go to subworkspace**.

You configure the detail view template colors, size, and label from this subworkspace.

### Configuring the Detail View Colors

The background color of the detail view controls the light-gray area of the view. The foreground color controls the color of the view label (“Entry Details”) unless the label color has been explicitly specified (its default text color is initially defined as `foreground-color`).

#### To configure the detail view colors:

- 1 Click the grey background of the detail view and choose **Color**.
- 2 To modify the background color, choose `background-color` to display a palette of colors from which to choose, then click a color.
- 3 To modify the foreground color, choose `foreground-color` and click a color.
- 4 To modify the detail area colors, click the white area of the detail and choose **color**. Choose `background-color`, `border-color`, and/or `text-color` from the menu to configure these colors in the same way. Do not configure the border color because the border does not appear on the detail view.

## Configuring the Detail View Size

You can change the size of the detail area, used to show the text of the detail.

### To configure detail view size:

- 1 Click the white area of the detail and choose **change min size**.
- 2 Drag the edges of the black rectangle that appears to adjust the size of the view, then click the Yes button.
- 3 Resize the detail view workspace by dragging the outside border corners.

## Configuring the Detail View Label

You can modify the colors and position of the detail view label. The label is initially “Entry Details,” but GDA changes its text depending on what it shows. You can edit the detail view label text directly or by looking up the value in a local text resource.

### To configure the detail view label:

- 1 To modify the label colors, click the (invisible) border around the label and choose **color**. You may need to try this several times to find the border. Choose **background-color**, **border-color**, and/or **text-color** from the menu to configure these colors of the label.
- 2 To modify the label location, drag the label to a new location.
- 3 To modify the label text, click the text and enter a new text value.

You can also localize the detail view label text by configuring a symbol to use for looking up the label text in a local text resource.

- 4 To specify a label symbol, click the border around the label to display its menu, choose **table**, and configure the `gqmv-label-symbol` attribute to specify a label symbol.

The label symbol points to entries in the `gqmv-local-text-resources` text resource. You can edit the text of the existing text resource or create new entries and have the template’s label point to that.

## Creating Your Own Detail View Template

You can create your own detail view template item to display different detail views for the Error, Explanation, or Message Queues. You can also create your own detail view template if you create a custom queue, as described in [Creating a New Queue](#).



**To create your own detail view template:**

- 1 Follows steps 1 or 2 under [Displaying the Default Detail View Templates](#) to display the template you want to clone.
- 2 To clone the template, click the item and choose **clone** to attach it to the mouse, then click on a workspace to place it.
- 3 Choose **go to subworkspace** on the cloned item and configure the detail view as described in these sections:
  - [Configuring the Detail View Colors](#)
  - [Configuring the Detail View Size](#)
  - [Configuring the Detail View Label](#)

**Associating the Detail View with the View Details Button**

The last step in creating a detail view is connecting the view with the View Details toolbar button. To make this connection, you modify the Detail View Template attribute of the button. For more information, see the table in [Modifying Button Attributes](#).

**Configuring the Detail View Position and Scale**

By default, the detail view is displayed in the middle of the window at full scale.

You can specify the position of the detail view workspace relative to the window by configuring these attributes:

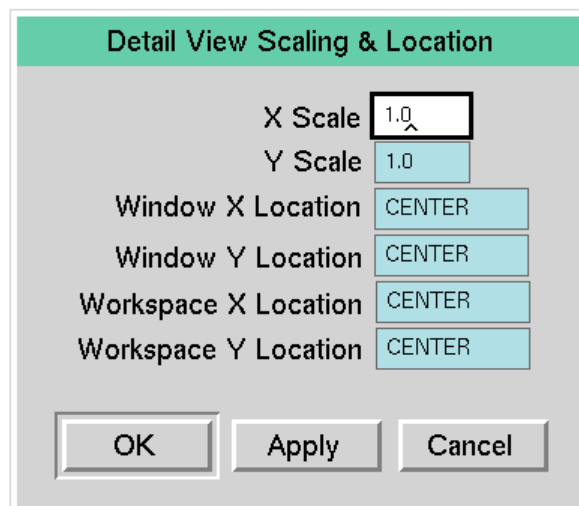
Attribute	Description
Window X Location	The X position of the window with respect to which the detail view is positioned. The options are: <b>center</b> , <b>right</b> , <b>left</b> , or the number of pixels, as an integer.
Window Y Location	The Y position of the window with respect to which the detail view is positioned. The options are: <b>center</b> , <b>top</b> , <b>bottom</b> , or the number of pixels, as an integer.
Workspace X Location	The X position of the workspace with respect to the window position. The options are: <b>center</b> , <b>right</b> , and <b>left</b> .
Workspace Y Location	The Y position of the workspace with respect to the window position. The options are: <b>center</b> , <b>top</b> , and <b>bottom</b> .

You can configure the scale of the detail view by configuring these attributes:

Attribute	Description
X Scale	The scale of the detail view along the X dimension. For example, to display the detail at half scale horizontally, specify 0.5.
Y Scale	The scale of the detail view along the Y dimension. For example, to display the detail at half scale vertically, specify 0.5.

**To configure the position and scale of the detail view:**

- 1 Choose Preferences > Queues > Buttons to display this workspace:



- 2 To specify the position of the detail view, configure the Window and Workspace X and Y Location attributes according to the table above.
- 3 To specify the scale of the detail view, configure the X Scale and Y Scale attributes according to the table above.

The attributes take effect the next time you display a detail view.

This table provides a number of common combinations of the location attributes:

To position the detail view in the...	Configure the attributes like this...
Top-left corner of the window	Window X Location = left Window Y Location = top Workspace X Location = right Workspace Y Location = bottom
Top-right corner of the window	Window X Location = right Window Y Location = top Workspace X Location = left Workspace Y Location = bottom
Bottom-left corner of the window	Window X Location = left Window Y Location = bottom Workspace X Location = right Workspace Y Location = top
Bottom-right corner of the window	Window X Location = right Window Y Location = bottom Workspace X Location = left Workspace Y Location = top
Top-center of the window	Window X Location = center Window Y Location = top Workspace X Location = center Workspace Y Location = bottom
Left-center of the window	Window X Location = left Window Y Location = center Workspace X Location = right Workspace Y Location = center

## Creating and Configuring the Access Manager

An **access manager** specifies the template that defines the queue view for a named user or window class accessing the application in a particular user mode.

If the application displays the contents of a queue using only one template, it is not necessary to use an access manager. If, however, the application provides more than one view for a particular queue, it is necessary to create and configure an access table to indicate which users view the queue using which view templates.

By specifying different combinations of user names or window classes, user modes, and queue view templates, you can control how every application user

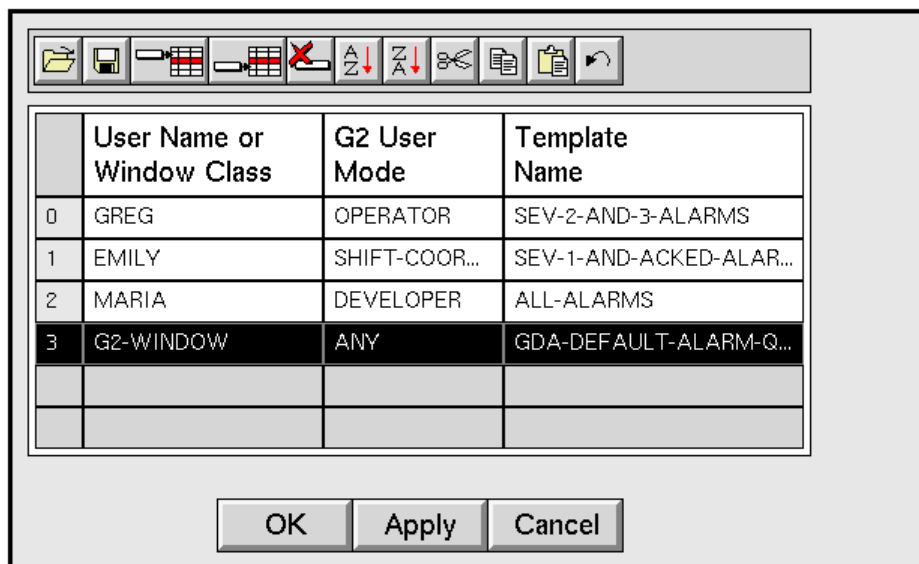
accesses the queues. The access table and the ability to define multiple queue views for a single queue enable you to manage access to the contents of all queues, presenting to each user or category of user only the needed information.

For information on configuring a view template, see [Configuring a Queue View Template](#).

## How the Access Manager Works

The access manager contains a sequenced list that associates a queue view template with either a specified user name or window class, accessing the application in a specified access mode.

This sample access table associates a queue view template with three different users and a default template for all others:



	User Name or Window Class	G2 User Mode	Template Name
0	GREG	OPERATOR	SEV-2-AND-3-ALARMS
1	EMILY	SHIFT-COOR...	SEV-1-AND-ACKED-ALAR...
2	MARIA	DEVELOPER	ALL-ALARMS
3	G2-WINDOW	ANY	GDA-DEFAULT-ALARM-Q...

The access manager determines which queue view template to use by following this sequence of steps:

- 1 First, the access manager compares the name of the currently logged in user with the user names in its list, starting with the user name specified in the top row in the list.
- 2 If the currently logged in user appears in the list, the access manager then compares the user mode for that user with the user mode for that user's entry. If the user is logged in to GDA in the user mode specified in the row for that user, then the access manager uses the view template specified for the user.
- 3 If the currently logged in user does not appear in the list or if the user is logged in to GDA in a different user mode (and no entry for that user mode is

found in the table), then the access manager compares the current window with the list of window classes in the table.

- 4 If the currently logged in user's window appears in the list and the user is logged in to GDA in the specified user mode, then the access manager uses the template associated with this window.
- 5 If neither the user name nor the window class is found in the list, or if the user mode is not satisfied for the current window, then the access manager signals an error. (In the sample access table, the last row associates a view template with all users not specifically addressed in the previous rows.)

Using the access table shown above, suppose the user *maria* logs in to GDA in developer mode. The access manager looks for the user name *maria*, which it finds. It then checks to see whether the user mode matches the mode *maria* is accessing GDA. Because the modes match, it uses the **all-alarms** view template.

Suppose the user *natasha* logs in to GDA in developer mode. Because the access manager does not include an entry for *natasha*, it uses the default alarm queue template (specified in row 3), which associates the **gda-default-alarm-queue-template** for all users not addressed by the other permission entries.

## Specifying the Queue View Template or Access Manager

Each queue has an attribute that specifies the queue view or access manager that determines the queue view that presents the queue entries to application users.

By default, the value of this attribute is the built-in queue view template for the queue. If you create a new template to provide an alternate queue view, you should change the value of this attribute to be the new template name. If you create more than one view for a particular queue and create an access manager to associate users with queue views, you should change the value of the attribute to be the access manager name.

This table indicates the default queue view template for each queue type:

Queue Type	Default Template
Alarm Queue	<b>gda-default-alarm-queue-view</b>
Error Queue	<b>gdabasic-default-error-queue-view</b>
Explanation Queue	<b>gda-default-explanation-queue-view</b>
Message Queue	<b>gdabasic-default-message-queue-view</b>

In addition, the default Message Queue access manager allows users logging in to any **UI-CLIENT-ITEM**, through either a classic Telewindows or Telewindows2 Toolkit

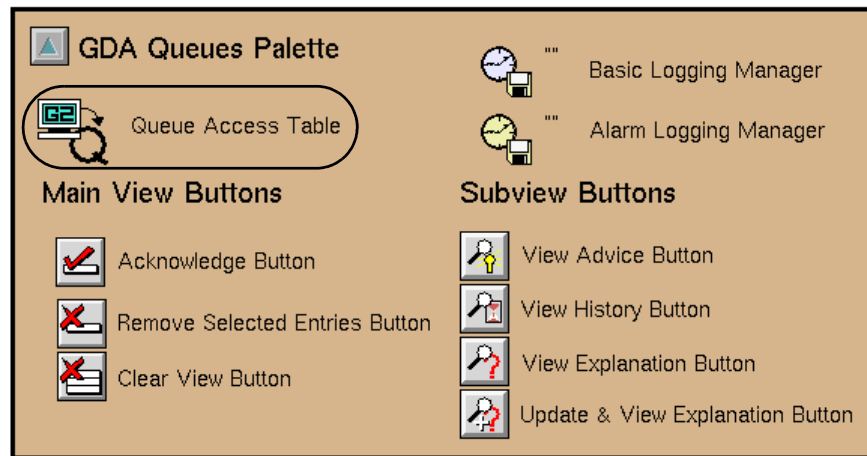
client, in any user mode, to access the Message Queue. When viewing queues through a Telewindows2 Toolkit client, you must specify the default template to use.

## Creating an Access Manager

To define how users access a queue using one of the queue view templates defined for the application, you must create an access manager.

### To create an access manager and associate it with a queue:

- 1 Using Inspect, go to alarm-queue.
- 2 Click the GDA Queue Palette button to display this workspace:



- 3 Click the Queue Access Table item to attach a new instance to the cursor, then move the item into a workspace.
- 4 Click the new instance and choose **name** to name the new access manager.
- 5 Click the queue item you want to associate with the new access manager and choose **table**. This queue can be one of the built-in queues or a new queue.

For example, to associate **my-access-table** with a new Alarm Queue you created, called **my-alarm-queue**, display the table for that queue.

- 6 Edit the `Gqs-view-template-or-access-table` attribute value to be the name of the new access manager.

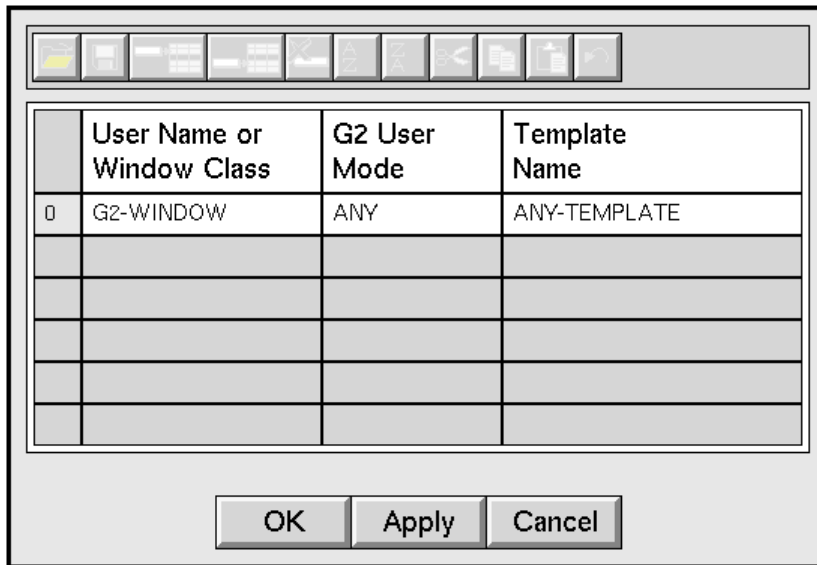
## Configuring an Access Manager



You can configure a new access manager to:

- Specify the default template used to create the queue view for users logging in to GDA from any G2 window in any user mode.
- Define additional entries in the access table to specify custom templates used to create the queue view for different users or users accessing the application in different user modes.

### To configure an access manager:

- 1 Display the workspace that contains the access manager item.
- 2 Click the access table, then choose **configure access table**. This is the default access table:



- 3 To edit the default settings, click the cell to edit, edit its value, then press Enter.
- 4 To delete the default settings, select the row by clicking on its left-most column, then click the Delete Selected Rows button .
- 5 To add a new entry after a row in the table, select the row, click the Insert Row After Selection button  to add a new row to the spreadsheet, then edit the cell values.

The order in which the rows appear is significant. When the access manager uses the table to determine which template to use for the queue view, it checks entries starting with the first row (row 0), moving through the rows until the last entry.

## The Access Manager Toolbar Buttons

The access manager provides these toolbar buttons:



This table describes how the toolbar buttons modify the access table:

This toolbar button...	Performs this function...
Load from File into Selection	Replaces the selected rows of the table with the contents of the file. The default file name is the file name most recently used with the Save selection to file button.
Save Selection to File	Saves the selected rows to a file. A dialog requests the file name. The default directory is the directory from which the application is run.
Insert Row Above Selection	Inserts an empty row before the selected row. Use this button to create a new row before the selected row.
Insert Row After Selection	Inserts an empty row after the selected row. Use this button to create a new row after the selected row.
Delete Selected Rows	Deletes the selected rows.
Sort Rows of Selection in Ascending Order	Sorts the selected rows in ascending order. The sort column is the User Name or Window Class column.
Sort Rows of Selection in Descending Order	Sorts the selected rows in descending order. The sort column is the User Name or Window Class column.
Cut Selection	Deletes the selected rows. Use the Paste contents of clipboard button to paste them into the table.
Copy Selection to Clipboard	Copies the selected rows. Use the Paste contents of clipboard button to paste them into the table.



<b>This toolbar button...</b>	<b>Performs this function...</b>
Paste Contents of Clipboard	Pastes the deleted or copied rows into the table, replacing the currently selected rows.
Undo the Last Operation	Returns the table to its state before the previous operation. You can only undo the last operation.



A B C D E F G H I J K L M  
N O P Q R S T U V W X Y Z

---

## A

**action block:** Performs actions on other blocks or on the environment when an inference value becomes true.

**action link:** Serves as a pointer when a block is to perform an action on a target object. For example, you can reset a block by using a control signal by attaching any block to the action link associated with the Reset block. When the block receives a control signal, GDA resets the block. Action links appear on many blocks on the Action menu.

**alarm:** Indicates the source of potential problems in the real-time data that the diagram is monitoring. GDA displays alarms in the Alarm Queue.

**API procedure:** When you are creating custom subclasses of custom blocks, statements in the block evaluator procedure call Application Programmers Interface (API) procedures that:

- Set values onto a path and trigger the evaluation of downstream blocks.
- Set values onto a path without triggering the evaluation of downstream blocks.
- Obtain values from a path.
- Resolve output path attributes for custom blocks.

**attributes:** Specify the particular behavior of a block. You specify block attributes in the configuration panel.

## B

**block evaluator:** Executes when a custom block receives a value on its input path(s). GDA supplies a template for the block evaluator, based on the input and output stubs and the type of block. You specify in the custom portion of the procedure.

**block menu:** Enables you to perform G2 operations on blocks, such as cloning, transferring, deleting, and configuring.

## C

**capability link:** Adds various types of features to blocks, such as charts, graphs, and clocks.

**configuration panel:** A dialog that enables you to specify the attributes of a block.

**configure:** To specify the attributes of a block in a configuration panel or in the attribute display of a block.

**connection post:** Enables a block on one workspace to pass data to a block on another workspace.

**control path:** A type of path that passes control signals, which cause downstream blocks to execute.

**custom block:** A subclass of GDA block that executes custom G2 procedures. There are three basic types of custom blocks:

- General - performs calculations on single or multiple inputs when the block needs to reference specific inputs by port name.
- Peer Input - performs calculations on single or multiple inputs when the block treats all inputs equally.
- Multiple Invocations - enables control over how the block processes simultaneous control signals.

**Custom Class Wizard:** Creates new subclasses of GDA blocks and edits existing subclasses. Each instance of a custom subclass inherits the definition of an existing GDA custom class or a custom block subclass.

## D

**data block:** Operates on numeric, textual, or symbolic values.

**data path:** An input or output path to a block that contains numeric, textual, or symbolic values.

**description:** An attribute of certain inference blocks that enables you to describe the current output value and provide advice to operators. For example, if the inference block passes `.true`, the description might be `the temperature is too high`.

**developer mode:** A user mode that provides access to all the basic functionality required for building schematic diagrams.

**disable evaluation:** Stops a block from passing its output value and responding to new values. Disabling evaluation enables you to “turn off” entire portions of a GDA diagram.

**discrete logic:** A form of inferencing whereby a block passes discrete inference values, for example, a `Status-value` of `.true`, `.false`, or `unknown`, and a `Belief-value` of 1.0, 0.0, or 0.5. *See also* fuzzy logic.

## E

**enable data input:** To run a diagram, you must enable data input, which:

- Starts data flowing into entry points.
- Evaluates signal generators.
- Evaluates clock capabilities.

**enable evaluation:** Allows a block to pass its output value and respond to new values.

**encapsulation block:** Enables you to hide complexity in a GDA diagram by placing a portion of the diagram on a subworkspace. There are two types:

- Single Source Encapsulation blocks enable you to create multiple instances of the encapsulation block when you need to propagate changes in the definition of the encapsulation to other blocks in the diagram.
- Simple Encapsulation blocks are the same except that you cannot propagate the changes in the definition to other blocks in the diagram.

**entry point:** Receives data externally from a variable, a GSI (G2 Standard Interface) variable, G2 procedure, or from an embedded variable in the table for the block. Entry points are the starting point of a GDA diagram. GDA supports five kinds of entry points: numeric, text, symbolic, belief, and control.

**evaluate:** To execute the procedure for a block. For example, when you evaluate an entry point, the current value is propagated with a new timestamp. When you evaluate a block with a single input control path, the block acts as if it has received a new control signal.

**explanation:** Provides a description of the combination of upstream blocks that resulted in the current output inference value of certain blocks.

## F

**filter:** A category of block that you use after data entry blocks in a diagram to filter out noise and find trends in data. GDA supports six kinds of filters: changeband, outlier, first-order exponential, nonlinear exponential, quadratic, and cubic.

**fuzzy logic:** A form of inferencing whereby a block passes a range of values, for example, a Status-value of `.true`, `.false`, or `unknown`, and a Belief-value that is a number between 0.0 and 1.0, where 0.0 is completely false and 1.0 is completely true.

## G

**G2 Main Menu:** Controls whether G2 is running or paused, and allows you to load and save applications.

**G2 menus:** Provide all the functionality of G2 within the GDA environment, for example, the creation of class definitions, variables, parameters, rules, and procedures.

## H

**history:** A store of past input values. Numerous GDA blocks operate on the stored values, such as computing the average of the last 25 input values.

**HTML:** Hypertext Markup Language. Online documentation is a collection of HTML files, which you can display in any HTML browser.

**hysteresis:** Passes the previous value in a condition block if the current changes are small. For example, if the belief value of a block whose Output Uncertainty is 0.5 changes from 0.8 (.true) to 0.7 (unknown), the block will continue to pass .true if the Hysteresis When attribute is set to .true.

## I

**inference block:** Translates data values to truth values and operates on truth values, including fuzzy truth values. For example, observations observe data values and pass inference values, and logic gates use Boolean logic to combine reference values.

**inference path:** A type of path that carries truth values. Inference paths carry two values:

- Belief value - a number between 0.0 to 1.0, where 0.0 is completely false and 1.0 is completely true.
- Status value - one of the symbols .true, .false, or unknown. GDA derives status values from belief values.

**initial value:** The value a block passes when you first start G2 or when you reset the block.

**input port:** Carries data to a block. A block has one or more input ports depending on the type of block.

**invoke:** To execute the procedure for a block. GDA invokes a block when:

- An entry point receives a value from its data source.
- A value is propagated onto the input path of the block due to the behavior of an upstream block.
- You evaluate a block manually.

**item path:** A type of path that passes any G2 item through the block. Use item paths with custom blocks for:

- Discrete event processing, for example, processing individual items in an assembly line.
- Complex data processing, for example, processing multiple items, using an item list or item array.

## K

**KB Workspace menu:** Enables you to create G2 definitions and objects on a workspace and set up applications.

## L

**link:** A special-purpose type of connection that you use to add features or behaviors to a block, or to perform actions on a block. For example, you use links to add a graph or chart capability to a block.

There are three types of links:

- Action
- Capability
- Restriction

**lock:** When locked, a block does not respond to input data or pass its output values. GDA locks a block when you manually override its value.

## M

**message:** GDA sends messages to the Message Queue when you use the Send Message block.

**multiple values:** Simultaneous control signals that a block receives. A block can ignore or use multiple values as needed.

## N

**no-value inputs:** Occur when stubs are unattached or when connected paths never receive a value. Peer input data blocks and peer input logic blocks ignore input paths with a **Quality of no-value**. Non-peer input blocks require all of their inputs to evaluate, and, therefore, never place a value onto an output path if the block has a no-value input.

## O

**observation:** A category of blocks that detect features in your data. Observation blocks take data as input, test it against a threshold, and pass as output the inference value that the test produced.

**output port:** Carries data from a block. A block has one or more output ports depending on the type of block.

**override:** To manually change a block's output value for testing purposes. Overriding a block locks the block and propagates a **Quality of manual** onto the output path.

## P

**parameter:** A G2 object that stores a data value and keeps a history of it over a specified time. A parameter can also initiate forward chaining.

**path:** The connection between two blocks. GDA supports data, inference, control, and item paths.

**path attributes:** Attributes that provide information about the value and status of the data on one path. Each type of path defines slightly different path attributes.

**path quality:** A path attribute that specifies the status of a path's data. There are three types:

- Manual
- No-value
- Expired

**path splitter:** Connects the input stub from one block to the path between two other blocks so that more than one downstream block can get input from the same upstream block.

**peer input block:** A category of block that can have any number of inputs and does not evaluate the inputs in a specific order. The inputs are treated all alike and are therefore peers.



**port:** Connection stub to which another stub can attach. Ports are either named or unnamed, depending on the type of block and whether the port is input or output.

## Q

**queue:** A special workspace that displays information about alarms, explanations, messages, and errors.

## R

**reset:** Causes the following to happen:

- Sets the block to its initial state, which propagates the block's initial value.
- Clears any error conditions.
- Unlocks the block, if it was locked.
- Erases the blocks history if the block maintains a history.

**restriction link:** Enables you to customize a block's capabilities. You can customize the override dialog and determine the source of input for a GDA diagram.

## S

**signal generator:** Generates a continuous signal to a diagram, to simulate real-time data. Examples of signal generators include the Sine Wave signal and White Noise signal.

**snapshot:** A file that contains the current state of the running application as backup. You can configure GDA to take snapshots automatically at regular intervals. When you restore a snapshot, GDA resumes running the application from the point at which you took the snapshot.

**Statistical Process Control (SPC):** A category of block that uses statistical methods to measure the quality and consistency of a process. The blocks in the SPC palette contain statistical tests, pattern recognition techniques, and graphs. An application that uses SPC monitors a process and compares current performance with expected performance.

**stub:** A connection port to which another connection port can attach.

**sweep:** An internal mechanism where GDA searches for invoked blocks, evaluates them, and continues until no more blocks are left to evaluate.

**system administrator:** A category of GDA users who works in Administrator mode, which enables access to additional attributes and menu choices used for debugging.

## T

**temporal logic:** A type of inferencing that allows you to analyze the timing of events.

**top menu bar:** Provides access to basic functionality, including controlling the diagram, cloning blocks from palettes, accessing queues, and customizing the environment.

## U

**uncertainty:** Defines a band around 0.5 that determines the status value **unknown**. For example, if the attribute Output Uncertainty is 0.25, then the **Belief-value** is **.true** above 0.625 and **.false** below 0.375 and **Unknown** between .65 and .35.

**user modes:** There are four user modes:

- Administrator - enables system administrators to access attributes and menu choices used for debugging.
- Developer - enables developers to access all the basic functionality used for building schematic diagrams.
- User and Browser - enables end users to view diagrams, display menus, and display configuration panels of objects but not to move blocks, clone blocks, or edit attributes. Browser mode is slightly more restrictive than User mode.

## V

**variable:** You use variables as starting points in a GDA diagram, either by referring to the variable in an entry point, or by connecting blocks directly to the variable. *See also* parameter.

**vertex:** An 90° bend in the connection between blocks.

@	A	B	C	D	E	F	G	H	I	J	K	L	M
#	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

---

**A**

- access manager
  - configuring
  - creating
  - definition
  - example
  - how it works
  - specifying
  - toolbar buttons
- accessing view templates
- Ack column, color
- Ack column, formatting contents
- Acknowledge button class
- Acknowledge button, description
- acknowledged attribute and filters
- acknowledge-selected-entries attribute
  - modifying
  - of Remove Entries button
- acknowledging alarms
- acknowledging alarms, example
- acquiring data
- action button, making temporary filters
  - permanent
- action link
- actions
- Activate Filter button class
- Add Comments button class
- Add Subworkspace Button menu item
- adding
  - columns to queue view
  - comment to queue entry
  - comments to blocks
  - custom connections to custom subclass
  - queue entry to queue, callback
  - toolbar buttons
- administrator mode
- advice
  - definition
  - example with detail view
  - providing for alarms
  - viewing for alarms
- advice attribute, evaluating expressions in
- Alarm Colors dialog
- alarm entries
  - expired
  - incremental logging of
  - time format of in log
- alarm log entries
- Alarm Message column, color
- Alarm Message column, displaying value
- Alarm Panels, customizing colors of
- Alarm Queue
  - built-in view template
  - default queue view
  - detail view
  - example with selected entry
  - history
  - locking view
  - sorting, default sort order
  - using
  - workspace containing built-in
- alarm-log-formatter attribute
- alarms
  - acknowledging
  - advice, providing
  - advice, viewing
  - colors, customizing
  - explanation, viewing
  - tracebacks
- Alarms menu item
  - network object default colors
- aligning blocks to grid
- allow other processing statement
- allow-scheduled-drawing? parameter
- Animate menu item
- animation
  - disabling on startup
  - togglng
- animation delay
- appearance of diagram, improving
- applications
  - animating
  - phases
  - planning
  - running
  - saving periodically

- top-level modules, renaming
- workspaces, creating
- Apply button
- apply changes locally menu choice
- Apply Filter button, description
- applying filters
- Asynchronous evaluation attribute
- attribute displays
- Attribute Editor dialog box
- Attribute Filter item
- attribute to display in column
- attribute-or-key attribute
- attributes
  - blocks
    - configuring using attribute displays
    - configuring using configuration dialogs
    - defining for custom blocks
    - evaluating expressions in
    - paths
- attribute-update-callback attribute
- autogenerate explanations attribute
  - configuring on queue
  - viewing explanations
- automatic explanation attribute and viewing explanations

## B

- backward-compatibility-features parameter
- beep-for-new-entry attribute
- belief-value path attribute
  - determining, using fuzzy logic
  - in inference paths
- Block & Path Colors dialog, block colors
- Block & Path Colors dialog, path colors
- block evaluation engine
- block evaluator, editing
- blocks
  - aligning with snap grid
  - attributes
  - cloning from palettes
  - colors and states
  - colors, customizing
  - commenting
  - connecting
    - general
    - peer input blocks
    - rules for
    - using path splitters
    - using vertices

- customizing block evaluators
- deleting
- disconnecting paths between
- errors, clearing
- errors, viewing in table
- evaluation, enabling and disabling
- evaluation, manual
- inhibiting data flow through
- invoking
- locking manually
- naming
- notes, viewing in table
- number of, in allow other processing
  - statement
- overriding manually
- overview of
- resetting manually
- searching for
- tables for
  - unlocking manually
- Blocks & Paths menu item
  - customizing block colors
  - customizing path colors
- broadcast attribute menu choice
- building diagram
- button classes
  - detail view
  - queue view

## C

- call function, evaluating in attributes
- call procedure, evaluating in attributes
- callbacks
  - attribute update
  - item addition
  - item removal
- Cancel button
- cancel diagram editing menu choice
- capability link
- cause of error or alarm
- cells-are-selectable attribute
- changing modes
- chart name attribute, evaluating expressions in
- classes of detail view buttons
- classes of queue view buttons
- clear error menu item for blocks
- Clear View button class
- Clear View button, description
- clearing block errors

- clone menu item
- cloning
  - blocks from palettes
  - custom blocks
  - existing blocks
- Close Subview button class
- Close View button class
- collection time path attribute
  - definition
  - in data paths
  - in inference paths
  - resolving for peer input blocks
- color formatter procedure
- colors
  - Ack column
  - Alarm Message column
  - alarms, customizing
  - blocks, customizing
  - column header, modifying
  - column, modifying
  - customizing in configuration dialogs
  - customizing in networks
  - detail view label, modifying
  - detail view, modifying
  - inference paths, behavior
  - paths, customizing
  - queue view
  - Severity column
  - Time column
- colors-on-1st-level-color-menu parameter
- column cells, editing values in
- column cells, selecting
- column headers
  - color, modifying
  - conversion procedure
  - font size, modifying
  - label text, modifying
- columns
  - adding to queue view
  - colors, modifying
  - deleting from queue view
  - displaying attributes in
  - float formatter
  - formatting contents
  - height, modifying
  - modifying
  - moving in queue view
  - queue view
  - sort order, changing
  - sorting on
  - sorting, attributes that control
    - values, monitoring
    - width, modifying
- comments
  - adding to blocks
  - adding to queue entry
  - definition for queues
  - queue entry, saving
  - showing for blocks
  - viewing for queue entry
- comments attribute
- compilation of workspace
- configuration dialogs
  - customizing colors of
  - displaying
- Configuration Panels menu item
- configure menu item
- configure tabular view menu item, queue view
  - layout
- configuring
  - access manager
  - attributes
  - custom blocks
  - date and time formats
  - detail view
  - function key for Find Block menu item
  - layout for queue view
  - Queue Message block
  - queues
  - queues in a queue class
  - specific queue
  - view templates
- confirm-deletions attribute
- connecting
  - links to blocks
  - remote processes
  - to external data
  - workspaces using connection posts
- connection path regions, customizing
- connection posts
  - highlighting
  - using
- connection-caching-enabled? parameter
- connections
  - creating between blocks
  - creating for peer input blocks
  - deleting
  - illegal
  - path splitters
- constrain moving clause
- control paths
  - initializing

- multiple signals for overriding
  - using
- controlling flow of data
- conversion procedure for column header
- converting timestamp format
- Copy Selection to Clipboard button
- creating
  - access manager
  - application workspaces
  - buttons
  - detail view template
  - filters, permanent
  - filters, temporary
  - item paths
  - log file
  - queue views
  - queues
  - sensors
  - stubs for custom blocks
  - subclasses for custom blocks
  - view templates
- creating new menu preference objects
- current explanation menu item
  - generating explanation
  - inference block output value
- custom blocks
  - applying class definition for attributes
  - block evaluators
  - class names
  - cloning
  - configuring
  - connection stubs
  - creating custom path connections
  - creating subclasses of
  - creating using wizard
  - deleting subclasses
  - editing block evaluators
  - editing subclasses
  - icons
  - input path values
  - modules
  - output path attributes
  - palettes, reconfiguring
  - palettes, specifying
  - setting output path values
- Custom Class Wizard Activity Selector dialog box
- custom connections, adding to custom subclass

- customer support services
- customizing
  - alarm colors
  - block colors
  - buttons
  - configuration dialog colors
  - menus
  - network colors
- Cut Selection button

## D

- data
  - acquisition
  - controlling flow
  - editing in column cells
  - inferencing
- data coercion
  - using variables and parameters as input
  - using variables and parameters as output
- data expiration
- data flow, inhibiting through block
- data input, disabling
- data input, enabling
- data paths
  - initializing
  - overriding
- data seeking, setting timeout
- data server attribute
  - sensors
  - variables
  - variables, values
- data sources, of variables
- data value attribute, in data paths
- data values, overriding manually
- date and time formats, configuring
- date and time formats, table of
- date-format attribute
- date-time-format attribute
- default menu preferences, how GDA manages
- default-priority attribute
- Define New Filter dialog box
- delete block menu item
- delete palette menu item
- delete path menu item
- Delete Selected Rows button
- deleting
  - blocks
  - columns from queue view
  - custom block subclasses

- entries, confirming
- filters
- paths
- stubs
- toolbar buttons
- Descriptions dialog for explanations
- detail view
  - associating with View Details button
  - button classes
  - colors, modifying
  - configuring
  - default template, displaying
  - functions provided by
  - label colors, modifying
  - label position, modifying
  - position and scale
  - showing
  - size, modifying
  - template, creating
  - template, example
  - toolbar buttons
- details, viewing for queue entry
- detail-view-template attribute, modifying
- developer mode
- diagrams
  - building
  - executing
  - improving appearance
  - labeling
- disable evaluation menu item, blocks
- disable evaluation menu item, workspaces
- disable G2 menu item vs disabling evaluation
- Disable workspace menu item
- disabling
  - animation on startup
  - data input
  - evaluation of blocks
  - logging
- discrete logic
- displaying
  - attributes
  - custom block evaluators
  - value on path
- display-messages attribute
- do not highlight menu item
  - connection posts
  - paths
- dynamic-color-formatter attribute

## E

- edit master diagram menu choice
- Enable Data Input menu item
  - disabling data input
  - enabling data input
- enable evaluation menu item, blocks
- enable evaluation menu item, workspaces
- enable G2 menu item vs enabling evaluation
- enabling
  - data input
  - evaluation of blocks
  - logging
- encapsulation block and evaluation
- encapsulation blocks, definition
- encapsulations, single source
- entry point invocations
- entry points, evaluating Name of Sensor, using
  - an expression
- entry-class attribute
- entry-lifetime attribute
- entry-limit attribute
- Environment menu item
  - animation delay
  - data expiration and block evaluation
  - Find Block key binding
  - maximum timeout for data seeking
  - number of blocks with allow other
    - processing statement
  - snap grid resolution
- erase history when reset attribute
  - behavior
  - resetting
- error attribute
- error log entries
- error message, traceback of
- error message, viewing details
- Error Queue
  - built-in view template
  - built-in, workspace containing
  - default queue view
  - sorting, default sort order
  - using
- errors
  - clearing
  - default color
  - resetting
- evaluate menu item
- evaluating blocks
  - enabling and disabling evaluation
  - manually

- evaluation engine
  - evaluators, editing for custom subclass
  - example
    - access manager
    - acknowledging alarm
    - advice for alarm
    - Alarm Queue with selected entry
    - filtering alarm entries
    - locking queue view
    - require-acknowledgement attribute
  - executing diagrams
  - exit if attribute, evaluating expressions in
  - expected value attribute, evaluating
    - expressions in
  - expiration time path attribute
    - in data paths
    - in inference paths
    - peer input blocks
    - Validity Interval
  - expired alarm entry
  - expired inputs
  - expired quality
  - explain alarm menu item
  - explanation log entries
  - explanation of last false menu item
  - explanation of last true menu item
  - explanation of last unknown menu item
  - Explanation Queue
    - showing inference block output value
    - sorting, default sort order
    - using
    - view template, built-in
    - workspace containing built-in
  - explanations
    - alarm, viewing
    - concatenating text in
    - configuring to generate automatically
    - current, viewing
    - displaying current, blocks
    - generating automatically
    - generating current, logic gates
    - generating manually
    - generating, in general
    - queue entry, viewing
    - specifying
  - expressions
    - evaluating in attributes
    - using G2
  - extending main menu items
  - external data, connecting to using variables
    - and parameters
  - external datasource attribute, evaluating
    - expressions in
- ## F
- F2 key, find block
  - false, changing color of
  - file-time-format attribute
  - filtering
    - hysteresis
    - inference path values
    - queue entries
  - filters
    - applying
    - attributes of
    - deleting
    - example with alarm entries
    - names, number of characters in
    - operators in definitions of
    - permanent, creating
    - permanent, from temporary
    - queue entry attributes used with
    - temporary, creating
    - temporary, making permanent
  - Find Block menu item
  - float variables and parameters
  - float-format attribute
  - font size of column headers, modifying
  - font size of queue entries
  - font size queue view attribute
  - font-for-attribute-tables parameter
  - Fonts menu item
  - fonts, changing defaults for text
  - formats
    - date and time, configuring
    - date and time, table of
    - floating point numbers in column
    - time in alarm log entries
  - formatting column contents
  - forward chaining
    - example
    - initiating
  - free text, labeling diagrams
  - full history
  - function, evaluating in attributes
  - functions in attributes
  - fuzzy belief values
  - fuzzy inference values, overriding
  - fuzzy logic
    - attributes



hysteresis  
uncertainty

## G

GDA  
  block evaluation engine  
  loading  
GDA menu item  
gda-acknowledge-time attribute  
  displaying value  
  filters  
gda-alarm-detail-view-template  
gda-alarm-queue class  
gdaapps KB, loading  
gdaapps top-level module  
gdaapps.kb initial loading  
gda-color-entry-by-acknowledgement  
  procedure  
gda-color-entry-by-severity procedure  
gda-format-acknowledged attribute  
gda-menu-configuration menu preference  
  object  
gda-require-acknowledgement attribute and  
  filters  
gda-severity attribute and filters  
gda-severity attribute, default sort column  
gdl-custom-block class  
gdl-custom-multiple-invocations-block class  
gdl-custom-peer-input-block class  
gdl-evaluate-block invoking block  
gdl-filter-tag attribute and filters  
gdl-get-time-attribute-value attribute  
gdl-item-path class  
gdl-simple-encapsulation class  
gdl-single-source-encapsulation class  
general custom block  
  definition  
  local name  
  reference information  
General Queue Settings dialog box  
General Settings menu item  
generating explanations automatically  
generating explanations manually  
GFR, translating menu items  
GMS  
  extending main menu items  
  menu bar  
gms-default-configuration menu preference  
  object

Go to Source button  
  attribute, modifying  
  class  
  description  
  Message Queue  
GQM Views module  
GQM Views palette  
gqm-creation-time attribute  
  default sort column  
  displaying value  
  filters  
gqm-message-text attribute and filters  
gqm-priority attribute and filters  
gqm-queue class  
gqmv-action-button class  
gqmv-count-id attribute  
gqmv-default-detail-view-template example  
gqmv-default-detail-view-template, workspace  
  that contains  
gqs-queue class  
gqsv-get-attribute-value attribute  
gqsv-view-template-or-access-table attribute  
gqsv-toolbar-button class  
green, path color  
GXL spreadsheets, using to edit data

## H

height of columns, changing  
hide diagram menu choice  
hide name menu item  
hierarchy of workspaces  
highlight menu item, connection posts  
highlight menu item, paths  
highlighting  
  connection posts  
  paths  
highlight-procedure attribute, modifying  
history  
  definition (queues)  
  fixed sample time  
  limit, setting on queue  
  nonmonotonic values  
  partial  
  performance issues  
  performance issues, updating  
  point-based  
  point-based updates  
  propagating data  
  reset behavior

- sample time options
  - size of
  - time-based
  - time-based updates
  - values to maintain
  - viewing for alarms
- history-limit attribute
- hysteresis when attribute
- hysteresis, definition

**I**

- icons, creating for custom blocks
- ignore-duplicate-list-element-error parameter
  - value
- illegal connections
- incremental logging of alarm entries
- inference paths
  - colors and states
  - colors and status
  - filtering data on
  - filtering using hysteresis
  - fuzzy belief values, overriding
  - fuzzy belief values, using
  - initializing
  - overriding
  - overriding discrete
- inference values, overriding manually
- inhibiting data flow through block
- initial-column-key-to-sort attribute
- initializing path values
- initial-margin-for-workspaces parameter
- initial-sorting-order attribute
- input attribute
- input path values for custom blocks
- input ports
- Insert Row Above Selection button
- Insert Row After Selection button
- Inspect menu item
- integer variables and parameters
- invalid workspace
- Invocations-running attribute
- Invocations-waiting attribute
- invoking block
- item paths
  - creating
  - creating customized connections
  - displaying path items for
  - placing items onto interactively
  - placing items onto programmatically

- item-addition-callback attribute
- item-configuration attribute on workspace
- item-removal-callback attribute

**J**

- junction block

**K**

- key-value-conversion-procedure attribute

**L**

- labeling diagrams
- labels, modifying
- layout, configuring for queue view
- lifetime of queue entry
- limit of entries in queue
- lines-per-row attribute
- link, definition
- links
  - action
  - capability
  - connecting to blocks
  - restriction
- Load from File into Selection button
- Load KB menu item, loading application
- Load KB menu item, loading GDA
- loading
  - application
  - GDA
- Local Diagram, single source encapsulations
  - understanding
  - viewing
- Lock block, inhibiting data flow through block
- lock icon
  - on entry points
  - on variables and parameters
- lock menu item
- lock menu item, blocks
- lock menu item, variables and parameters
- Lock View button class
- Lock View button, description
- locking alarm queue view
- locking alarm queue view, example
- locking blocks
  - by overriding
  - manually
- log file

- alarm log entries
    - creating new
    - error log entries
    - explanation log entries
    - header
    - message log entries
    - name and location of
    - name in header
  - logging queue entries
    - alarm log formatter
    - enabling and disabling
  - logic
    - discrete
    - fuzzy
    - type
  - logic attribute
  - logical variables and parameters
- M**
- M override symbol on block icon
  - manual invocations
  - manual quality path attribute value
  - manual quality, when overriding blocks
  - Master Diagram, single source encapsulations
    - cancelling editing
    - editing
    - saving
      - locally
      - updating all instances
    - understanding
  - maximum data seeking timeout
  - menu items, translating
  - menu preference objects, creating new
  - menu preferences, how GDA manages the
    - default
  - menus
    - customizing
    - extending items
    - preferences
  - Menus menu item
    - creating menu preference
    - default menu preferences, using
    - deleting menu preference
    - displaying GMS menu bar diagrams
    - editing menu preference
  - message log entries
  - Message Queue
    - built-in view template
    - built-in, workspace containing
      - default queue view
      - sorting, default sort order
      - using
  - Message Queue view, Go to Source button
  - messages, sending to queues
  - milliseconds-to-sleep-when-idle parameter
  - minimum-scheduling-interval parameter
  - modes, changing
  - modes, definitions
  - modifying
    - button attributes
    - column colors
    - column header color
    - column header font size
    - column header text
    - column height
    - column width
    - columns in queue view
    - custom class definitions
    - data in column cells
    - detail view colors
    - detail view label colors
    - detail view label position
    - detail view size
    - evaluator of custom subclass
    - new queue views
    - number of rows in queue view
    - queue entry counters
    - queue entry font size
    - queue view colors
    - queue view labels
  - modules
    - about
    - renaming top-level
  - monitoring column values
  - monitor-this-attribute attribute
  - mouse tracking for detail view buttons
  - mouse tracking for queue view buttons
  - moving
    - columns in queue view
    - toolbar buttons
  - multiple control signals
  - multiple invocations attribute
  - Multiple Invocations attribute, Multiple Invocations custom block
  - multiple invocations custom blocks
    - definition
    - local names
    - reference information
  - multiple queue views

**N**

- name menu item
- name of sensor attribute, evaluating
  - expressions in
- names, hiding
- Network blocks
- Network menu item
- networks, customizing colors
- New Free Text menu item
- new menu preference objects, creating
- new value prompt attribute, evaluating
  - expressions in
- New Workspace menu item
- nonmonotonic history data
- non-peer input blocks
- notes attribute
- no-value quality
  - customizing path color of
  - how blocks use paths with
  - path attribute value
- number-of-columns-on-1st-level-color-menu
  - parameter

**O**

- OK button
- ok quality
- operators for filters
- option 1 description attribute, evaluating
  - expressions in
- option 2 description attribute, evaluating
  - expressions in
- option 3 description attribute, evaluating
  - expressions in
- ordination, configuring
- output path attributes for custom blocks
- output ports
- output uncertainty attribute
- override menu item, blocks
- override menu item, variables and parameters
- override text attribute, evaluating expressions
  - in
- overriding
  - blocks
  - blocks manually
  - control paths
  - data paths
  - inference paths, discrete
  - inference paths, fuzzy
  - parameters manually

- variables manually

**P**

- paint-mode? parameter
- palettes
  - creating for custom blocks
  - deleting for custom blocks
  - overview of
- parameters
  - coercing data, when using as input
  - coercing data, when using as output
  - connecting to blocks
  - creating
  - overriding
  - overriding manually
  - types
  - using in diagrams
- partial history
- Paste Contents of Clipboard button
- path attributes, peer input blocks, resolving
- path attributes, using
- path displays
- path vertices, creating
- paths
  - colors and types
  - colors, customizing
  - connecting using connection posts
  - connecting, for peer input blocks
  - connecting, general
  - control
  - customizing for custom blocks
  - data
  - deleting
  - displaying values of
  - highlighting
  - inference
  - initial values
  - item
  - links
  - peer input blocks
  - resetting
  - splitters
  - tables
  - using
- peer input blocks
  - connecting
  - definition
  - no-value inputs
  - resolving path attributes for

- peer input custom blocks
  - definition
  - local names
  - reference information
- performance issues, block history
- permanent filters, creating
- permanent filters, from temporary filters
- phases of application
- planning applications
- ports
- position
  - detail view
  - detail view label, modifying
  - queue entry counters
- Post Global Initializer attribute
- Post Global Resetter attribute
- preferences, menus
- preventing blocks from evaluating
- procedure, evaluating in attributes
- procedures
  - customizing for blocks
  - running at G2 startup
  - using in attributes

## Q

- quality hierarchy
- quality path attribute
  - control paths
  - data paths
  - inference paths
  - peer input blocks
  - using
  - using **no-value** inputs
- quantitative variables and parameters
- Queue Access Table
- queue entries
  - acknowledging and removing
  - adding, callback
  - beeping when posted to queue
  - class of
  - confirming deletion of
  - definition
  - details, viewing
  - displaying in queue
  - displaying message text
  - filters
  - font size, modifying
  - lifetime of
  - number of lines

- recurring alarms
- removing from queue view
- removing, callback
- reusing alarm entries
- saving
- saving details
- sending to another queue
- sorting attributes
- sorting on queue view
- source, showing
- total in queue
- updating attribute, callback
- visible in view
- queue entry counters
  - modifying
  - types
- Queue Message block
- Queue Message dialog box
- queue view, definition
- queue views
  - button classes
  - colors, modifying
  - creating
  - labels, modifying
  - locking
  - modifying new
  - multiple
  - removing entries from
  - unlocking
- queues
  - configuring all in a class
  - configuring individual
  - creating
  - definition
  - displaying explanations
  - new, using
  - number of entries

## R

- recurring-entry-class attribute
- red, path color
- Remote G2 Process block, changing color
- Remote Process menu item
- remote process query timeout period
- remote process retry period
- remote processes, connecting to
- Remove Entries button
  - acknowledge-selected-entries attribute
  - description

- modifying attribute
- Remove Selected Entries button
  - class
- removing
  - queue entries from queue view
  - queue entry from queue, callback
- require full history attribute, path quality
- require full history attribute, using
- require-acknowledgement attribute
- require-acknowledgement attribute, example
- reset all blocks menu item, resetting blocks
- reset all blocks menu item, resetting paths
- Reset All Blocks menu item, running
  - procedure after
- reset menu item, blocks
- Reset menu item, caution
- reset menu item, resetting blocks
- reset path menu item
- resetting blocks manually
- resort-new-entries attribute
- restriction link
- Reuse Entry preference and alarm history
- reuse-entry attribute
- rows in queue view, changing number of
- rule terminals, single source encapsulations
- running application

## S

- save diagram menu choice
- Save Entry button, description
- Save Selected button class
- Save Selection to File button
- Save Subview button class
- save-selected* file name
- saving
  - application periodically
  - queue entry
  - queue entry comments
  - queue entry details
- scale of detail view
- scheduling-mode parameter
- searching for block
- Select Filter dialog box
- Select Filters button
  - class
  - description
  - filters, creating
- selecting column cell
- Send Entries button class

- Send Entries button, description
- sending entries to another queue
- sending messages to queues
- sensors
  - creating
  - using in applications
- setting
  - animation delay
  - maximum timeout for data seeking
  - output path values for custom blocks
  - remote process query timeout period
  - remote process retry period
  - sweep interval
- Setup Application Workspace menu item
- severity
  - changing color of
- Severity column, color
- Severity column, displaying value
- show item menu item
- showing
  - comments for blocks
  - source of entry
- simple encapsulation blocks, definition
- single source encapsulation blocks
  - reference information
- single source encapsulation blocks, definition
- single source encapsulations
  - cancelling editing Master Diagram of
  - converting to
  - creating
    - instances of
    - subclasses of
  - displaying Local Diagram of
  - editing
    - block attributes of
    - Master Diagram of
    - Local Diagram of
    - Master Diagram of
    - restrictions when using
    - saving Master Diagram
      - locally
      - updating all instances
    - using with rule terminals
  - size of detail view, modifying
  - snap grid
    - limitations
- Sort Entries button, description
- sort order
- Sort Order button class
- sort order, changing

- Sort Rows of Selection in Ascending Order button
- Sort Rows of Selection in Descending Order button
- sorting entries
  - Alarm Queue, default sort order
  - configuring attributes for
  - Error Queue, default sort order
  - Explanation Queue, default sort order
  - Message Queue, default sort order
  - on queue view
- source of entry, showing
- splitters
- spreadsheets, GXL
- Start menu item, after reset
- Start menu item, starting GDA
- Startup menu item
  - disable animation on startup
  - enabling data input at startup
  - running procedure after reset
  - running procedure when starting G2
- status on initialization attribute, resetting
- status on initialization attribute, using
- status value path attribute
  - determining for fuzzy logic
  - inference paths
- strings, concatenating
- stub tools, creating stubs for custom blocks
- stubs
  - connecting blocks
  - creating for custom blocks
  - creating for peer input blocks
  - deleting
  - direction of flow
- superior object, evaluating in attributes
- sweep interval, setting
- symbolic variables and parameters
- system invocations
- system table, use of

## T

- table block menu item
- table path menu item
- tables
  - block
  - paths
- target attribute, evaluating expressions in
- target variable attribute, evaluating expressions in

- Telewindows Toolkit client
- temporary filters, creating
- temporary filters, making permanent
- text
  - concatenating
  - variables and parameters
- Time column, color
- time format of alarm log entries
- time-format attribute
- timeout for data seeking, setting
- timestamp path attribute
  - control paths
  - data paths
  - inference paths
  - using
- timestamp, converting format of
- Toggle Snap Grid menu item
- tooggling animation
- toolbar buttons
  - access manager
  - adding
  - creating
  - customizing
  - deleting
  - manipulating
  - modifying attributes
  - moving
- top-level module, renaming
- total entry counter
- tracebacks for alarms and errors
- translating menu items
- true, changing color of
- turning logging on and off

## U

- uncertainty, fuzzy logic
- Undo the Last Operation button
- uninterrupted-procedure-execution-limit parameter
- unknown, changing color of
- unlock menu item, blocks
- unlock menu item, for variables and parameters
- unlocking blocks
  - after overriding
  - by resetting
- unlocking queue view
- Update & View Explanation button class
- updating queue entry, callback

- use expired inputs attribute
- Use Expired Inputs attribute, Peer Input
  - custom block
- use of system table
- user mode
- user modes in access manager
- user names in access manager

## V

- validity interval attribute
  - determining expiration
  - values
- value on initialization attribute
  - control paths
  - data paths
  - resetting
- value on path, displaying
- values, monitoring in column
- variables
  - coercing data when using as input
  - coercing data when using as output
  - connecting to blocks
  - creating
  - data servers for
  - overriding manually
  - overriding, general
  - types
  - using in diagrams
- vertices, creating in path connections
- View Advice button class
- View Comments button class
- View Details button
  - associating with detail view
  - description
  - modifying attribute
- view entry counter
- View Explanations button class
- View History button class
- view local diagram menu choice
- View Message (detail view) button class
- View Message (queue view) button class
- view template
  - Alarm Queue
  - definition
  - detail view, displaying
  - Error Queue
  - Explanation Queue
  - Message Queue
  - specifying default

- view templates
  - accessing
  - creating
  - in access manager
- view templates, configuring
- viewing
  - alarm explanation
  - alarm history
  - current explanation
  - error message details
  - queue entry comments
  - queue entry details

## W

- when false attribute, evaluating expressions in
- when true attribute, evaluating expressions in
- when unknown attribute, evaluating
  - expressions in
- when-to-allow-multiple-menus parameter
- white, path color
- width of columns, changing
- window classes in access manager
- window-x-location attribute
- window-y-location attribute
- workspace hierarchy
- workspaces
  - connecting using connection posts
  - creating
- workspace-x-location attribute
- workspace-y-location attribute

## X

- x-scale attribute

## Y

- y-scale attribute