G2 Gateway

Bridge Developer's Guide

Version 2015

G2 Gateway Bridge Developer's Guide, Version 2015

January 2016

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

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

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

Part Number: DOC016-1200

Contents Summary

Contents

Preface

Describes the G2 Gateway Bridge Developer's Guide and the conventions that it uses.

About this Guide

This guide describes the G2 Gateway standard interface (GSI), which allows you to create generic bridges between G2 and external systems. Gensym provides a number of higher-level bridges between G2 and standard databases and standards such as ActiveX, Java, CORBA, and OPC. If your application needs to communicate with databases or standards, use one of these bridges instead of G2 Gateway.

Product Name

In G2 5.0, the GSI product name changed to G2 Gateway. Within G2, however, no such change has been made: grammar prompts and item names still refer to GSI, rather than to G2 Gateway. Changing these references would caused existing applications to fail.

This manual uses “G2 Gateway” to refer to the product as a whole, and it refers to “GSI” when an internal component of it is described, such as a GSI interface.

Audience

This guide is intended for developers of G2 Gateway bridge applications, whom it addresses throughout as “you”. It assumes that you have a working knowledge of programming in the C language. It also assumes that you know how to create and configure G2 objects such as classes, class instances, rules, and procedures.

Organization

This guide contains 12 chapters and six appendixes in four parts:

	Title	Description
Part I	**User’s Guide**	
1	G2 Gateway Solutions for Connectivity Problems	Describes how you can develop solutions to your communication problems, by using G2 Gateway bridges, through which G2 applications and dynamic external processes can communicate with each other.
2	Configuring the G2 Knowledge Base	Describes how to create GSI interfaces, GSI variables, and GSI message servers that enable your G2 knowledge base to communicate with a G2 Gateway bridge.
3	Preparing the Bridge User Code	Describes how to organize and code the customized portion of the G2 Gateway bridge.
4	Remote Procedure Calls	Describes how a G2 Gateway bridge and a G2 application can make remote procedure calls to each other.
5	Error Handling	Describes how G2 Gateway handles errors by default, and how you can customize error handling in your G2 Gateway bridge.
6	Troubleshooting Guidelines	Describes how to identify problems in your G2 Gateway bridge user code.

	Title	Description
Part II	**Reference**	
7	G2 Gateway Data Structures	Describes how G2 Gateway data structures store information that is useful to your application, and how your G2 Gateway user code can access this information.
8	Callback Functions	Describes the callback functions that you complete to implement your G2 Gateway user code.
9	API Functions	Describes the capabilities and syntax of the API functions supported by G2 Gateway.
10	Preprocessor Flags and Runtime Options	Describes C preprocessor macros and runtime options that you can use to modify the behavior of your G2 Gateway bridge.
G	Upgrading G2 Gateway Applications	Describes how to upgrade existing GSI applications to G2 Gateway 6.0.
11	Building and Running a G2 Gateway Bridge	Describes how to compile, link, and run a G2 Gateway bridge executable image, and how to start and stop a G2 Gateway bridge process from within a G2 procedure.
Part III	**Appendixes**	
A	Functions by Argument and Return Type	Lists the API and callback functions provided by G2 Gateway, grouped by the data types of their arguments and their return values.
B	Constants	Lists symbolic constants defined in G2 Gateway header files.
C	G2 Gateway Error Messages	Lists and describes the standard error messages returned by G2 Gateway.
D	G2 Gateway Data Types	Describes the data types defined for use in G2 Gateway user code.

	Title	Description
E	Limits and Ranges	Describes limits and ranges applicable in G2 Gateway.
F	How G2 and G2 Gateway Exchange Data	Provides a brief summary of techniques for exchanging data between a G2 Gateway bridge and a G2 KB.

A Note About the API

The G2 Gateway API, as described in this guide, is not expected to change significantly in future releases, but exceptions may occur. A detailed description of any changes will accompany the release that includes them.

Therefore, it is essential that you use G2 Gateway exclusively through its API, as described in this guide. If you bypass the API, you cannot rely on your code to work in the future, since G2 Gateway may change, or in the present, because the code may not correctly manage the internal operations of G2 Gateway.

If G2 Gateway does not seem to provide the capabilities that you need, contact Gensym Customer Support at 1-781-265-7301 (Americas) or +31-71-5682622 (EMEA) for further information.

Conventions

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

Typographic

Convention Examples	Description
g2-window, g2-window-1, ws-top-level, sys-mod	User-defined and system-defined G2 class names, instance names, workspace names, and module names
history-keeping-spec, temperature	User-defined and system-defined G2 attribute names
true, 1.234, ok, “Burlington, MA”	G2 attribute values and values specified or viewed through dialogs

Convention Examples	Description
Main Menu > Start KB Workspace > New Object create subworkspace Start Procedure	G2 menu choices and button labels
conclude that the x of y ...	Text of G2 procedures, methods, functions, formulas, and expressions
new-argument	User-specified values in syntax descriptions
text-string	Return values of G2 procedures and methods in syntax descriptions
File Name, OK, Apply, Cancel, General, Edit Scroll Area	GUIDE and native dialog fields, button labels, tabs, and titles
File > Save Properties	GMS and native menu choices
workspace	Glossary terms
`c:\Program Files\Gensym\`	Windows pathnames
`/usr/gensym/g2/kbs`	UNIX pathnames
`spreadsh.kb`	File names
`g2 -kb top.kb`	Operating system commands
`public void main()` `gsi_start`	Java, C and all other external code

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

Procedure Signatures

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

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

Related Documentation

G2 Core Technology

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

G2 Utilities

- *G2 ProTools User's Guide*
- *G2 Foundation Resources User's Guide*
- *G2 Menu System User's Guide*
- *G2 XL Spreadsheet User's Guide*
- *G2 Dynamic Displays User's Guide*
- *G2 Developer's Interface User's Guide*
- *G2 OnLine Documentation Developer's Guide*
- *G2 OnLine Documentation User's Guide*

- *G2 GUIDE User's Guide*
- *G2 GUIDE/UIL Procedures Reference Manual*

G2 Developers' Utilities

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

Bridges and External Systems

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

- *G2 CORBALink User's Guide*
- *G2 WebLink User's Guide*

G2 JavaLink

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

G2 Diagnostic Assistant

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

Customer Support Services

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

To obtain customer support online:

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

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

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

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

➔ Use the following numbers and addresses:

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

Part I

User's Guide

Chapter 1: G2 Gateway Solutions for Connectivity Problems

Describes how you can develop solutions to your communication problems, by using G2 Gateway bridges, through which G2 applications and dynamic external processes can communicate with each other.

Chapter 2: Configuring the G2 Knowledge Base

Describes how to create GSI Interfaces, GSI variables, and GSI message servers that enable your G2 knowledge base to communicate with a G2 Gateway bridge.

Chapter 3: Preparing the Bridge User Code

Describes how to organize and code the customized portion of the G2 Gateway bridge.

Chapter 4: Remote Procedure Calls

Describes how a G2 Gateway bridge and a G2 application can make remote procedure calls to each other.

Chapter 5: Error Handling

Describes how G2 Gateway handles errors by default, and how you can customize error handling in your G2 Gateway bridge.

Chapter 6: Troubleshooting Guidelines

Describes how to identify problems in your G2 Gateway bridge user code.

1

G2 Gateway Solutions for Connectivity Problems

Describes how you can develop solutions to your communication problems, by using G2 Gateway bridges, through which G2 applications and dynamic external processes can communicate with each other.

Introduction

This manual describes how to use Gensym's G2 Gateway to develop interfaces, or **bridges**, that support two-way communication between dynamic external processes and G2 applications.

Through a G2 Gateway bridge to an external system, a G2 application can quickly obtain real-time data that it needs to make intelligent control decisions in a time-critical processing environment. The G2 application can also update the state of an external system — for example, by writing or updating a record in a database, or by controlling a PLC.

A G2 Gateway bridge process and a G2 process run concurrently, enabling the G2 application to continue to perform its tasks while the G2 Gateway bridge manages the communication between G2 and an external system.

G2 Gateway bridges enable G2 KBs to communicate with a wide variety of external systems, such as:

- Database management systems (DBMSs)
- Programmable logic controllers (PLCs)
- Supervisory control and data-acquisition (SCADA) systems
- Distributed control systems (DCSs)
- C/C++ programs
- Non-G2 operator consoles or displays
- External simulation software

G2 Gateway bridges can communicate across networks that use the TCP/IP protocol. Gensym's Intelligent Communications Protocol (ICP), which is built into G2 Gateway, handles details of network communication automatically, enabling you to develop distributed systems among heterogeneous platforms without having detailed knowledge of protocols or of network software in general.

Gensym and its Solution Partners provide bridge products implemented with G2 Gateway that support communication between G2 and a wide range of external systems, including many common databases and programmable logic controllers. Call your Gensym account representative for information about Gensym's bridge products.

Capabilities of G2 Gateway Bridges

G2 Gateway is shipped with C code libraries of high-level functions that enable a G2 application and a G2 Gateway bridge to provide real-time data service for G2 variables, make remote procedure calls into each other, and exchange copies of G2 objects.

Providing Data Service for G2 Variables

G2 Gateway can act as a high-performance data server for variables in G2. Through a G2 Gateway bridge, a G2 application can both send data values to and receive values from an external system. The data can be numbers, symbols, truth values, text messages, arrays, or lists.

The G2 application can actively solicit data from an external system, and an external system can send data to G2 without having received a request from G2 for the data.

Invoking Remote Procedures

A G2 KB can invoke user-written functions in a G2 Gateway bridge, and a G2 Gateway procedure can invoke G2 methods and procedures. When a remote procedure is invoked by G2, G2 continues to perform its other tasks while the procedure call returns values and completes.

Through remote procedure calls, a G2 application and a G2 Gateway bridge can exchange data values, references to G2 objects, or copies of G2 objects as explained in the following section.

Passing Objects

A G2 application typically stores important real-time data in attributes of G2 objects. Through remote procedure calls, a G2 application and a G2 Gateway bridge can exchange copies of these data-rich objects. A G2 application and a G2 Gateway bridge can exchange copies of any G2 object that inherits from the G2 item class.

A G2 application can pass a copy of a G2 object to the bridge by invoking a G2 Gateway bridge function as a remote procedure. G2 specifies the object to be passed as an argument of the remote procedure call.

When a G2 Gateway bridge receives a copy of the G2 object, it creates its own data structures to represent the object. The G2 Gateway bridge can read from and write to the data contained in these data structures. The G2 Gateway bridge can return these data structures to G2 through remote procedure calls to G2 procedures. G2 creates objects of existing G2 classes based on the data structures that it receives from the G2 Gateway bridge.

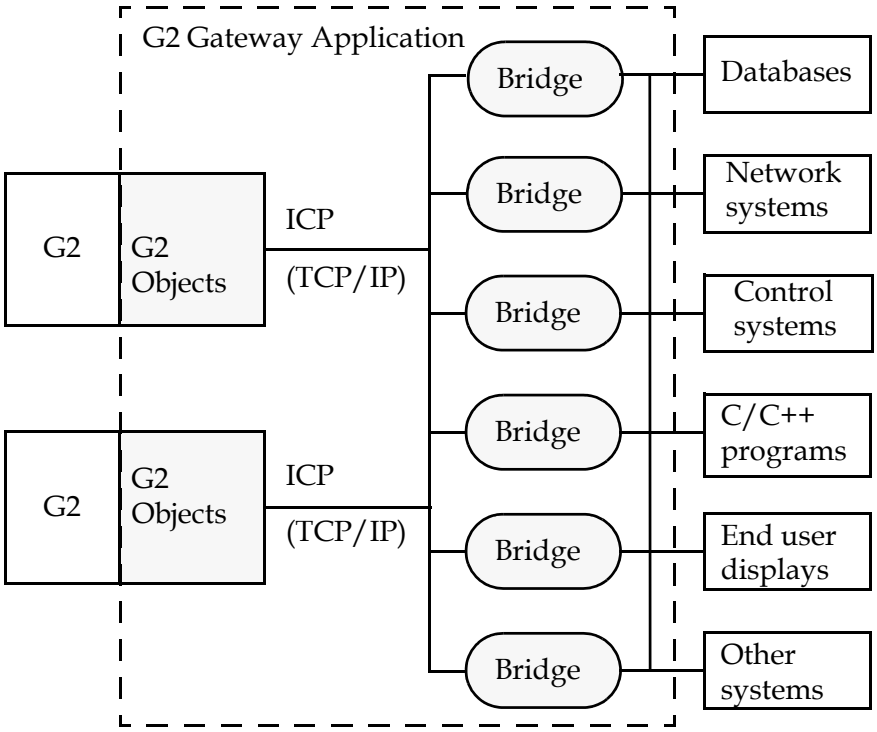
Other Support for Dynamic Real-Time Processing

G2 Gateway provides other features in support of dynamic real-time processing, including:

- Time-stamping of data exchanged between a G2 application and a G2 Gateway bridge.
- Exchange of text messages with the external system: a G2 application can send text messages to the G2 Gateway bridge, and the G2 Gateway bridge can post messages on the G2 Message Board.

Developing G2 Gateway Applications

To implement a solution to your connectivity problems, you create a **G2 Gateway application** consisting of one or more G2 Gateway bridges, and one or more G2 KBs. The following figure illustrates the possible components of a G2 Gateway application.



As the figure above illustrates:

- Each G2 KB can be connected to more than one G2 Gateway bridge process. Each G2 KB contains objects that support communication with the bridge processes.
- Each G2 Gateway bridge can be connected to more than one G2 KB (as many as 50 KBs), and to more than one external system.

Steps for Developing a G2 Gateway Application

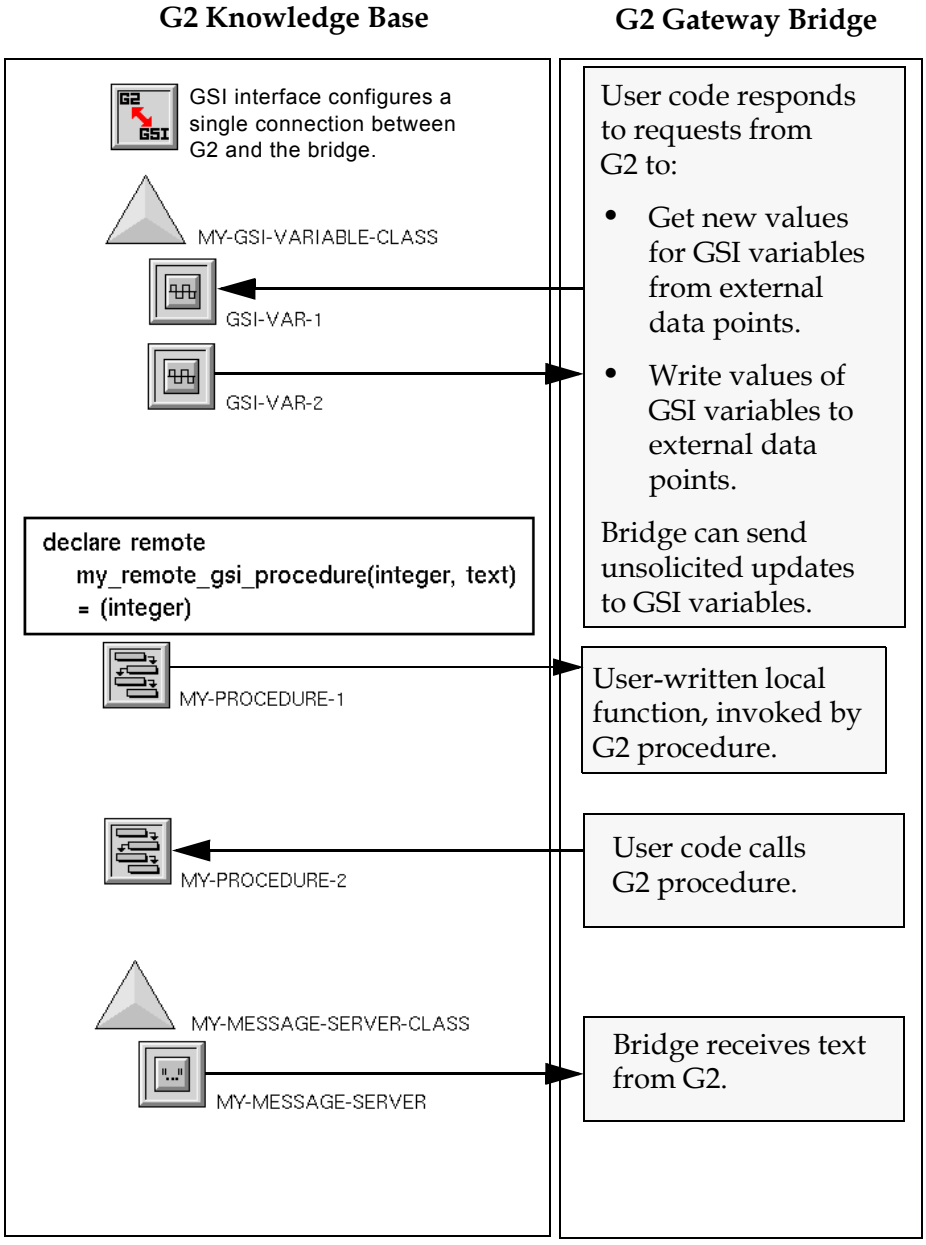
To develop a G2 Gateway application, you must:

- Create and configure G2 Gateway objects in each G2 KB that will communicate with a G2 Gateway bridge process.
- Create one or more executable G2 Gateway bridges.

Preparing a G2 KB to Communicate with a G2 Gateway Bridge

The following figure illustrates the G2 objects that you create and configure in a G2 KB to enable it to communicate with a G2 Gateway bridge:

G2 Objects that Support Communication with a G2 Gateway Bridge



To prepare the G2 KB for communication with a G2 Gateway bridge:

1 Create a **GSI interface** to define operating characteristics of each connection between a G2 KB and a G2 Gateway bridge process. A GSI interface is *required* for any communication between a G2 KB and a G2 Gateway bridge process.

Each GSI interface must be an instance of the standard class gsi-interface, or of a subclass of this class. You create a GSI interface using the standard G2 menu for creating new objects. You then edit attributes of the GSI interface to identify the G2 Gateway process with which G2 will communicate over this connection and to define operating characteristics of the connection.

A GSI interface is created automatically when a G2 Gateway bridge initiates a connection to G2 by calling the API function `gsi_initiate_connection()`. For information about this function, see *gsi_initiate_connection*.

2 Create a **GSI variable** to represent each data point in an external system that your G2 KB needs to read from or write to. These variables will receive data service from the G2 Gateway bridge.

To create GSI variables, you first define an object class that includes the G2 mixin class gsi-data-service as a direct superior class. You then create instances of your GSI variable class to represent separate data points in the external system.

3 Create G2 procedures that the G2 Gateway bridge can call as remote procedures. You create these procedures using the standard G2 menu for creating new object definitions.

In your G2 Gateway user code, call `gsi_rpc_declare_remote()` to declare each G2 procedure that your G2 Gateway bridge process needs to invoke as a remote procedure.

4 To support remote procedure calls from G2 to the G2 Gateway bridge, create one or more local functions in your G2 Gateway user code.

Through calls to G2 Gateway procedures, your G2 KB can send copies of G2 objects, references to G2 objects, and data values to the G2 Gateway bridge. The bridge can return objects, references, and values to G2 through remote procedure calls.

In your G2 Gateway user code, invoke `gsi_rpc_declare_local()` to declare each local function that G2 needs to invoke as a remote procedure.

In your G2 KB, create a remote procedure declaration for each G2 Gateway local function that your G2 KB needs to invoke as a remote procedure. You create remote procedure declarations using the standard G2 menu for creating new object definitions.

For information about how to make remote procedure calls, from G2 to G2 Gateway and from G2 Gateway to G2, see Remote Procedure Calls.

5 Create a **GSI Message Server**, to enable your G2 KB to send text messages to an external system.

To create a GSI Message Server, you first define a new object class that includes the G2 mixin class gsi-message-service as a direct superior class. You then create an instance of your message server class. To send a text message to the G2 Gateway bridge, a G2 KB runs an Inform action on the message server. In your G2 Gateway bridge process, you must complete the callback *gsi_receive_message()* to receive the message from G2 and send it to the external system.

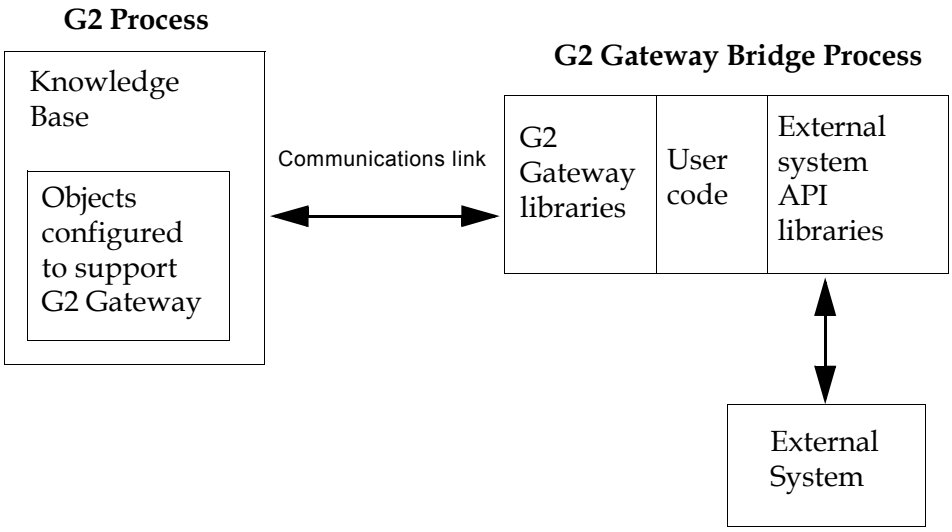
The steps that you follow to create GSI interfaces, GSI variables, and GSI message servers are described in Configuring the G2 Knowledge Base. For information about *gsi_receive_message()*, see *gsi_receive_message*.

Building a G2 Gateway Bridge Executable

You build the executable image of your G2 Gateway bridge by compiling and linking your G2 Gateway user code written in the C or C++ programming language with G2 Gateway libraries, and, optionally, with libraries of API functions provided with external systems.

The following figure illustrates the components that you build into the executable image of a G2 Gateway bridge process:

Components of a G2 Gateway Bridge Process



On Windows platforms, the G2 Bundle ships with two G2 Gateway directories, one called *gsi-intc*, which contains the GSI libraries and examples compiled with the Intel compiler, and the other called *gsi-msvc*, which contains the GSI

libraries and examples compiled with Microsoft Visual Studio. The components of the executable image are:

- **G2 Gateway libraries** of network-oriented application programmer interface (API) functions that can perform the following tasks for your bridge:
  - Establishing and maintaining the communications link to the G2 process, automatically managing all communications across the link.
  - Supporting the G2 Gateway main processing loop, in which the G2 Gateway bridge process responds to network activity on connections to G2 by invoking appropriate user code functions.
  - Receiving requests to send values to data points in the external system from the G2 knowledge base and calling appropriate user code functions to handle the requests.
  - Sending data to G2 at the request of the G2 Gateway user code.
- **User code**, which processes G2 requests and reacts to events in external systems. User code includes:
  - *gsi_main.h*: This is a source code header file provided by Gensym that you must include in all your user code files. Do not modify this header file.
  - *gsi_main.c*. This is a C source code file provided by Gensym that contains a sample of the *main()* routine from which G2 Gateway is started. Modify this file or replace it to suit your application.
  - (on Windows) *gsimmain.c*. This is a C source code file provided by Gensym that performs special initializations required only on Windows platforms when building a windows application, and then calls the *main()* function that you define in the *gsi_main.c* file. The file *gsimmain.c* is not needed when building a console application.

Note On Windows platforms, you must compile and link the file *gsimmain.c* with your G2 Gateway application. When building a windows application, *gsimmain.c* performs special initializations required only on Windows platforms, and then calls the *main()* function that you define in the *gsi_main.c* file. You do not need to make any changes to *gsimmain.c*. The file *gsimmain.c* is not needed when building a console application on Windows.

  - One or more source files of **callback functions**. G2 Gateway invokes each callback function *automatically* in response to a particular network event on a connection between the G2 Gateway bridge and G2, such as the activation of the connection or a request from G2 for a new value for a variable.

    Gensym provides a source file, *skeleton.c*, of uncompleted callback functions. You complete the code of the callback functions that you need

for your application, and leave the other callback functions in their uncompleted form.

Within the callback functions, you include code that implements your G2 Gateway bridge's response to the network event. Callback functions can include calls to G2 Gateway API functions, to API functions of an external system, to user written procedures, or to any other procedures or functions available to a C or C++ program.

- User-written functions that G2 can invoke through remote procedure calls.

- **External system application programmer interface libraries**, which provide a programmable interface between the G2 Gateway bridge and the external system.

To build the G2 Gateway bridge executable image:

1. Complete callback functions in the *skeleton.c* file provided with G2 Gateway.

   In order to link properly, your G2 Gateway bridge code must include at least the stub version of every callback function in *skeleton.c*. The G2 Gateway functions and the G2 Gateway data structures that callback functions can invoke and access are described in Part II, Reference, of this manual.

2. Modify or replace the *main()* routine in the *gsi_main.c* source code file as needed for the purposes of your G2 Gateway bridge.

   The *main()* function initializes G2 Gateway data structures and passes control to the G2 Gateway bridge's own processing loop.

   On Windows, you can edit the *main()* and *WinMain()* procedures before compiling and linking *gsi_main.c* and *gsimmain.c*. Use this option if your application includes Windows code, which you place in *WinMain()*.

3. Write and declare G2 Gateway functions that G2 can call as remote procedures.

4. Compile your user code and link it with the G2 Gateway libraries and with any libraries of external API functions required by your bridge.

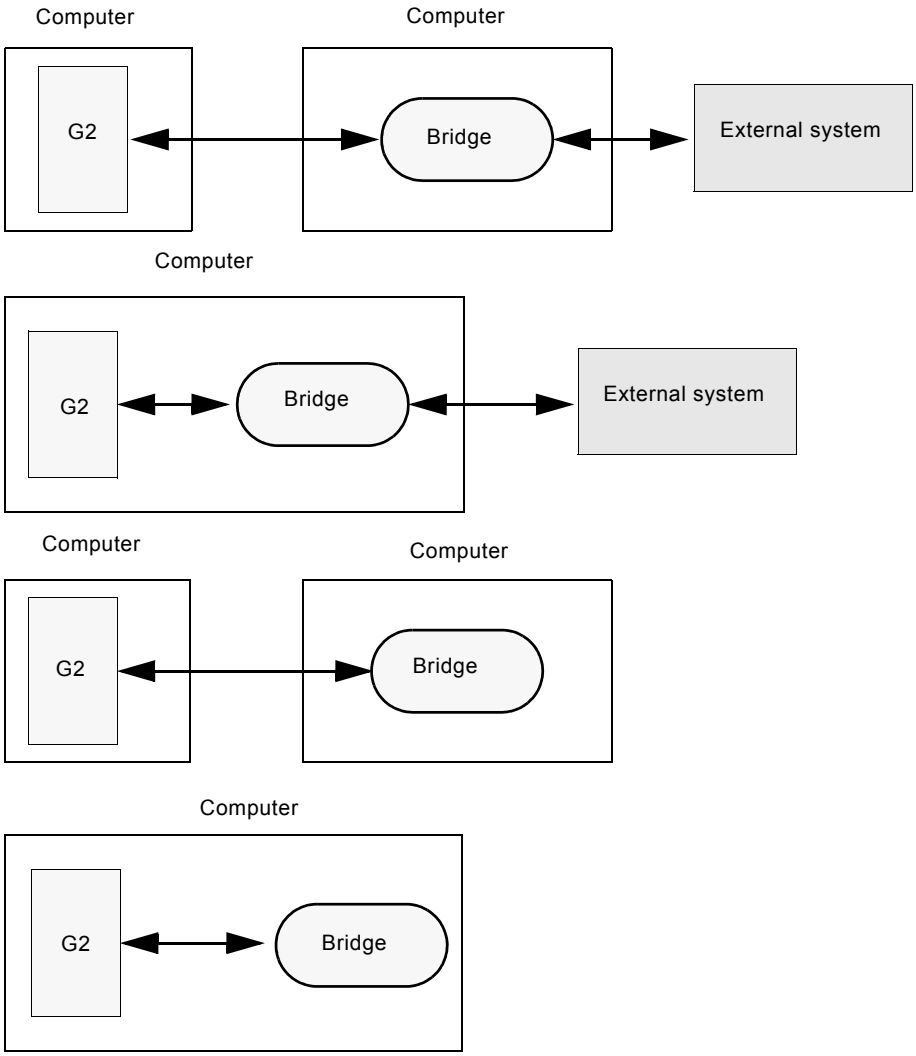
These steps are described in detail in Preparing the Bridge User Code and in Remote Procedure Calls.

Building and Running a G2 Gateway Bridge describes the requirements for compiling and linking the executable image on each supported platform.

Deploying G2 Gateway Bridges

The following figure illustrates several ways that you can deploy G2 Gateway bridges and the G2 KBs with which they communicate:

Deploying G2 Gateway Bridge Processes



As the figure above illustrates:

- A G2 process and the G2 Gateway bridge with which it communicates can run on the same computer or on different computers.
- A G2 Gateway bridge can be used to exchange data with a G2 process alone, or with both a G2 process and an external system. A G2 Gateway bridge that is not connected to an external system can provide data for G2 variables, or perform a special computation at the request of the G2 KB.

Starting G2 Gateway Bridge Processes

In both of the previous cases, G2 Gateway bridge runs as a separate process, distinct from any G2 process. You can start a G2 Gateway bridge process from within a G2 process, or independently of any G2 process.

For information about how to start a G2 Gateway bridge process from G2, see Starting a G2 Gateway Bridge from within G2.

How a G2 Gateway Bridge Works

After you start a G2 Gateway bridge process, the bridge process initializes both its private data structures and the application-defined data structures.

The G2 Gateway bridge process then waits for one or more G2 processes to initiate a connection to it. The G2 Gateway bridge process can itself initiate a connection to a G2 KB, calling the API function `gsi_initiate_connection()`.

Procedural Flow of a G2 Gateway Bridge Process

The `main()` function in your G2 Gateway bridge user code must call the G2 Gateway API function `gsi_start()` before it calls any other API function. `gsi_start()` takes as input the *argc* and *argv* arguments that were passed to the `main()` function of your G2 Gateway program and uses these arguments to set up listeners as specified in the command line.

The `gsi_start()` function in turn calls:

- `gsi_set_up()`, a callback that is useful for performing tasks that are required only once during the lifetime of the G2 Gateway bridge process, such as installing customized error handlers, setting or resetting initial G2 Gateway run-time options, declaring local functions in the user code as remote procedures, and allocating arrays of G2 Gateway data structures.
- `gsi_get_tcp_port()`, a callback that can provide a default TCP/IP port number that G2 Gateway will listen on for connections from a G2 process, if no port number or a port number of 0 is specified on the command line used to start the G2 Gateway bridge process.

- *gsi_run_loop()*, an API function that provides the main processing loop of the G2 Gateway bridge process. When *gsi_run_loop()* is executing, G2 Gateway does the following:
  - Makes any new connections requested by G2.
  - Responds to all outstanding messages received from G2 over each currently active connection. In response to these messages, G2 Gateway can invoke local functions called by G2 as remote procedures, or invoke callback functions in response to particular network events or requests from G2. For information about how to use *gsi_run_loop()*, see Processing Events through gsi_run_loop().

Run-Time Modes of Bridge Operation

To provide you with greater flexibility in the design of your user code, G2 Gateway supports two run-time modes of bridge operation: **continuous mode** and **one-cycle mode**. The modes affect the behavior of *gsi_run_loop()* and the procedural flow of the G2 Gateway bridge process:

- In continuous mode (the default mode), *gsi_run_loop()* loops repeatedly as long as no fatal error occurs. (For a list of G2 Gateway fatal errors see Appendix C, G2 Gateway Error Messages.) Your user code does not call *gsi_run_loop()* explicitly. At the end of each loop, *gsi_run_loop()* calls *gsi_pause()*, which causes the bridge process to enter an interruptible sleep. The bridge awakens when it detects network activity to which it must respond.

  Continuous mode is the better mode for bridges designed to obtain data from external systems by polling them actively.

- In one-cycle mode, *gsi_run_loop()* executes once automatically, and then exits. To reenter *gsi_run_loop()*, you must include an explicit call to *gsi_run_loop()* in your G2 Gateway user code. Running the bridge in one-cycle mode enables you to pass control from *gsi_run_loop()* to other functions within your G2 Gateway bridge process, as required by your application.

  One-cycle mode is the better mode for bridges designed to respond to network activity on connections to external systems, rather than those designed to poll the external systems actively.

Providing Data Service for GSI Variables in a G2 KB

A G2 Gateway bridge can provide data service for G2 variables in the current KB of a G2 process. G2 variables that receive data service from a G2 Gateway bridge process are called **GSI variables.**

GSI variables in a G2 KB can receive values from a G2 Gateway bridge through either solicited or unsolicited data service, as follows:

- **Solicited data service** occurs when G2 requests the G2 Gateway bridge to produce (or to obtain from the external system) data at regular time intervals. G2 passes this time interval to the G2 Gateway bridge before data service begins. If the G2 Gateway bridge must, in turn, request that data from the external system, it does so at the same time interval.

  The G2 Gateway bridge can also provide solicited data to G2 on demand — for example, when G2 requests an updated value for a particular GSI variable.

- **Unsolicited data service** occurs when G2 receives data from the G2 Gateway bridge that it has not requested. A G2 Gateway bridge process can obtain unsolicited data from an external system by polling the external system (in continuous mode), or by responding to messages sent to it by the external system (in one-cycle mode). The G2 Gateway bridge then transfers the data to G2.

Each GSI variable references a GSI interface, which identifies a single network connection to a G2 Gateway bridge process and records the status of that connection. See Configuring the G2 Knowledge Base, for information about how to create and configure GSI variables and GSI interfaces.

Setting Data Values in an External System

You can use the G2 set action to assign values to data points in an external system. You run the set action on a GSI variable, which causes G2 to send the value specified in the set action to the G2 Gateway bridge. Your bridge in turn sets a particular data point in the external system to the value that it receives from G2. The G2 Gateway bridge maintains a mapping between the GSI variable in the G2 KB and a data point in the external device, application, or data-processing system.

When the G2 KB performs the set action on a GSI variable, G2 sends the specified value to the G2 Gateway bridge by means of the specified GSI interface. When the G2 Gateway bridge receives the value, it calls the *gsi_set_data()* callback function in the user code of the G2 Gateway bridge. You must include code in this callback to send the communicated value to the appropriate bridge data structure or to the external system.

To echo that value back to the GSI variable in G2, you can include a call to the API function *gsi_return_values()* in the *gsi_set_data()* callback.

Sending Text Values to and from the G2 Gateway Bridge

A G2 KB can pass text messages to an external system through a G2 Gateway bridge process. To do this, in the G2 KB you define a user-defined class that must inherit from the standard G2 mixin class gsi-message-service. Your G2 KB sends the text by running an inform action on an instance of this user-defined class. In your G2 Gateway bridge process, you must complete the callback *gsi_receive_message()*, so that it responds to the message from G2 as required by your application. For more information about how a G2 KB can send text messages to an external system, see Creating and Configuring GSI Message Servers.

To send a text message to G2, your G2 Gateway user code can call the API function *gsi_return_message()*, which prints a message on the G2 message board. For information about how to do this, see *gsi_return_message*.

Making and Receiving Remote Procedure Calls

Using the remote procedure call (RPC) mechanism, a procedure in a G2 KB can directly invoke a user-written function in a G2 Gateway bridge process. G2 can send simple data values, copies of G2 objects, or references to G2 objects to the G2 Gateway bridge process, through remote procedure calls to user-written G2 Gateway functions.

Similarly, a user-written function in a running G2 Gateway bridge can directly invoke a procedure in the current KB of a running G2 process.

For more information about how a G2 Gateway bridge and G2 can communicate through remote procedure calls, see Remote Procedure Calls.

2

Configuring the G2 Knowledge Base

Describes how to create GSI Interfaces, GSI variables, and GSI message servers that enable your G2 knowledge base to communicate with a G2 Gateway bridge.

Introduction

This chapter describes how to:

- Configure a connection between the G2 knowledge base and a G2 Gateway bridge. To do this, you create a GSI interface in the G2 knowledge base.
- Create GSI variables that your G2 knowledge base uses to write to and read from data points in an external system. You create GSI variables as instances of a GSI variable class, which you must define.
- Create a GSI Message Server in the G2 knowledge base, and use it to send text messages to a G2 Gateway bridge. To send the messages, you run the G2 inform action on the GSI Message Server.

For information about how to prepare your G2 knowledge base to communicate with a G2 Gateway bridge through remote procedure calls, see Remote Procedure Calls.

Configuring Connections between G2 and G2 Gateway

You must create and configure a **GSI interface** for each connection between a G2 knowledge base and a G2 Gateway bridge. You edit attributes of the GSI interface to configure one connection between G2 and G2 Gateway.

A GSI interface serves the following purposes:

- To identify a G2 Gateway bridge with which this G2 process will attempt to establish a network connection.
- To specify whether G2 or the G2 Gateway bridge determines when data is passed from the bridge to G2.
- To indicate whether the G2 Gateway bridge will receive unsolicited data from the external system.
- To designate as many as six user-defined attributes of each class of GSI variables as the **identifying attributes** of that class. The values of a GSI variable's identifying attributes must distinguish it from all other variables in the KB. The identifying attributes of a GSI variable provide a unique identification for the variable, which G2 Gateway needs in order to map it to a data point in an external system.
- To contain a status value for a particular connection between this G2 and the G2 Gateway bridge. This status reflects the condition of the connection to the G2 Gateway bridge.

  After a network connection has been established between a G2 process and a G2 Gateway bridge across a particular GSI interface, the G2 knowledge base can consult that GSI interface for the status of that connection.

A GSI interface must be activated in order to support a connection between the G2 process and a G2 Gateway bridge. When the GSI interface is *activated*, G2 attempts to make a network connection with the specified G2 Gateway bridge. When the GSI interface is *deactivated*, G2 breaks the connection (if one still exists) between itself and the G2 Gateway bridge. For information about how to activate and deactivate a GSI interface, see Activating and Deactivating a GSI Interface.

A GSI interface can reside on any kb-workspace in the G2 knowledge base. A GSI interface can be an attribute of another G2 object.

Number of GSI Interfaces Required

A G2 knowledge base must contain at least one GSI interface for each G2 Gateway bridge with which it is communicating.

A G2 knowledge base can also communicate with the same G2 Gateway bridge through more than one GSI interface.

Using more than one GSI interface might be necessary when a G2 Gateway bridge provides data service for the G2's variables, but the bridge obtains data for some of the variables in a solicited manner and for others in an unsolicited manner. Using more than one GSI interface might also be necessary when the same G2 Gateway bridge provides both data service for a G2's variables and functions that the G2 can invoke through remote procedure calls.

Note Variables that require G2 Gateway data service, items that require GSI message service, and the knowledge base's executable items can use the *same* named GSI interface to refer to the same G2 Gateway bridge.

Creating a GSI Interface

A GSI interface is an instance either of the G2 standard gsi-interface class or of a user-defined subclass of the gsi-interface class.

To create and configure a GSI interface:

1 Create an instance of gsi-interface by selecting:

KB Workspace > New Object > network-interface > gsi-interface (or a subclass of gsi-interface)

2 Open the attribute table of the GSI interface for editing by selecting:

gsi interface > table

Setting Attributes of a GSI Interface

The following table summarizes the attributes of a GSI interface that you edit to configure a connection between a G2 knowledge base and a G2 Gateway bridge.

GSI Interface Attributes

Attribute	Description
names	One or more unique names for the object.
Allowable values:	Any valid object name (symbol).
Default value:	none
Notes:	Every GSI variable and Message Server must reference one of the names listed in the names attribute of a GSI interface.

GSI Interface Attributes

Attribute	Description
gsi-connection-configuration	Specifies the communications protocol and location of the G2 Gateway bridge process.
Allowable values:	tcp-ip host "*hostname*" port-number *tcp-ip-port-number* secure {yes | no}
Default value:	none
Notes:	See Gsi-Connection-Configuration Attribute.
external-system-has-a-scheduler	Specifies whether G2 or the G2 Gateway bridge determines when data is returned to G2.
Allowable values:	yes: G2 assumes that the G2 Gateway user code handles the return of data to G2, without explicit requests from G2. no: G2 Gateway continuously reads a queue of requests for data from G2.
Default value:	no
Notes:	See External-System-Has-a-Scheduler Attribute.

GSI Interface Attributes

Attribute	Description
poll-external-system-for-data	Determines whether the G2 Gateway bridge receives unsolicited data from the external system by executing the callback function *gsi_g2_poll()* every cycle.
Allowable values:	yes: The G2 Gateway bridge calls *gsi_g2_poll()* every cycle. no: The G2 Gateway bridge does not call *gsi_g2_poll()*.
Default value:	no This is the recommended setting.
Notes:	You must set this attribute to yes if you want the bridge to receive unsolicited data. For more information, see Poll-External-System-for-Data Attribute.
interval-to-poll-external-system	Controls the polling interval when the poll-external-system-for-data attribute is set to yes.
Allowable values:	Time interval from 1 second to G2's maximum allowable time interval Specify the time interval in the form: *integer* {second[s] | minute[s] | hour[s] | day[s] | week[s] } use default (equivalent to 1 second)
Default value:	use default

GSI Interface Attributes

Attribute	Description
grouping-specification	(Optional) Enables you to group requests for data service using one or more of the identifying attributes of a variable.
Allowable values:	group requests by *attribute* [, *attribute*]... no grouping
Default value:	no grouping
Notes:	If this attribute specifies group requests by an attribute or attributes, G2 uses a single message to request data service for all GSI variables that have the same values for the specified attribute(s). If more than 21 GSI variables have the same values for the specified attributes, G2 requests data service for GSI variables in separate groups of 21. You can specify any identifying attribute or attributes of a GSI variable, including class-qualified attributes. If this attribute specifies no grouping, G2 does nothing to group requests for data service for GSI variables.
identifying-attributes	List of names of attributes whose values together uniquely identify each GSI variable that receives GSI data service through this GSI Interface.
Allowable values:	*attribute* [, *attribute*] ... none
Default value:	none
Notes:	You can specify class-qualified attribute names in this attribute. See the section Identifying-Attributes Attribute.

GSI Interface Attributes

Attribute	Description
remote-process-initialization-string	String passed from G2 to the `gsi_initialize_context()` function in the G2 Gateway bridge user code, when G2 directs the G2 Gateway bridge to initialize itself.
Allowable values:	"*series-of-gensym-character-set-characters*"
Default value:	"" (empty string)
Notes:	The maximum length of this string is 65,535 characters. See the section Remote-Process-Initialization-String Attribute.

GSI Interface Attributes

Attribute	Description
interface-timeout-period	Specifies how long G2 waits for a response after sending a request to the G2 Gateway bridge. The following three timeout intervals apply to G2-to-G2 and GSI interfaces: **1** Establish a connection. **2** Initialize the connection. **3** Wait for a response. This attribute specifies the timeout period for the first and third intervals. The interface-initialization-timeout-period attribute applies to the second interval. In addition, if the poll-external-system-for-data attribute is set to yes, this attribute specifies how long the G2 Gateway bridge waits for calls to `gsi_g2_poll()` to complete. If the call to `gsi_g2_poll()` does not complete within the specified interval, the bridge sets the gsi-interface-status value of this GSI interface to -1. If the poll-external-system-for-data attribute is set to no, this attribute has no meaning after the connection is established. If you specify use default for this attribute, G2 uses the default time interval of 10 seconds.
Allowable values:	Time interval from 1 second to G2's maximum allowable time interval Specify the time interval in the form: *integer* {second[s] | minute[s] | hour[s] | day[s] | week[s] } use default (equivalent to 10 seconds)
Default value:	use default
Notes:	For more information, see Interface-Timeout-Period Attribute.

GSI Interface Attributes

Attribute	Description
interface-initialization-timeout-period	Specifies how long G2 waits to initialize a connection using Gensym (ICP) protocols. The following three timeout intervals apply to G2-to-G2 and GSI interfaces: **1** Establish a connection. **2** Initialize the connection. **3** Wait for a response. This attribute applies to the second interval. The interface-timeout-period attribute specifies the timeout period for the first and third intervals.
Allowable values:	Possible values are: • An integer specifying some number of seconds • unlimited: the initialization interval never times out • use default: the interface-initialization-timeout-period is the same as the interface-timeout-period
Default value:	unlimited, which specifies that the initialization interval never times out

GSI Interface Attributes

Attribute	Description
gsi-interface-status	Status of the connection between this G2 and the G2 Gateway bridge specified in this item's Gsi-connection-configuration attribute. This attribute is automatically updated by G2 after each transmission between G2 and the G2 Gateway bridge.
Allowable values:	2 (ok) 1 (in transition) 0 (inactive) -1 (timeout) -2 (error) For more information about the meaning of these values, see Identifying the Status of a Connection.
Default value:	0 (zero)
Notes:	This attribute is read-only.
gsi-interface-is-secure	Whether the GSI interface is a secure connection.
Allowable values:	yes | no
Default value:	no
Notes:	This attribute is read-only.

GSI Interface Attributes

Attribute	Description
interface-warning-message-level	Sets the severity level for error and warning messages that G2 provides for the interface object.
Allowable values:	0 to 3: Level 0 is the lowest severity level, and provides the least error information. Increasing the warning message level causes G2 to provide more information about errors and failures that are otherwise only detectable through the value of the gsi-interface-status attribute. Messages are posted to the Operator Logbook by default. For example, when the warning message level is at 0 or 1, a failure to connect to a bridge causes the gsi-interface-status to change to -2 (Error), but no information is made available about why the failure occurred. If the warning message level were set to 3 and the same connection failure occurred, G2 would post a message describing why the connection failed.
Default value:	The value of interface-warning-message-level defaults to warning message level. This value causes the Interface-warning-message-level to take on the value of the Warning-message-level attribute in the Debugging Parameters system table.

GSI Interface Attributes

Attribute	Description
disable-interleaving-of-large-messages	Controls whether G2 Gateway interleaves (changes the transmission order of) message packets.
Allowable values:	yes: G2 Gateway transmits messages without interleaving, preserving their transmission order. With this setting, overall performance can suffer when the messages have very different lengths, because many short messages may have to wait for one long message to complete. no: G2 Gateway transmits messages with interleaving, which reorders the message packets so that large messages do not lock out smaller messages during large message transmission.
Default value:	no

Note The notes, item-configuration, and names attributes of a GSI interface are common to all G2 items. For information about these attributes, see the *G2 Reference Manual*.

Names Attribute

You must specify one or more names for the GSI interface in the names attribute. An unnamed GSI interface cannot support communication between a G2 Gateway bridge and a G2 KB.

G2 objects refer to a named GSI interface as follows:

- For a GSI variable to receive data service through a G2 Gateway bridge, the variable's gsi-interface-name attribute must name the activated GSI interface that, in turn, identifies the G2 Gateway bridge that provides the actual data service.
- In an item that can execute actions (such as rules, procedures, and methods), a start or call action invokes a function in the G2 Gateway bridge through a remote procedure call. The start or call action must specify which bridge by including the across *gsi-interface-object* phrase, as described in the *G2 Reference Manual*.
- For a GSI message server item to be the source or destination of a text message sent between a G2 and a G2 Gateway bridge, its gsi-interface-name attribute

must specify an activated GSI interface that, in turn, identifies an active connection to a running G2 Gateway bridge.

Note An unnamed GSI interface created by the API function *gsi_initiate_connection()* can be used for remote procedure calls, but not for data service for GSI variables. For information about his function, see *gsi_initiate_connection*.

Gsi-Connection-Configuration Attribute

This attribute contains an expression that identifies a running G2 Gateway bridge. In order for G2 to establish a connection to the specified G2 Gateway bridge, the gsi-connection-configuration attribute must match:

- The name of the machine that runs the G2 Gateway bridge, or
- The IP address of the machine that runs the G2 Gateway bridge, or
- localhost, a special hostname that represents the local machine, and
- The TCP/IP port number displayed by the G2 Gateway bridge on the command line where the bridge is started, and
- Whether the connection is secure.

For example, the following expression specifies a G2 Gateway bridge process with port number *22041* that runs on the local computer (the computer on which G2 is running) and to which this G2 connects using the TCP/IP protocol:

```
tcp-ip host "localhost" port-number 22041
```

Note The TCP/IP port number of a G2 Gateway bridge can range from 3001 to 29,999, but cannot be in use by another process on that computer.

A G2 Gateway bridge process's TCP/IP port number can be specified or accepted as follows:

- Specified as a command line argument (see Building and Running a G2 Gateway Bridge.)
- Specified within the user code function *gsi_get_tcp_port()*
- Accepted by default, using TCP/IP port number 22041

External-System-Has-a-Scheduler Attribute

The external-system-has-a-scheduler attribute affects when the callback function *gsi_get_data()* is invoked.

G2 Gateway calls *gsi_get_data()* to respond to requests from G2 for values for GSI variables. For information about how to use *gsi_get_data()*, see Returning Solicited Data to G2.

When the value of external-system-has-a-scheduler is yes, *gsi_get_data()* is called:

- At creation, activation, or enabling of a GSI variable, provided that the variable has a default-update-interval other than none.
- At the request of a rule, a readout table, or a collect data statement within a G2 procedure, if the validity-interval of the GSI variable has expired.
- Unconditionally as a result of an update action within a G2 rule or procedure.

When the value of external-system-has-a-scheduler is no, *gsi_get_data()* is called: under the same conditions, with one addition. It is also called at the expiration of a GSI variable's default-update-interval, if this interval is other than none.

Regardless of the setting of external-system-has-a-scheduler attribute, a GSI variable is registered in the following circumstances:

- If the default-update-interval of a GSI variable is a value other than none, G2 sends to G2 Gateway a request to register and update the variable when the GSI interface is activated.
- If the default-update-interval of a GSI variable is none, the variable is registered the first time the G2 set or update action is performed on it.

When a GSI variable is registered (and again whenever a set or update action is subsequently performed on the variable), G2 sends the default-update-interval value of the variable to the G2 Gateway bridge. The value is stored in data structures that your user code can access through API functions. Your bridge user code can use this value when it communicates with an external system, or it can ignore the value, as your G2 Gateway application requires. For information about how your user code can access the default update interval values stored in the data structures, see G2 Gateway Data Structures.

Regardless of the setting of the external-system-has-a-scheduler attribute, G2 can always explicitly request values for GSI variables from the G2 Gateway bridge.

Poll-External-System-for-Data Attribute

If this attribute is set to yes, the G2 Gateway bridge obtains unsolicited data by executing the *gsi_g2_poll()* callback at regular intervals. For information about how to use this callback, see *gsi_g2_poll*.

If this attribute is set to no, the G2 Gateway bridge does not call the *gsi_g2_poll()* callback.

Interval-to-Poll-External-System Attribute

This attribute controls the polling interval when the poll-external-system-for-data attribute is set to yes. By default, polling happens every second, but you can change this by setting the value of the interval-to-poll-external-system attribute. Allowable values are any time interval from 1 second to G2's maximum allowable time interval.

When the poll-external-system-for-data attribute is yes and the interface is connected, G2 sends a message to G2 Gateway at intervals specified by the interval-to-poll-external-system attribute. If G2 Gateway is running (if it is in `gsi_start()` or in `gsi_run_loop()`), it reads the message, sends a reply back to G2, and then calls the `gsi_g2_poll()` callback. G2 does not send any further poll messages until it receives a reply from G2 Gateway.

If G2 does not receive a reply within the interval specified by the interface-timeout-period attribute, then:

- The gsi-interface-status attribute of the gsi-interface object is changed to -1.
- G2 invokes any rules that check for a -1 value for this attribute.

During the interval that the gsi-interface-status remains at -1, the connection remains open. If G2 receives a reply at some later time, the value of the gsi-interface-status attribute is set to 2.

Identifying-Attributes Attribute

This attribute lists class-specific attributes of GSI variable classes that provide a unique identifier for each GSI variable. The values of the identifying attributes of each GSI variable must distinguish that GSI variable from all other GSI variables in the KB.

G2 and the G2 Gateway bridge use the identifying attribute values of a GSI variable to maintain a one-to-one mapping between that GSI variable and the source of data in an external system.

When a GSI variable class is defined with one identifying attribute, G2 and the G2 Gateway bridge assume that each GSI variable of that class in the KB has a unique value in this attribute.

When a GSI variable class is defined with more than one identifying attribute, G2 and the G2 Gateway bridge assume that each GSI variable of that class within the KB has a unique *combination* of values for these attributes.

Note Data should not be returned to identifying attributes of GSI variables. Another way of saying this is that the identifying attributes of a GSI variable should not themselves be GSI variables. If the G2 Gateway bridge returns data to one or more of a GSI variable's identifying attributes, it causes the GSI variable to be deregistered and then reregistered with the bridge.

The identifying-attributes attribute of a GSI interface lists the aggregate of the identifying attributes of all classes of GSI variables that use this particular GSI interface. If more than one class of GSI variables uses a particular GSI interface, individual GSI variables of any of these classes will have only some of the identifying attributes listed in the identifying-attributes attribute of the GSI interface.

For information about how to specify identifying attributes, see Defining Identifying Attributes.

Remote-Process-Initialization-String Attribute

This attribute specifies a string that G2 passes at startup time to the *gsi_initialize_context()* function in the G2 Gateway bridge user code.

In the GSI bridge user code, you can add code to the *gsi_initialize_context()* callback function that uses this initialization string during the initialization of the G2 Gateway bridge process. For example, the initialization string can specify a list of run-time parameters for the bridge process or can specify the pathname for a file that the bridge process should open during bridge initialization.

Interface-Timeout-Period Attribute

For systems using the TCP/IP protocol, this attribute indicates how long G2 should wait for a response after sending a request to the G2 Gateway bridge. G2 uses this value when connecting to the G2 Gateway bridge process and after each transmission that requests data from the bridge process. G2 measures a timeout interval from the most recent time that data was received.

Three timeout intervals apply to G2-to-G2 and GSI interfaces. These set the maximum time to:

1. Establish a connection at the TCP/IP network level.
2. Initialize the connection using Gensym (ICP) protocols.
3. Wait for a response to a message sent by G2.

The interface-timeout-period sets the timeout period for the first and third intervals. The interface-initialization-timeout-period attribute sets the timeout period for the second interval.

To specify all three timeouts to be the same:

1. Set the interface-timeout-period attribute to the desired interval.
2. Use the default setting for the interface-initialization-timeout-period attribute.

The interface-timeout-period time interval begins at the time that the request is made. If the G2 process cannot obtain the status of the G2 Gateway bridge process, the GSI interface times out. If tracing is at level 3, G2 reports the error on its Operator Logbook.

If you specify use default for this attribute, G2 uses the default time interval of 10 seconds.

Interface-Initialization-Timeout-Period Attribute

This attribute specifies how long G2 waits to initialize a connection using Gensym (ICP) protocols. G2 measures a timeout interval from the most recent time that data was received.

The possible values for interface-initialization-timeout-period are:

- An integer specifying some number of seconds.
- unlimited: the initialization interval never times out.
- use default: the interface-initialization-timeout-period is the same as the interface-timeout-period.

The default value is unlimited, which specifies that the initialization interval never times out.

Three timeout intervals apply to G2-to-G2 and GSI interfaces. These set the maximum time to:

1. Establish a connection at the TCP/IP network level.
2. Initialize the connection using Gensym (ICP) protocols.
3. Wait for a response to a message sent by G2.

The interface-initialization-timeout-period attribute sets the timeout period for the second interval. The interface-timeout-period sets the timeout period for the first and third intervals.

To specify all three timeouts to be the same:

1. Set the interface-timeout-period attribute to the desired interval.
2. Use the default setting for the interface-initialization-timeout-period attribute.

To specify a different timeout for the initialization interval:

➔ Set the interface-initialization-timeout-period attribute to the desired interval, or to unlimited if no timeout is desired.

Disable-Interleaving-of-Large-Messages Attribute

This attribute controls whether G2 Gateway interleaves (changes the transmission order of) message packets.

If set to no (the default):

- G2 transmits messages in packets. A large message occupies several packets. A small message occupies a single packet.
- When more than one message requires transmission across an interface, G2 interleaves the packets that constitute the messages.

As a result of this behavior, large messages do not lock out smaller messages during large message transmission, which can improve performance when messages of different lengths are transmitted over the same interface.

However, a shorter message that begins transmitting after a longer message begins can finish transmitting before the longer message finishes. This reverses the effective transmission order of the messages, because a message cannot be acted on until its transmission is complete. Messages of the same length can also be reordered if network errors require packet retransmission.

If the original transmission order was significant, reordering of interleaved messages can cause errors. For example, procedures that should have been invoked in one order could be executed in another.

To prevent message interleaving from reordering messages:

➔ Set the disable-interleaving-of-large-messages attribute of the interface object to yes before opening the connection.

Messages then transmit without interleaving, preserving their transmission order. However, overall performance can suffer when the messages have very different lengths, because many short messages may have to wait for one long message to complete.

Changing disable-interleaving-of-large-messages does not affect an existing connection.

To change the interleaving of an existing connection:

1 Set disable-interleaving-of-large-messages to yes or no.

2 Conclude the connection into itself as a structure.

The conclude closes and reopens the connection, which then reflects the new attribute value. For example, to reconnect a G2-to-G2 interface object:

```
conclude that the icp-connection-specification of g21 = structure
(network-transport : the symbol tcp-ip, hostname: "london", port: 1114)
```

Initializing Attributes in Subclasses

There are some system-defined attributes that you can initialize for a GSI interface subclass and some that you can initialize for a G2-to-G2 interface. The following table lists the system-defined attributes that you can initialize for each:

Attribute	G2-to-G2	G2 Gateway
external-system-has-a-scheduler		✓
poll-external-system-for-data		✓
grouping-specification		✓
remote-process-initialization-string		✓
interval-to-poll-external-system		✓
identifying-attributes	✓	✓
interface-warning-message-level	✓	✓
interface-timeout-period	✓	✓
interface-initializiation-timeout-period*	✓	✓
disable-interleaving-of-large-messages*	✓	✓
attribute-displays	✓	✓
stubs	✓	✓

Identifying the Status of a Connection

The gsi-interface-status attribute of each GSI interface indicates:

- Whether a connection exists between G2 and the specified G2 Gateway bridge.
- If a connection exists, the last event that took place over the connection.

G2 automatically updates the value of the gsi-interface-status attribute. If a connection exists between G2 and the specified G2 Gateway bridge, G2 updates the value of the gsi-interface-status attribute whenever a network message is passed between G2 and the G2 Gateway bridge.

When a gsi-interface object is activated and there is no bridge to connect to (i.e., bridge connection failure), the value of the gsi-interface object gsi-interface-status attribute depends on whether the gsi-connection-configuration attribute has a connection value or is none:

- If the gsi-connection-configuration attribute has a value, the value of gsi-interface-status is 1 (In transition).
- If the gsi-connection-configuration attribute is none, the value of gsi-interface-status is 0 (Inactive).

The following table lists and describes the possible values of the gsi-interface-status attribute:

Possible Values of Gsi-Interface-Status Attribute

-2 (error)	Following a connection attempt (gsi-interface-status value is 1), a value of -2 (Error) indicates a connection failure. Possible causes: • No bridge process at the port specified. • The bridge process rejected the connection by returning GSI_REJECT. After a connection has been made, a value of -2 (Error) indicates that an error condition occurred, and the connection has broken between G2 and the bridge process.
-1 (timeout)	The G2 process has not heard from the bridge process within the interface-timeout-period specified for the GSI interface. This status may also indicate that a communications overload has occurred. This status can occur only when the poll-external-system-for-data attribute of the GSI interface is set to yes.
0 (inactive)	The GSI interface is either disabled or inactive. The GSI interface is inactive when it is either on an inactive workspace, has no name, is otherwise not ok, or G2 is not running. When the GSI-connection-configuration attribute value is none, activation cannot result in a connection.

Possible Values of Gsi-Interface-Status Attribute

1 (in transition)	This state indicates that G2 is either completing or breaking a connection to G2 Gateway. In either case, no G2 data service, RPC action, or message service can be done on the GSI interface. If a GSI interface was in state 0, and changed to 1, this indicates that G2 is negotiating a new G2 Gateway connection. The next state will be 2 if successful or -2 if the connection fails. When a GSI interface is in state 2, and is deactivated, it changes to state 1 until it is certain that the G2 Gateway bridge has received and acknowledged the connection shutdown. It will time out after 15 seconds.
2 (ok)	The connection between the G2 process and the bridge process is successful and being maintained.

Triggering Rules when the Interface Status Changes

Changes to the gsi-interface-status attribute value of a GSI interface that occur just before the interface becomes active can trigger rules.

Consistency of Status between a GSI Interface and Other G2 Objects

There can be a gap between the moment when G2 updates the status of a GSI interface and when G2 updates the current value of a GSI variable that receives G2 Gateway data service through that GSI interface. For this reason, your G2 knowledge base should *not* be designed to assume that if the gsi-interface-status of a GSI interface is not OK, then the statuses of the variables that rely upon that GSI interface for G2 Gateway data service are also not OK.

For example, when a G2 Gateway bridge is heavily loaded, it is possible for a GSI variable in the G2 knowledge base to fail to receive a value, even though the G2 Gateway bridge is still running. The gsi-variable-status attribute of the GSI variable might contain the value 0 (indicating that the data object in the G2 Gateway bridge memory was most recently known to be OK), although the gsi-interface-status attribute of the variable's associated GSI interface has the value -1, indicating that the G2's connection to the G2 Gateway bridge has timed out.

In this case, it is not necessary to break and reestablish the connection between G2 and the G2 Gateway bridge. Instead, you can design your G2 knowledge base to allow additional time for the G2 Gateway bridge to return a new value for the variable when the gsi-interface-status attribute of a GSI interface receives a new value that is not OK.

Determining Whether the Interface is Secure

The gsi-interface-is-secure attribute determines whether the interface was started with the secure yes option in the gsi-connection-configuration.

For more information, see secure.

Updating GSI Interface Attributes While the KB is Running

You can apply edits to only two attributes of a GSI interface while the G2 knowledge base is running: grouping-specification and poll-external-system-for-data. These attributes are updated *immediately* after you edit them.

Edits that you make to any other attributes of a GSI interface do not take effect until you reset and start the G2 knowledge base.

Activating and Deactivating a GSI Interface

When a fully configured GSI interface becomes activated, G2 attempts to establish a network connection to the G2 Gateway bridge identified in the gsi-connection-configuration attribute of the GSI interface.

What fully configured means depends on which kinds of objects in the G2 knowledge base refer to that GSI interface. That is, it depends on whether a GSI variable, or GSI Message Server, or executable item (i.e. one containing a start or call action that invokes a G2 Gateway bridge function through a remote procedure call), or any combination of these refers to the GSI interface.

Note An enabled item becomes activated when its parent workspace or superior object becomes activated. See the *G2 Reference Manual* for information about how to activate workspaces.

Successful Activation

A GSI interface becomes activated in the following ways:

- Explicitly, as a result of programmatically activating either the GSI interface's workspace or another workspace that is superior to the object in the knowledge base's workspace hierarchy.
- Implicitly, as a result of starting the G2 knowledge base.
- Through user actions that enable or disable the GSI interface itself.

If G2 successfully establishes a network connection with a particular G2 Gateway bridge process, it automatically sets the GSI interface's gsi-interface-status attribute to indicate that the connection is open and functioning. If G2 fails to establish the connection, it automatically sets the gsi-interface-status attribute to

indicate that the attempt failed. See Identifying the Status of a Connection for a list of the possible values of the gsi-interface-status attribute.

The G2 knowledge base can use the value of the gsi-interface-status attribute in its processing. For example, given that G2 automatically updates the value of the gsi-interface-status attribute of any GSI interface whenever any data is exchanged between the G2 and the G2 Gateway bridge process, you can include a whenever rule that is triggered each time the gsi-interface-status attribute of any GSI interface receives a value, as follows:

```
for any custom-gsi-interface C
    whenever the gsi-interface-status of C receives a value
     and when the gsi-interface-status of C = -1
        then in order invoke timeout rules
          and inform the operator that
          "The GSI interface [the name of C] has timed out."
```

Hint Concluding a new value to a GSI interface's gsi-connection-configuration attribute or identifying-attributes attribute causes G2 to break the connection (if it exists) with the G2 Gateway bridge process and then immediately to attempt to make a new connection. You can use this technique to perform operations such as resetting and reestablishing communication with a G2 Gateway bridge process or switching the connection to another process. For example, to programmatically toggle the connection of a GSI interface perform the following action: change the text of the gsi-connection-configuration of MYIO to the text of the gsi-connection-configuration of MYIO. Note that the value does not necessarily have to be a different value.

Unsuccessful Activation

After G2 activates a GSI interface, G2 attempts to use it to establish a network connection with a G2 Gateway bridge process. If the values in the GSI interface's attributes are incomplete or invalid, G2 places one or more messages to that effect on the G2's Operator Logbook. The Operator Logbook is described in the *G2 Reference Manual*.

Deactivating a GSI Interface

A GSI interface becomes deactivated in these ways:

- Explicitly, as a result of programmatically deactivating either the GSI interface's workspace or another item that is superior to the object in the knowledge base's workspace hierarchy.
- Implicitly, as a consequence of resetting the G2 knowledge base.
- Explicitly, when a user disables the GSI interface itself.

When you deactivate an activated GSI interface that refers to an established network connection between the G2 knowledge base and a G2 Gateway bridge process, G2 automatically breaks that network connection.

Note Breaking a network connection between a G2 and a G2 Gateway bridge process does *not* kill the G2 Gateway bridge process and does *not* cause it to stop executing, unless the bridge programmer has arranged for this to happen explicitly, in the bridge user code.

Deleting a GSI Interface

If a GSI interface is deleted, the connection is lost, but other connections are not affected. The bridge does not exit unless the user has programmed that behavior.

Configuring GSI Variables in the KB

To enable your G2 knowledge base to read values from and write values to data points in an external system, you must create **GSI variables**. A GSI variable is a G2 variable that receives data service from a G2 Gateway bridge.

You map each GSI variable to a data point in an external system. Your G2 knowledge base then reads from and writes to the external data points through the corresponding GSI variable. Each GSI variable must refer to a GSI interface.

To create GSI variables that your G2 knowledge base can use to read from and write to data points in an external system, you must:

1. Define a class of GSI variables.
2. Create an instance of this class for each external data point that your G2 application needs to read from or write to.
3. Edit the attributes of each GSI variable to specify the GSI interface that it references to communicate with the G2 Gateway bridge process, and to specify other characteristics of the GSI variable.

Defining GSI Variable Classes

You can create different application-specific classes of GSI variables to represent different types of data exchanged with the G2 Gateway bridge and with the G2-based application's external system.

The class definition for GSI variables can also be a subclass of any class that inherits from these standard variable classes.

To define a GSI variable class:

1 Create a class definition for your GSI variables by selecting:

KB Workspace > New Definition > class-definition > object-definition

An icon representing the new class definition appears on the workspace where you are creating the class definition.

2 Open the menu for the class definition and select table.

The attribute table of the class definition appears.

3 In the attribute table, specify a name for your new class under class-name.

4 Under direct-superior-classes, specify one of the following standard variable classes: integer-variable, float-variable, quantitative-variable, logical-variable, symbolic-variable, text-variable, or sensor.

5 Also under direct-superior-classes, specify the G2 mixin class gsi-data-service.

A variable must inherit from this class in order to receive data service from a G2 Gateway bridge.

6 Under class-specific-attributes, specify any attributes that you want to add to this class definition.

For more information about defining new classes, see the *G2 Reference Manual*.

Attributes of GSI Variables

You can modify all attributes of GSI variables, including their identifying attributes, while the knowledge base is running (although the user code must be written correctly for it to be effective).

The following table lists attributes that are of particular importance to the behavior of a GSI variable. For information about the attributes of variables that do not relate to G2 Gateway, see the *G2 Reference Manual*.

GSI Variable Attributes

Attribute	Description
validity-interval	The length of time that the last-recorded-value of the variable remains current.
Allowable values:	Any time-interval indefinite
Default value:	supplied
Note:	A validity interval of supplied is not valid for a variable whose data server is GSI data service. You must specify a value for this attribute. For information about how to set the validity interval attribute, see the *G2 Reference Manual*.
data-server	The data server for this variable.
Allowable values:	The symbol gsi-data-server, or any unreserved symbol that is defined in the knowledge base's Data Server Parameters system table as an alias for gsi-data-server. For information about how to define an alias for a standard G2 data server, see "Data Server Parameters" under "System Tables" in the *G2 Reference Manual*.
Default value:	gsi-data-server
Note:	To prototype GSI variables in your KB, you can specify the G2 Inference Engine as their data server, and use formulas specified in formula attributes of the GSI variables to compute values for the variables. The formula attribute of a GSI variable is not useful for any purpose other than prototyping the variable. To provide a GSI variable with data service from a G2 Gateway bridge, you *must* specify GSI data server as the data server of the variable.

GSI Variable Attributes

Attribute	Description
default-update-interval	Specifies a regular time interval at which G2 obtains a value for this variable.
Allowable values:	Any non-negative number none
Default value:	none
gsi-interface-name	Name of the GSI interface that supports data service for this variable. Must be specified to allow this variable to exchange data with the G2 Gateway bridge. **Note**: If you modify this attribute, you must reset and start the knowledge base for the changes to take effect.
Allowable values:	Any unreserved symbol that names a GSI Interface none
Default value:	none
Notes:	When you create GSI variables programmatically, this should be the last attribute that you set, because setting this attribute causes the GSI variable to be immediately eligible for mapping to an external data point. You may want to delay mapping until all the attributes of the GSI variable are set.
gsi-variable-status	Status of the data point or variable in an external system that the G2 Gateway bridge maps to this GSI variable
Allowable values:	Integer value of 0 (zero) or higher
Default value:	0 (zero)
Notes:	See Identifying the Status of the GSI Variable.

Defining Identifying Attributes

A GSI interface can designate as many as six different attributes of a GSI variable class as **identifying attributes** for variables of that class. The values of a GSI variable's identifying attributes must distinguish it from all other variables in the KB. The identifying attributes of a GSI variable provide a unique identification for the variable, which G2 Gateway needs in order to map it to a data point in an external system.

The identifying attributes of a GSI variable must be *user-defined attributes* – either class-specific-attributes of the GSI variable's own class, or class-specific-attributes that the GSI variable inherits from a superior class. Class-qualified attribute names can be used as identifying attributes.

The data type of an identifying attribute can be either a G2 value (an integer, float, truth-value, symbol, or text) or a G2 parameter. An identifying attribute cannot be an array, list, variable, or any other G2 item that is not a parameter.

For more information about defining a variable class, see the *G2 Reference Manual*.

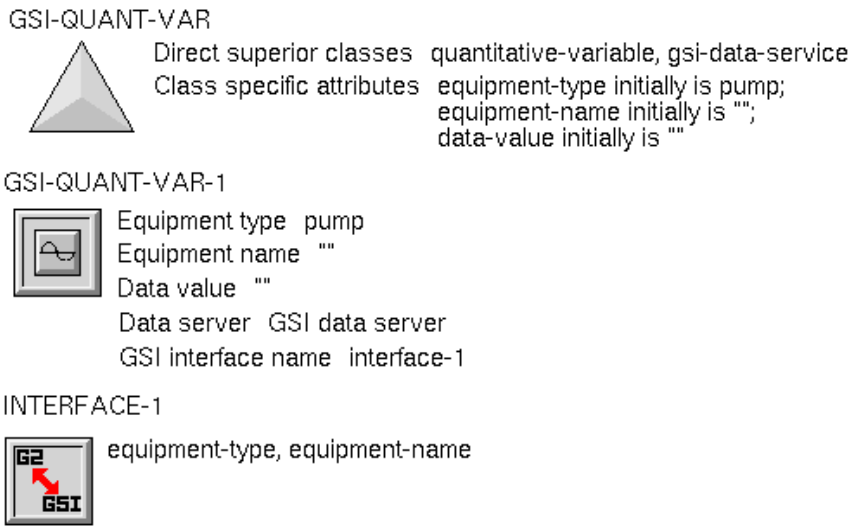
You list the identifying attributes of a GSI variable class in the identifying-attributes attribute of a GSI interface. Each instance of a GSI variable class must use the GSI interface that lists its identifying attributes. The identifying-attributes attribute of a GSI interface contains the aggregate of the identifying attributes of all GSI variable classes that use that GSI interface.

G2 communicates the values of identifying attributes to the G2 Gateway bridge whenever a GSI variable is registered or reregistered with the G2 Gateway bridge.

Whenever a value of an identifying attribute of a GSI variable is changed, that variable is deregistered and reregistered. Updating the value of the identifying-attributes attribute of a GSI interface causes G2 to request the G2 Gateway bridge to deregister all registered variables that use this GSI interface, then to register the set of variables that use the new set of identifying attributes listed in the GSI interface.

Caution Because a GSI variable is reregistered whenever the values of any of the identifying attributes are changed, do not return values, directly or indirectly, to identifying attributes from the G2 Gateway bridge user code. This would result in unnecessary exchanges of data between the G2 and the G2 Gateway bridges.

The following figure illustrates a GSI variable class definition named gsi-quant-var and a variable of this class named gsi-quant-var-1:



In the preceding figure, the GSI interface interface-1 specifies the equipment-type and equipment-name attributes of the gsi-quant-var class as identifying attributes for variables of this class. Note that the data-value attribute is not suitable for use as an identifying attribute, because the G2 Gateway bridge sends values to this attribute.

Identifying the Status of the GSI Variable

GSI variable status codes indicate the status of the external data point or variable that is mapped to a GSI variable in a G2 KB. The status code is registered in the gsi-variable-status attribute. Its value is set by G2 Gateway, depending on the value of the `status` field set within the user code for this variable. This attribute is read-only.

The possible values for the gsi-variable-status attribute of a GSI variable are:

- 0 (zero): the external variable is OK
- 1 to 5: reserved for use by G2 Gateway
- 6 and higher: Values set in the G2 Gateway bridge's user code, which can represent status messages from the bridge to G2 about this GSI variable's corresponding external data point

You can create whenever rules that depend on the value of this attribute. For example, G2 invokes the following rule whenever the gsi-variable-status of any GSI variable receives a value:

```
for any GSI variable G
    whenever the gsi-variable-status of G receives a value then
        inform the operator for the next 4 seconds that "The status of [the
          name of G] is [the gsi-variable-status of G]."
```

For more information about whenever rules, see the *G2 Reference Manual*.

Specifying Initial Values for GSI Variables

It is good practice to provide initial values for all GSI variables, either by:

- Specifying an initial value for all the GSI variables of each GSI variable class. To do this, specify the initial value in the attribute-initializations attribute of the GSI variable class definition. For example, you can specify the following in the attribute-initializations attribute of a class of integer variables:

  ```
  initial value for integer-variable: 1
  ```

- Providing a value for the GSI variable when G2 registers the variable for data service, through a call to *gsi_return_values* or related function in the callback function *gsi_receive_registration()*. For information about this function, see *gsi_receive_registration*.

If G2 is unable to obtain a value for a GSI variable within the amount of time specified in the timeout-for-variables parameter of the Inference Engine Parameters system table, G2 repeatedly attempts to obtain a value for the variable at the interval specified by the retry-interval-after-timeout parameter of the Inference Engine Parameters system table. This can result in repeated calls to the callback function gsi_get_data(), at the specified interval specified by retry-interval-after-timeout.

Creating and Configuring GSI Message Servers

A G2 KB can send text messages to a G2 Gateway bridge using a **GSI message server**. A GSI message server is a G2 object that inherits from the G2 mixin class gsi-message-service and at least one G2 class from which you can create instances.

To send the text message to the external system, G2 runs an inform action on the GSI message server. The inform action specifies the text message that G2 sends to the G2 Gateway bridge.

When the G2 Gateway bridge receives the text message, it calls the *gsi_receive_message()* callback function, which can pass the text that it receives from G2 to an external system or handle the message in any other way that your application requires. For information about *gsi_receive_message()*, see Callback Functions.

To create and configure a GSI message server, follow these steps:

1. Create a message server class definition by selecting:

   KB Workspace > New Definition > object-definition

   An icon representing a class definition appears on the workspace.

2. Open the menu for the class definition and select table. The attribute table of the class definition appears.

3. In the attribute table, specify a name for the message server class under class-name.

4. Also in the attribute table, specify the following classes under direct-superior-classes:

   - The gsi-message-service mixin class.
   - A G2 class from which you can create instances. This class can inherit from the object, message, or connection standard G2 classes. For example: text-variable.

   A GSI message server class can optionally include the gsi-data-service mixin class among its direct superior classes, but is not required to.

5. Close the attribute table of the message server class definition.

6. Create a GSI message server.

   To do this, select create instance from the menu of the message server class definition. The GSI message server appears on the workspace near the icon for the message server class definition.

7. In the attribute table of the GSI message server, specify a name for the message server under names.

8. Also in the attribute table of the GSI message server, specify the name of a gsi-interface under gsi-interface-name. Use the name of the GSI interface that represents the connection between the G2 that sends the text message and the G2 Gateway bridge that you intend to receive it.

Attributes of a GSI Message Server

The following table summarizes the class-specific attributes of a GSI message server:

GSI Message Server Attributes

Attribute	Description
gsi-interface-name	Name of the GSI interface that supports data service for this GSI message server. You must specify a GSI interface to enable the GSI message server to send data to the G2 Gateway bridge.
Allowable values:	Any unreserved symbol that names a GSI Interface none
Default value:	none
Notes:	Setting the attribute causes the variable to be immediately eligible for mapping. To delay mapping until all the attributes are set properly, set this attribute last.
data-server-for-messages	The symbol gsi-data-server or any symbol defined as an alias for the G2 standard data server gsi-data-server.
Allowable values:	The symbol gsi-data-server, or any unreserved symbol that is defined in the knowledge base's Data Server Parameters system table as an alias for gsi-data-server. For information about how to define an alias for a standard G2 data server, see "Data Server Parameters" under "System Tables" in the *G2 Reference Manual*.
Default value:	gsi-data-server

These attributes are contributed by the G2 standard gsi-message-service mixin class.

Running an Inform Action on a GSI Message Server

To send a text message to a G2 Gateway bridge through a GSI message server, a G2 KB runs the inform action on the GSI message server. In the inform action, you specify the text of the message that you want to send in double quotation marks.

For example, the following rule runs the inform action on a GSI message server named gsi-msg-var-1, and sends the text message: "The temperature of vat-1 is too high."

```
If the temperature of gauge-1 is too-high then
    inform gsi-msg-var-1 that "The temperature of vat-1 is too high."
```

Similarly, the following rule runs the inform action on a GSI message server named operator-2:

```
if product-ph < 7 and hydroxide-feed-rate = max-hydroxide-feed-rate then
    inform operator-2 that "Product is too acid to neutralize. pH =
    [ product-ph ] with maximum hydroxide input."
```

If the pH of the product is less than 7 (for example, 5.72), and the system is unable to feed the hydroxide any faster, G2 invokes this rule and sends the text message:

```
"Product is too acid to neutralize. pH = 5.72 with maximum hydroxide input."
```

to the G2 Gateway bridge through the GSI interface used by the GSI message server named operator-2.

For information about how to invoke the inform action, see the *G2 Reference Manual*.

3

Preparing the Bridge User Code

Describes how to organize and code the customized portion of the G2 Gateway bridge.

Introduction

The **user code** is the part of a G2 Gateway bridge that acts as an interface between the bridge and G2. This chapter describes the structure of your user code, the tasks that it can perform, and the functions, data structures, and data types that it uses to perform these tasks.

This chapter does *not* describe the G2 Gateway bridge code that you must write to provide an interface between your G2 Gateway bridge and external systems such as databases or PLCs. For information about how to write this part of your bridge code, see the documentation for your external system.

Language Support for G2 Gateway User Code

Gensym supports user code development in the C programming language. If you use any other language to develop your bridge, you must provide the appropriate links to G2 Gateway C routines called by your user code functions.

The following discussion of G2 Gateway user code assumes that you are developing the code in C.

Single-Threaded Programming and Reentrancy

Applications developed with GSI 4.0, G2 Gateway 5.0, and later versions are designed to run in single-threaded programming environments.

The G2 Gateway library of API functions is not thread-safe; that is, it cannot be used by programs running under a thread package that supports multiple threads of control. Attempts to run a G2 Gateway application in a multi-threaded programming environment will cause serious errors.

A G2 Gateway program is not re-entrant; this is, it cannot be executed simultaneously by multiple threads of control. However, you may have any number of G2 Gateway processes running simultaneously.

Components of G2 Gateway User Code

You develop the user code part of your G2 Gateway bridge by:

- Completing the code for predeclared **callback functions** provided with G2 Gateway. G2 Gateway callback functions form the basis of your user code. For information about these functions, see Callback Functions.

  G2 Gateway calls callback functions automatically in response to events in G2, such as requests by G2 to get or set the values of data points in an external system. You do not need to invoke callback functions explicitly from anywhere in your user code, and should not attempt to do this.

- Writing functions that G2 can invoke as remote procedures. For information about how to write these functions, see Remote Procedure Calls.
- Writing user functions to perform computations or any customized processing required by your G2 Gateway application.

Callback functions, remote procedures, and user functions can invoke API functions provided with G2 Gateway to perform a wide variety of tasks, such as accessing data received from G2, supporting data service of G2 variables, supporting remote procedures, and error handling. For information about API functions, see API Functions.

You can write user code that handles all interactions between G2 and G2 Gateway through remote procedure calls. For information about how to do this, see Developing a Bridge Using Only Remote Procedure Calls.

For information about how to compile, link, and run your code, see Building and Running a G2 Gateway Bridge.

Structure of G2 Gateway User Code

As in any C program, `main()` is the first function in the user code portion of your bridge to be executed.

From within `main()`, you can call G2 Gateway API functions and any other functions available to a C or C++ program. Note, however, that you do not call G2 Gateway callback functions directly from within `main()`. G2 Gateway calls the callback functions automatically in response to network events on a connection to a G2 KB. Instead, you supply code that performs tasks for your application for the time the function is called.

Contents of the main() Function

The only function that your `main()` function *must* call is `gsi_start()`. This API function initializes G2 Gateway, performs setup operations, and passes control to the API function `gsi_run_loop()`, which establishes the main event processing loop of your G2 Gateway bridge process. For more information about `gsi_start()`, see Using gsi_start().

Your `main()` function can optionally include:

- (In one-cycle mode) Code defining an event processing loop that executes under control of your user code. For information about how to establish a processing loop under control of your user code, see Processing Events through gsi_run_loop().
- Statements that set global GSI variables to specify the version of the G2 Gateway user header file `gsi_main.h` to use, and to ensure that this header

file matches the version of the G2 Gateway object libraries with which you link your user code.

To do this, include the following statements in *main()*:

```
gsi_include_file_major_version = GSI_INCLUDE_MAJ_VER_NUM;
gsi_include_file_minor_version = GSI_INCLUDE_MIN_VER_NUM;
gsi_include_file_revision_version = GSI_INCLUDE_REV_VER_NUM;
```

Note You cannot use the *gsi_include_file_major_version*, *gsi_include_file_minor_version, and gsi_include_file_revision_version* variables with a G2 Gateway that is delivered as a DLL. Thus, if you are using G2 Gateway on WIN32 platforms (in addition to being delivered as three libraries, as before), use the *gsi_set_include_file_version()* function to specify the minor version, major version, and revision.

You can use *gsi_set_include_file_version()* to specify versions on any platform.

The following illustrates the basic structure of your G2 Gateway user code.

Structure of G2 Gateway User Code

What the user codes (API functions and callbacks)	What G2 Gateway does (callbacks)

`main(argc, argv)`
`{`
`  gsi_set_options_from_compile()`
`  /* Only if not present in gsi_main.c. */`
`  ...`
`  gsi_start(argc, argv)`
`  ...`
`  /* User-defined event loop (Only in one-cycle mode) */`
`  for (;;)`
`  {`
`  ...`
`  gsi_run_loop()`
`  ...`
`  /* Other function calls.*/`
`  }`
`}`
 	
`/*Callback functions:*/`
 	
`gsi_set_up()`
`{`
`  ...`
`  /* Invoke API functions to perform application-specific operations */`
`  ...`
`}`
 	
`gsi_get_tcp_port()`
`{`
`  ...`
`  /* Specifies default TCP/IP port number used if port number is omitted from command line */`
`}`
	`gsi_start(argc,argv)`
	`{`
	`  ...`
	`  gsi_set_up()`
	`  ...`
	`  gsi_get_tcp_port()`
	`  ...`
	`  gsi_run_loop()`
	`}`
	
	`gsi_run_loop()`
	`{`
	`  ...`
	`  do {`
	`  ...`
	`/* Accept new connections from G2 */`
	`  ...`
	`/* Invoke callback functions to process events on connection to G2 */`
	`  ...`
	`  } while in continuous mode`
	`}`

Passing Command-Line Arguments to the Bridge Through main()

Your main() function receives two arguments from the operating system's command-line interpreter: *argc* and *argv*:

- *argc* represents the number of command line arguments.
- *argv* is an array of strings in which the command line arguments are stored. *argv[0]* is reserved for the name of your bridge program and *argv[1]* is reserved for the TCP/IP port number. Other elements of *argv* are available to users.

  On systems that use the TCP/IP protocol, if a 0 is passed as the first argument (*argv[1]*), the callback function *gsi_get_tcp_port()* is called to return a default port number. For information about this function, see Callback Functions.

Your *main()* passes the *argc* and *argv* arguments to the API function *gsi_start()*, which starts and initializes the bridge process.

If your application requires command line arguments other than those expected by *gsi_start()*, you can remove the non-standard arguments from *argv* and adjust *argc* before passing them to *gsi_start()*. See your C manual for information about how to manipulate the command line argument structures.

The main() Function in Continuous and One-Cycle Modes

G2 Gateway supports two modes of bridge operation: continuous and one-cycle. These modes affect the behavior of *gsi_run_loop()* and the flow of control in your G2 Gateway bridge:

- In continuous mode, the bridge runs entirely within the *gsi_run_loop()* event-processing loop initiated by the API function *gsi_start()*. If you intend to run your bridge only in continuous mode, a call to *gsi_start()* is the only required statement in the *main()* function.
- In one-cycle mode, *gsi_start()* exits after *gsi_run_loop()* completes one cycle, and control passes to your user code. If you intend to run your bridge in one-cycle mode, you must add code to *main()* that defines an event processing loop, which receives control of your program when *gsi_start()* finishes executing the first time.

Your user code can change modes at any time, using the following statements:

- *gsi_set_option(gsi_one_cycle)*, which sets the bridge to run in one-cycle mode.
- *gsi_reset_option(gsi_one_cycle)*, which sets the bridge to run in continuous mode.

By default, the *gsi_one_cycle* option is reset, causing your bridge to run in continuous mode.

For information about the uses of continuous and one-cycle modes, see Behavior of gsi_run_loop() in Continuous and One-Cycle Modes.

Sample main() Function

Gensym provides a source file named *gsi_main.c* as a sample *main()* function. The default contents of this file are:

```
#include "gsi_main.h"

int main(argc, argv)
      int   argc;
      char *argv[];

{
   GSI_SET_OPTIONS_FROM_COMPILE();
   gsi_start(argc,argv);
   return 0;
}
```

For continuous mode operation, you can use the *main()* function in *gsi_main.c* without modification. If you want to operate your bridge in one-cycle mode, you must add code to *main()* that establishes and controls an event processing loop.

Using gsi_start()

Within your *main()* function, the first G2 Gateway API function called must be the API function *gsi_start()*, which starts and initializes G2 Gateway.

Caution *main()* cannot call any other function before *gsi_start()*.

gsi_start() does the following:

- Receives the *argc* and *argv* arguments from *main()*.
- Initializes G2 Gateway internals.
- Executes the callback function *gsi_set_up()*.
- Establishes network listeners as specified by the command line arguments or by the callback function *gsi_get_tcp_port()*.
- Passes control to the API function *gsi_run_loop()*, which provides the main event loop of the G2 Gateway bridge. For information about this function, see Processing Events through gsi_run_loop().

Performing Once-Only Operations through gsi_set_up()

The intended use of the callback function *gsi_set_up()* is to perform operations that need to be performed only once in the lifetime of a G2 Gateway process. For this reason, *gsi_start()* calls *gsi_set_up()* only once.

You can use *gsi_set_up()* to perform operations such as:

- Selecting initial G2 Gateway runtime options. G2 Gateway's run-time options are global variables that control G2 Gateway operations and communications. You can set and reset options at any time during the execution of a bridge. For more information about G2 Gateway options. see Preprocessor Flags and Runtime Options.
- Registering customized error handlers. For information about error handlers, see Error Handling.
- Declaring local functions in the bridge that G2 can invoke as remote procedures. For information about how to write and declare these functions, see Remote Procedure Calls.
- Allocating G2 Gateway data structures that you intend to use repeatedly.

Note Calls to *gsi_set_option()*, *gsi_rpc_declare_local()*, and other functions that are best called from *gsi_set_up()* do not work if *gsi_start()* has not been called.

The following example illustrates a *gsi_set_up()* function used in a G2 Gateway application that handles alarms:

```
void gsi_set_up()
{
   gsi_attr *attrs;

/* Set this application to run in one-cycle mode. */
   gsi_set_option(GSI_ONE_CYCLE);

/*
* Install customized error handler to be called after
* G2 Gateway internal errors occur and are handled by
* the internal error handler.
*/
gsi_install_error_handler(my_custom_error_handler);

/* Declare local functions to be remotely invocable. */
gsi_rpc_declare_local(enable_alarming,
   "ENABLE-ALARMING");
gsi_rpc_declare_local(disable_alarming,
```

```
    "DISABLE-ALARMING");
}
```

This example of a *gsi_set_up()* function does the following:

- Sets the runtime option to one cycle mode.
- Installs a customized error handler. For information about how to create and install customized error handlers, see Error Handling.
- Declares user-written G2 Gateway functions as remote procedures that G2 can call. For information about how to declare G2 Gateway functions as remote procedures, see Making Remote Procedure Calls from G2 to the G2 Gateway Bridge.

For information about the functions invoked from this example, see API Functions.

Specifying a Default TCP/IP Port Number

G2 Gateway invokes the callback function *gsi_get_tcp_port()* only when the command line used to start the G2 Gateway bridge process specifies no port number, or 0 for the port number.

gsi_get_tcp_port() returns a user-specified default TCP/IP port number for the ICP socket used by G2 Gateway to connect to an external system over a TCP/IP link. If *gsi_get_tcp_port()* returns 0, G2 Gateway uses Gensym's default port number 22041. If 22041 is not available, G2 Gateway uses the first available port number within the next 99 addresses.

In the following example, *gsi_get_tcp_port()* returns 0, which causes G2 Gateway to use the default port number 22041 or the first available number within the next 99 addresses:

```
#define TCPIP_PORT_NUMBER  0

...

gsi_int gsi_get_tcp_port()
{
   return(TCPIP_PORT_NUMBER);
}
```

For information about the syntax of the command line for starting the bridge, see Building and Running a G2 Gateway Bridge.

If TCP/IP is not installed, *gsi_get_tcp_port()* has no effect.

Managing a Connection between G2 and a G2 Gateway Bridge

You can use the following G2 Gateway callback functions to control the connection between the G2 Gateway bridge process and G2:

- `gsi_initialize_context()`, called when a connection between G2 and G2 Gateway is established.
- `gsi_pause_context()`, called when G2 pauses its KB.
- `gsi_resume_context()`, called when G2 resumes its KB.
- `gsi_shutdown_context()`, called when a connection between G2 and G2 Gateway is shut down.

Initializing a Connection

Whenever a GSI interface is enabled, G2 sends an initialization request to the G2 Gateway bridge process, which then calls the callback function `gsi_initialize_context()` Each *context* corresponds to a single GSI interface in G2.

Hint Since context numbers are contiguous integers they can be used as array indices to access contexts stored in an array.

You can use `gsi_initialize_context()` to initialize a connection between a single GSI interface in G2 and GSI, and give G2 Gateway the option of rejecting the connection. `gsi_initialize_context()` can perform tasks such as:

- Validating connections from G2, as for a login procedure.
- Declaring G2 procedures as remote procedures that your G2 Gateway bridge can invoke, using the API function `gsi_rpc_declare_remote()`. These remote procedure declarations are valid only for the context through which the G2 process is connected to the G2 Gateway bridge.
- Allocating and/or initializing global tables on a per-connection basis; that is, tables that are unique to this connection.

For more information about `gsi_initialize_context()`, see Callback Functions.

Pausing a Connection

G2 Gateway calls the callback function `gsi_pause_context()` whenever G2 pauses a KB. The `gsi_pause_context()` function accepts no arguments and returns no value.

gsi_pause_context() is useful for pausing any functions in your G2 Gateway bridge that operate independently of G2. *gsi_pause_context()* can suspend these functions until G2 resumes operation. For example, you can use *gsi_pause_context()* to halt unsolicited data collection from a queue in the external system, record an event in a log file, or stop the G2 Gateway watchdog timer invoked through the API function *gsi_watchdog()*.

Calls to *gsi_g2_poll()* are stopped when the G2 process pauses its current KB.

Resuming a Connection After a Pause

G2 Gateway calls the callback function *gsi_resume_context()* whenever G2 resumes a paused KB.

You can use *gsi_resume_context()* to prepare the external system to access data, resume unsolicited data collection, record events in a log file, or inform a G2 operator that the application has resumed. *gsi_resume_context()* accepts no arguments and returns no value.

If your G2 application initiates all read and write access requests, you can leave *gsi_pause_context()* in its stub form, because all read and write requests are stopped when the G2 application is stopped. If you do not use *gsi_pause_context()*, you can also leave *gsi_resume_context()* in its stub form.

Shutting Down a Connection

G2 Gateway calls the callback function *gsi_shutdown_context()* whenever G2 is reset or the GSI interface is deactivated or deleted. *gsi_shutdown_context()* is called once for each active G2 Gateway context. It accepts no arguments and returns no value.

You can use *gsi_shutdown_context()* to perform the tasks necessary to shut down the external system and clean up the G2 Gateway bridge process. Possible uses for *gsi_shutdown_context()* include:

- Shutting down your external system.
- Disabling all data collection.
- Resetting your data collection functions.
- Closing files.
- Recording events in a log file.
- Freeing any memory allocated by the bridge.

Before you call *gsi_shutdown_context()*, make certain that you free any dynamically allocated memory not freed in the body of the *gsi_receive_deregistrations()* callback. For information about how to do this, see Using gsi_receive_deregistrations().

Processing Events through gsi_run_loop()

The API function *gsi_run_loop()* initiates the main event loop of the G2 Gateway bridge. This event loop handles network activity on connections to G2 processes and to external systems.

Each time *gsi_run_loop()* is executed, it does the following:

1. Makes any new connections requested by G2.
2. In each currently active context, responds to all outstanding messages received from G2.

   gsi_run_loop() processes all messages that are outstanding at the time when it is called. Because G2 Gateway is single-threaded, *gsi_run_loop()* processes only those messages that are already completed at the time when *gsi_run_loop()* is called. Any messages that arrive or are completed during the current call to *gsi_run_loop()* are processed by the next call to *gsi_run_loop()*.

gsi_run_loop() executes until it has nothing to do, or until the run loop timeout period is reached, whichever happens first. By default, the run loop timeout period is 200 ms. You can call *gsi_set_run_loop_timeout()* to specify a different timeout period for *gsi_run_loop()*. For information about this function, see *gsi_set_run_loop_timeout*.

gsi_run_loop() is executed automatically by the API function *gsi_start()*. In one-cycle mode, you can execute *gsi_run_loop()* explicitly, outside of *gsi_start()*. The following sections describe the behaviors and uses of *gsi_run_loop()* in continuous and one-cycle modes.

Behavior of gsi_run_loop() in Continuous and One-Cycle Modes

In continuous mode, the bridge executes entirely inside the call to *gsi_start()*, which calls *gsi_run_loop()* repeatedly as long as no fatal error occurs. Continuous mode is the default mode of bridge operation.

In one-cycle mode, *gsi_start()* exits after the first call to *gsi_run_loop()* is completed. Your G2 Gateway bridge then executes under control of user code that you provide.

gsi_run_loop() in Continuous Mode

When the G2 Gateway bridge is running in continuous mode, it executes entirely within *gsi_start()*, which invokes *gsi_run_loop()* repeatedly as long as no fatal error occurs.

Control passes from *gsi_run_loop()* to your user code only when GSI, in response to network activity, invokes callback functions or functions declared as

remote procedures. Control returns to *gsi_run_loop()* when the callback or remote procedure finishes executing.

When there is no network activity to which the bridge must respond, *gsi_run_loop()* calls the API function *gsi_pause()* internally, which causes the G2 Gateway bridge to enter an interruptible sleep. The bridge awakens when it detects network activity to which it must respond. The bridge can sleep for no longer than 1 second, after which time it awakens automatically; if there is no network activity to which it must respond, it reenters the interruptible sleep.

Continuous mode is the better mode for polling an external system for data. The bridge can poll the external system using the callback function *gsi_g2_poll()*, which is invoked by G2 Gateway approximately once per schedule cycle (one second by default).

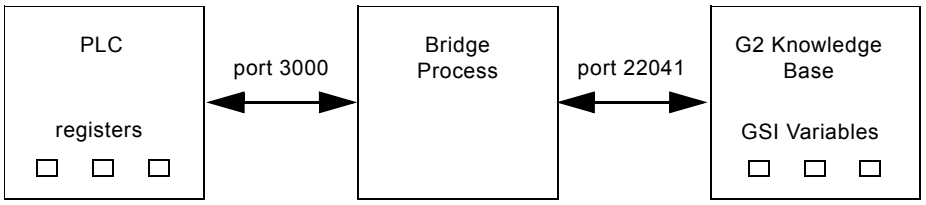
gsi_run_loop() in One-Cycle Mode

When the G2 Gateway bridge is running in one-cycle mode, control passes from *gsi_start()* when *gsi_run_loop()* completes execution for the first time. Control then passes to your user code, which can subsequently execute *gsi_run_loop()* as needed.

In one-cycle mode, your user code must provide the main event loop of the bridge. This loop must make periodic calls to *gsi_run_loop()* to respond to the network events that the G2 Gateway bridge detects on active connections to G2 applications. When *gsi_run_loop()* has responded to all these events, it returns control to the main event loop provided by your user code, which can process other events before calling *gsi_run_loop()* again.

One-cycle mode is the better mode for bridges designed to respond to network activity on connections to external systems, rather than to poll the external systems actively.

The following figure illustrates a G2 Gateway bridge process that enables a G2 knowledge base to communicate with a PLC:



The following code illustrates how user code running in one-cycle mode can respond to messages both from the PLC and from the G2 knowledge base:

```
main(argc, argv)
   int argc;
   char *argv[];
{

/* One-cycle mode is selected by the gsi_set_up()
   callback invoked through gsi_start(). */

   gsi_start();

/* Code to connect to PLC */
   .
   .
   .
/* A file descriptor is specified by user-defined
   global variable PLC_FD, corresponding to port
   30000. */

   gsi_watch_fd(PLC_FD);

/* User-defined event loop. */

   for (;;) {
      gsi_pause();

   /* Check for data or messages from the PLC.*/
      my_get_plc_data();

   /* Check for data or messages from G2.*/
      gsi_run_loop(); }
}
```

The example above assumes that the callback *gsi_set_up()*, invoked by *gsi_start()*, calls *gsi_set_option(GSI_ONE_CYCLE)* to set the bridge to run in one-cycle mode. (Continuous mode is the default.)

The *gsi_watch_fd(PLC_FD)* statement then sets the bridge to watch the file descriptor specified by a global variable, *PLC_FD*. This global variable corresponds to port 30000.

Control then passes to the *for()* loop, which loops indefinitely as long as no fatal error occurs. The first function called in the *for()* loop is *gsi_pause()*, which causes the G2 Gateway bridge to enter an interruptible sleep. The G2 Gateway bridge wakes up when:

- The PLC sends data to the bridge on port 30000. The bridge then calls the user-defined procedure *my_get_plc_data()*, which receives the data from the PLC and sends it to G2. To send the data to G2, *my_get_plc_data()* can call the API functions *gsi_return_values()*, *gsi_return_attrs()*, *gsi_return_timed_values()*, or *gsi_return_timed_attrs()*. When *my_get_plc_data()* completes, control passes to *gsi_run_loop()*. If *gsi_run_loop()* finds no messages from G2, it exits and control passes to the beginning of the *for()* loop.
- G2 sends a message to the bridge on any active connection. The bridge then calls *my_get_plc_data()*, which finds that there is no message from the PLC. Control then passes to *gsi_run_loop()*, which responds to the message from G2. When *gsi_run_loop()* completes one loop, it exits and control passes to the beginning of the *for()* loop.

Communicating with G2 After Running Outside of gsi_run_loop()

If your G2 Gateway bridge has been running outside of *gsi_run_loop()* for any length of time, your user code should call *gsi_run_loop()* at least once before it attempts to send values to G2 or attempts to communicate with G2 in any other way. Running *gsi_run_loop()* gives your user code a chance to respond to any outstanding messages from G2 that it may have received while running outside of *gsi_run_loop()*. Thus, your user code can become informed about changes that have taken place within the G2 KB before it attempts to communicate with the KB.

For example, while the bridge was running outside of *gsi_run_loop()*, GSI variables may have been deregistered by an event in G2 such as the deactivation of the workspace on which the variables reside. The G2 Gateway bridge user code remains unaware of the deregistrations because *gsi_receive_deregistrations()* is never invoked while the bridge is running outside of *gsi_run_loop()*. Thus, your user code might attempt to communicate with GSI variables that have been deregistered. Before your user code attempts to send values to the GSI variables, it should call *gsi_run_loop()* so that *gsi_receive_deregistrations()* can be invoked, if necessary, to deregister any GSI variables that were deregistered by G2 while the bridge was running outside of *gsi_run_loop()*.

Interruptible Sleep

To save system resources, a G2 Gateway bridge can enter an **interruptible sleep** when there is no network activity to which it must respond. What causes a bridge

to enter an interruptible sleep depends on the mode in which the bridge is running.

In continuous mode, G2 Gateway causes a bridge to enter an interruptible sleep *automatically* when there is no network activity to which the bridge must respond. If the bridge detects such network activity, it awakens. The bridge can sleep for no longer than 1 second, after which it awakens automatically; if there is no network activity, the bridge goes back to sleep.

In one-cycle mode, the bridge enters an interruptible sleep only when your user code calls the `gsi_pause()` function. The bridge does not enter an interruptible sleep automatically.

In either mode, the bridge is awakened when network activity is detected on:

- Any active connections to G2 processes. G2 Gateway automatically watches these connections. You do not have to instruct the bridge process to watch connections to G2.
- Particular connections to external systems that you instruct the bridge to watch for network activity.

  You use the `gsi_watch_fd()` API function to instruct the bridge to watch the file descriptor for a particular connection. You can use the `gsi_unwatch_fd()` function to cause the bridge not to watch a particular file descriptor.

  The `gsi_watch_fd()` API function is useful mainly in one-cycle mode, which is the better mode for applications designed to respond to network activity on connections to external systems, rather than to poll the external systems actively.

  However, you can use `gsi_watch_fd()` in continuous mode if you want your bridge process to awaken as soon as there is activity on a connection to an external system, rather than waiting for activity on a connection to a G2 process. In this case, you can use the callback function `gsi_g2_poll()` to respond to messages on the connection specified in the `gsi_watch_fd()` function call.

  For information about `gsi_watch_fd()` and `gsi_unwatch_fd()`, see API Functions.

Implementing a Customized Sleep Facility in One-Cycle Mode

For some purposes, you may want to implement a customized sleep facility for your G2 Gateway application, or integrate an existing sleep facility with it. To use a customized sleep facility, you must run the bridge process in one-cycle mode, which gives control over sleep to your user code.

You implement a customized sleep facility using the select statement or other operating system facility.

G2 Gateway provides two API functions, *gsi_listener_socket()* and *gsi_context_socket()* for implementing a customized sleep facility. These functions enable you to obtain a complete list of the file descriptors on which input and output can occur:

- *gsi_listener_socket()* returns the UNIX file descriptor associated with the bridge's TCP listener.
- *gsi_context_socket()*, which returns the file descriptor associated with a particular context.

For more information about *gsi_listener_socket()* and *gsi_context_socket()*, see API Functions.

Handling Interrupts

Interrupts can occur during execution of a G2 Gateway bridge process. Your G2 Gateway bridge can invoke interrupt handling routines that you write to manage these interrupts.

Caution Your interrupt handling routines must not invoke G2 Gateway API functions, because doing this can interfere with the work that was interrupted.

Implementing Data Service in G2 Gateway

G2 Gateway supports the G2 data service feature, which provides an interface for reading from and writing to data points in an external system controlled by GSI variables in a G2 KB.

G2 uses GSI variables to map data in your KB to data points in external systems. A GSI variable is a G2 variable that inherits from the class gsi-data-service.

Using the data service feature, G2 can write the values of GSI variables to external data points or read values from external data points to update the values of GSI variables. G2 can provide data service only for GSI variables, and not for G2 arrays, lists, or other items.

Note History data can be passed between a G2 Gateway bridge and a G2 only by means of remote procedure calls, and not as a side-effect of data service on GSI variables. For information about remote procedure calls, see Remote Procedure Calls.

Solicited and Unsolicited Data Transfers

G2 Gateway supports both solicited and unsolicited data transfers:

- In solicited data transfers, G2 schedules regular requests for values from the external system through G2 Gateway.
- In unsolicited data transfers, G2 Gateway is responsible for updating the values of GSI variables at regular intervals, or whenever their values change in the external system.

You can use the following G2 Gateway callback functions in your user code to support data service for GSI variables:

- *gsi_get_data()*, which G2 Gateway calls whenever G2 requests a value for a GSI variable. You can implement this callback to retrieve a value from an external system and return it to G2.
- *gsi_g2_poll()*, which G2 Gateway calls at regular intervals. You can implement this callback to seek data from an external system and return it to G2, or to perform other user-specified actions.
- *gsi_set_data()*, which G2 Gateway calls whenever G2 requests the bridge to set the value of an external data point. G2 makes this request when a set action is run on a GSI variable.

The *gsi_get_data()* and *gsi_g2_poll()* callbacks can use the following API functions to return data to GSI variables in G2:

- *gsi_return_values()*, which returns one or more values to the last-recorded-value attributes of one or more GSI variables.
- *gsi_return_timed_values()*, which returns one or more timestamped values to the last-recorded-value attributes of one or more GSI variables. If the GSI variables have history-keeping specifications, the values are stored in G2 history.
- *gsi_return_attrs()*, which returns a value to the last-recorded-value attribute of a GSI variable, and returns one or more values (an array) to attributes of the same GSI variable.
- *gsi_return_timed_attrs()*, which returns a timestamped value to the last-recorded-value attribute of a GSI variable, and returns one or more values (some, all, or none of which may be timestamped) to attributes of the same GSI variable.

For more information about these functions, see Part II, Reference, of this manual.

Returning Solicited Data to G2

When G2 asks G2 Gateway for a value for a GSI variable, G2 Gateway calls the *gsi_get_data()* callback to get the value from the external system and return it to G2. Data requested by G2 under these conditions is known as **solicited data**.

G2 asks G2 Gateway for a value for a GSI variable only if the validity-interval of the variable has expired, indicating that the current value of the variable can no longer be considered valid. The conditions under which G2 then requests G2 Gateway for a value depend on the setting of the external-system-has-a-scheduler attribute of the GSI interface that G2 is using to communicate with G2 Gateway.

If the external-system-has-a-scheduler attribute is set to yes, G2 Gateway calls *gsi_get_data()* when:

- A GSI variable is created, activated, or enabled, and the default-update-interval attribute of the variable is set to a value other than none.
- A display item such as a readout table, trend chart, dial, or meter requests a value for the GSI variable.
- A local name declaration or a collect data or wait until statement within a G2 procedure requests a value for the GSI variable.
- A rule that refers to the GSI variable is invoked.
- An update request is made through the G2 update action within a G2 rule or procedure.

If the external-system-has-a-scheduler attribute of the GSI interface is set to no, G2 Gateway calls *gsi_get_data()* in the same circumstances as when this attribute is set to yes. But in addition, G2 Gateway calls *gsi_get_data()* at each expiration of a GSI variable's default-update-interval, if this interval is other than none.

At every G2 scheduler cycle:

- The handles of any GSI variables that require updates are sent to G2 Gateway and packaged into *gsi_registered_item* structures, and are then passed to *gsi_get_data()*. The handle is an integer that uniquely identifies the item. Procedures in a G2 Gateway bridge can use the handles to refer to particular objects in G2.
- For all requests for set operations, either through a rule or a procedure, that have occurred since the last clock tick, handles are sent to G2 Gateway and packaged into *gsi_registered_item* structures, and are then passed to *gsi_set_data()*.

If you have specified a value other than none for the grouping-specification attribute of the GSI interface, G2 groups the requests according to the identifying attributes you specified for the attribute. You can group requests using more than

one identifying attribute, but you can only group requests with identifying attributes. For best performance, do not use grouping unless it is needed for other reasons.

Using gsi_get_data()

G2 Gateway calls the callback function *gsi_get_data()* when G2 requests a value for one or more GSI variables. Within *gsi_get_data()*, you can include:

- Code that gets the value of a data point in an external system.
- The API functions *gsi_return_values()*, *gsi_return_timed_values()*, *gsi_return_attrs()*, or *gsi_return_timed_attrs()*, which return values to the GSI variable in the G2 knowledge base.

G2 Gateway builds an array of *gsi_item* structures and assigns to each structure the handle of a GSI variable requesting data. When G2 Gateway receives a data request from G2 for one or more GSI variables, it calls *gsi_get_data()* and passes to it the array of *gsi_registered_item* structures.

For information about the *gsi_registered_item* structure, see G2 Gateway Data Structures.

To return the data to G2, place a call inside *gsi_get_data()* to one of the G2 Gateway API return functions. In the typical case where *gsi_return_values()* is used to return data to G2, you would reuse the array of *gsi_registered_item* structures passed to *gsi_get_data()*, looping through them and filling in the *value* and *status* fields of each element, and passing the same array of structures as an argument to *gsi_return_values()*.

Using gsi_set_data()

G2 Gateway calls the callback function *gsi_set_data()* when G2 requests the bridge to set the value of a data point in an external system. Your G2 KB can set the value of an external data point using the set action on a GSI variable that is mapped to the data point.

If the GSI variable is not already registered with the bridge when the set action is run on it, the variable is registered with the bridge automatically, causing G2 Gateway to call the callback function *gsi_receive_registration()*.

After the values are successfully set in the external system, you can echo the values back to the GSI variables in G2. Echoing values back to G2 ensures that values just set in the external system are consistent with the last recorded values of the variables in the G2 knowledge base.

The G2 set action does *not* change the last-recorded-value attribute of a GSI variable. However, you can set last-recorded-value to the value that you specify in the set action by causing G2 Gateway to echo the value back to the GSI variable.

To echo back the value, call one of the G2 Gateway data return functions *gsi_return_values()*, *gsi_return_attrs()*, *gsi_return_timed_values()*, or *gsi_return_timed_attrs()* and use the same *gsi_item* structure you passed to *gsi_set_data()*. A value set with *gsi_set_data()* will only be echoed back to the GSI variable in G2 if *gsi_set_data()* returns the value through a data return function. Using the set action with a GSI variable does not automatically cause that value to be returned to G2 as the last recorded value of the variable.

Using the Gsi-Variable-Status Attribute

The gsi-variable-status attribute of a GSI variable indicates the status code value of the external data point or variable to which a GSI variable in G2 is mapped.

The possible values for the gsi-variable-status attribute are:

- 0: OK.
- 1 - 5: Reserved by Gensym for future releases.
- 6 and up: These status codes are application-specific. You can establish any set of status codes needed for your application. See Identifying the Status of the GSI Variable for information about how the G2 knowledge base can use these status codes.

You set the gsi-variable-status in your G2 Gateway code with the *gsi_set_status* API function. For information on this function see *gsi_set_status*.

When you set the status of your *gsi_registered_item* data structure to 0:

- Your variable will receive its value if the value returned is a non-null data type.
- You will receive a G2 Gateway programming error if you return a null data type. Never give a null data type a *status* code of 0.

When you set the status of your *gsi_registered_item* data structure to a value other than 0:

- Your variable will receive its value if the value returned is a non-null data type.
- Your variable will not be updated if your value has a null data type.

Sending Unsolicited Data to G2

Unsolicited data is data that a bridge sends to G2 without having received a request from G2 for the data. Common examples of unsolicited data are:

- Alarm conditions raised in an external system.
- Reports of values in an external system that have gone out of range.
- Updates sent to G2 automatically to relieve G2 of the processing load of scheduling and sending update requests to G2 Gateway.

Unsolicited data can be sent to G2 in two ways:

- When an external system sends the unsolicited data to the bridge without having received a request for the data from the bridge. Getting unsolicited data this way is better done in one-cycle mode.
- When G2 Gateway retrieves data from the external system by calling the callback function *gsi_g2_poll()*. Getting unsolicited data this way is better done in continuous mode.

Note G2 causing *gsi_g2_poll* to be called would seem like solicitation. It is not, however, because G2 is not soliciting values for any specific variable.

Reporting by Exception

If your external system supports reporting by exception, you can achieve best performance by running your G2 Gateway application in one-cycle mode and placing the user code that responds to exceptions outside the *gsi_run_loop()* call tree. Your user code can return the exceptions to G2 as they occur, using the G2 Gateway data return functions.

If you return values reported as exceptions to G2 in this way, it is important to identify the numbers of the contexts in which the variables are defined. Outside of the *gsi_run_loop()* call tree, the current context number is undefined, and you must specify the context number in order to refer to the intended variable.

To get data to G2 from an external system that supports reporting by exception:

1. Verify that your external system supports reporting by exception.
2. Find out which file descriptor represents the socket to which the external system is connected.
3. In your *gsi_set_up()* callback function, include the following call to change the mode to one-cycle:

   ```
   gsi_set_option(GSI_ONE_CYCLE);
   ```

4. In your *main()* function, call *gsi_watch_fd()* to specify the socket to watch for messages from the external system.

5 Write a for *(;;)* loop that repeatedly calls *gsi_pause()*, *gsi_run_loop()*, and the code that you write to handle the data from the external system. *gsi_pause()* causes G2 Gateway to enter an interruptible sleep from which it awakens when there is network activity on connections to G2 or on connections to external systems that the bridge is watching.

gsi_run_loop() handles messages sent to the bridge from G2, and your user code handles data and messages from the external system in whatever way is appropriate for your application.

The following code illustrates one way to get unsolicited data from an external system. Note that this example is similar to the one-cycle PLC code in the section called gsi_run_loop() in One-Cycle Mode.

```
gsi_int EXT_SYS_FD = 30000; /* Global variable
                               representing socket to
                               external system */
main(argc, argv)
   int argc;
   char *argv[];
{
   /* gsi_start() must be called in main() before any
      Gateway calls */

   gsi_start(); /* Invokes gsi_set_up() to change mode
                   to one-cycle. */

   /* User code to connect to external system */

   gsi_watch_fd(EXT_SYS_FD); /* Specify file
                                descriptor to watch. */
   for (;;)
   {
      gsi_pause();     /* Enter interruptible sleep. */
      gsi_run_loop(); /* Check for messages from G2. */
      my_code();       /* Check for messages or data
                          from external system. */
   }
}
```

Polling an External System for Data

A G2 Gateway bridge can also receive unsolicited data by polling the external system for data, without being requested by G2 to get new data values. You can choose to send values to G2 only when the data values that the bridge obtains differ from previously stored values.

When the poll-external-system-for-data attribute for the GSI interface is set to yes, G2 Gateway calls *gsi_g2_poll()* to enable the external system to return unsolicited values to G2.

When the external-system-has-a-scheduler attribute for the GSI interface is set to yes, the default update interval is sent to G2 Gateway in calls to *gsi_receive_registration()*, *gsi_set_data()*, and *gsi_get_data()* which are then called only in cases of explicit updates (G2 update action).

You can use *gsi_initialize_context()* to initialize any structures used for providing unsolicited data. *gsi_g2_poll()* can access the handle for a variable and any information for filtering from *gsi_receive_registration()*.

When your user code retrieves data from the external system, it can allocate arrays in which to return the data to G2:

- For single data values, it can allocate arrays of *gsi_item* or *gsi_registered_item* structures.
- For arrays of attribute values, it can allocate arrays of *gsi_attr* structures.

To send the data values to G2, your user code writes the data values into these arrays and passes the arrays to the API functions *gsi_return_values()*, *gsi_return_attrs()*, *gsi_return_timed_values()*, or *gsi_return_timed_attrs()*. Each of these API functions requires handles to identify the GSI variables to which it is returning values.

Setting Values in the External Application

You can use the G2 set action to assign values to external data points that are mapped to the GSI variables in your knowledge base. Running a set action on a GSI variable causes the variable to be registered automatically, if it is not currently registered.

You can invoke the set action in a G2 rule or procedure. The following are examples of the set action in a rule:

```
unconditionally set spindle-motor-on-off to 0

initially set the set-point of device-controller-1 to the symbol on
```

If the GSI variable is not registered with the bridge when you run the set action on it, it is registered automatically; this causes G2 Gateway to invoke the callback function *gsi_receive_registration()*.

Note The set action does not conclude a value back to the last-recorded-value of the GSI variable. To conclude the value back to the GSI variable in the G2 knowledge base, your G2 Gateway user code must call the API function `gsi_return_values()`. You can place a call to `gsi_return_values()` in the `gsi_set_data()` callback function that G2 Gateway invokes to process the set action. See Using gsi_set_data() for more information.

If you want to be able to set the value of an attribute of a GSI variable, you must define that attribute to be a GSI variable itself. For more information on the G2 set action, refer to the *G2 Reference Manual*.

Caution Because a GSI variable is reregistered whenever the values of any of the identifying attributes are changed, do not return values, directly or indirectly, to identifying attributes from the G2 Gateway bridge user code. This would result in unnecessary exchanges of data between the G2 and the G2 Gateway bridges.

When G2 makes a set request to the bridge, G2 Gateway builds an array of `gsi_registered_item` structures to hold the information for each GSI variable whose external data point is affected by the request. G2 Gateway then calls the `gsi_set_data()` callback function and passes to it the array of `gsi_registered_item` structures.

The syntax for `gsi_set_data()` is:

```
void gsi_int gsi_set_data(registered_items, count)
   gsi_registered_item *registered_items;
   gsi_int  count;
```

where:

registered_items is an array of `gsi_registered_item` structures. It is defined by G2 Gateway and initialized to point to the first element of the `objects` array built by G2 Gateway. Each element of the array contains the handle and value of the corresponding data point to be *set*.

count is a `gsi_int` value specifying the number of `gsi_registered_item` structures in the array. Use `count` to loop through and process each `gsi_registered_item` in the array.

Requests to set GSI variables can be grouped by G2 according to the grouping-specification attribute of the GSI interface. For information about this attribute, see Setting Attributes of a GSI Interface.

Message Passing

A G2 KB can send text messages to a G2 Gateway bridge, using GSI Message Servers and the *gsi_receive_message()* callback function. A G2 Gateway bridge can send text messages to the Message Board in G2 from the G2 Gateway bridge, using the *gsi_return_message()* API function.

Sending Messages from G2 to the External System

A G2 KB can send text messages to a G2 Gateway bridge using a **GSI Message Server**. A GSI Message Server is a G2 object that inherits from the G2 mixin class gsi-message-service and from at least one other G2 class.

To send the text message to the external system, G2 runs an inform action on the GSI Message Server. The inform action specifies a text message that is sent to the G2 Gateway bridge. For information about how to create and configure a GSI Message Server, see Creating and Configuring GSI Message Servers.

When the G2 Gateway bridge receives the text message, it invokes the *gsi_receive_message()* callback function, and passes the message to this function. You complete the code of the *gsi_receive_message()* callback to specify how the callback sends the message to the external system.

gsi_receive_message() can also return data to G2, by calling one of the data return API functions *gsi_return_values()*, *gsi_return_timed_values()*, *gsi_return_attrs()*, or *gsi_return_timed_attrs()*. For more information about *gsi_receive_message()*, see Callback Functions.

Returning Text Messages to G2

GSI Message Servers *cannot* be used to send a text value from the G2 Gateway bridge to G2.

A G2 Gateway bridge can send text messages to the Message Board in G2 by calling the API function *gsi_return_message()*. For information about *gsi_return_message()*, see API Functions.

Item Passing

Through remote procedure calls, a G2 application can send G2 items with their attributes to the G2 Gateway bridge, and the bridge can send structures that correspond to the objects and their attributes to G2. This exchange of data is called **item passing**.

For information about how to do this, see Remote Procedure Calls.

Registering and Deregistering Items

G2 **registers** certain items when it passes them to a G2 Gateway bridge over a G2-to-G2 Gateway network connection. G2 registers items to provide a G2 Gateway bridge with the information that it needs to refer to and access the items in G2.

An item's registration is valid for one specific network connection between G2 and G2 Gateway, as configured by a GSI interface.

Kinds of Items Registered by G2

G2 registers two kinds of items:

- Data-served GSI variables.

  When data service is requested for a GSI variable in a G2 KB, G2 creates a copy of the variable, registers the copy, and sends the copy to G2 Gateway. A G2 Gateway bridge uses information in the registered copy to service the request for an updated value. G2 Gateway assigns each registered variable a handle, which is an integer that uniquely identifies the item, beginning with the number 1.

- Items passed to G2 Gateway as handles through remote procedure calls made by G2.

  The handle is an integer that uniquely identifies the item, beginning with the number 1. Procedures in a G2 Gateway bridge can use the handles to refer to particular objects in G2. In G2, the procedure that passes the item as a handle must be declared as a remote procedure with the as handle grammar.

G2 does *not* register the following items, which can be exchanged between G2 and G2 Gateway without being registered:

- Values of variables, values of parameters, or object copies (as opposed to handles) that G2 passes to G2 Gateway as arguments to remote procedure calls.
- Text sent to G2 Gateway through a gsi-message-server.

Registering Items Automatically

G2 registers items *automatically* when the following events occur:

- A GSI variable is registered automatically if it cites a connected GSI interface, is active and has a default-update-interval other than none or, if its validity-interval is indefinite, the first time a read or write operation from or to a G2 Gateway bridge is performed on a data-served GSI variable for the first time after the GSI variable is created or activated.

- One of the identifying attributes of a GSI variable is modified. This causes the GSI variable to be deregistered and then reregistered automatically.
- G2 passes an object as a handle, through a remote procedure call that is declared in G2 with the as handle grammar.

A G2 item is registered when it passes to a G2 Gateway bridge process through a particular context for the first time. Thus, a single G2 item can have more than registration and more than one item handle. For example, if data service is performed on a GSI variable through two separate contexts, the GSI variable is registered twice. The variable now has two separate registrations and two separate handles. Handles are unique only within a context. If a G2 item has handles in several different contexts, there is no guarantee that they will be different from each other.

Registering Items Explicitly

You can register any item explicitly by calling the G2 system procedure g2-register-on-network(). This procedure enables you to register a G2 item that you can later pass to G2 Gateway as an item handle, by calling a remote procedure declared with the as handle grammar.

For information about g2-register-on-network(), see the *G2 System Procedures Reference Manual*.

What G2 Gateway Does When G2 Registers an Item

When G2 registers a GSI variable or a handle, G2 Gateway does the following:

1. Creates a `gsi_registration` structure. This structure contains information, provided by G2, that the bridge can use to complete any read or write operations on the registered item.

   Because the bridge can use the information in the `gsi_registration` structure to service any number of read or write requests on the same item, the `gsi_registration` structure remains in existence until the item is deregistered.

2. Calls the callback function `gsi_receive_registration()`, and passes the `gsi_registration` structure to the function call.

   You can use `gsi_receive_registration()` to perform tasks such as initializing the external data point to which you are mapping the registered item, allocating memory, or returning the handle to an attribute of the variable for some future use.

   For more information about the `gsi_registration` and `gsi_registered_item` data structures, see G2 Gateway Data Structures.

3 For each request to read from or write to a data-served GSI variable, G2 Gateway creates a *gsi_registered_item* structure, and passes this structure to the *gsi_set_data()* or *gsi_get_data()* that it calls to service this request.

   The *gsi_registered_item* structure contains information needed to complete the current read or write operation on the registered GSI variable. For this reason, the *gsi_registered_item* structure remains in existence only during execution of the current *gsi_get_data()* or *gsi_set_data()* procedure.

How G2 Gateway Stores Information Associated with Registered Items

When G2 Gateway receives registered items, it creates internal data structures that store information associated with the registered items. These internal structures are referenced by the following *void ** pointers provided with G2 Gateway:

- *gsi_registration*, which is created by G2 Gateway when G2 first registers a data-served GSI variable or a handle that G2 passes through a remote procedure call. The *gsi_registration* structure stores the item handle, name, data type, six identifying attributes, attribute count, and default update interval of the variable being registered. It can also store user data that your G2 Gateway user code chooses to associate with this registered item.
- *gsi_registered_item*, which is created by G2 Gateway each time G2 requests G2 Gateway to read from or write to a data-served GSI variable. The *gsi_registered_item* structure stores the item handle, status, default update interval, and a pointer to a *gsi_item* structure, which contains additional information associated with a registered item, such as its value.

  At every G2 scheduler cycle:

  – For any GSI variables that require updates, G2 sends handles to G2 Gateway, which packages them into *gsi_registered_item* structures and passes them to the callback function *gsi_get_data()*.

  – For any requests for set actions on GSI variables, either through a rule or a procedure, that have occurred since the last clock tick, G2 sends handles of the GSI variable to G2 Gateway, which packages them into *gsi_registered_item* structures and passes them to the callback function *gsi_set_data()*.

Your user code can access the information stored in these structures by calling API functions provided with G2 Gateway. For information about these API functions, see API Functions.

For a description of the information in G2 Gateway data structures that you can access through API functions, see G2 Gateway Data Structures.

Associating User Data with a Registered Item

For some purposes, your application may need to store application-specific information on the objects that it registers with G2 Gateway. To do this, your G2 Gateway user code can associate data with each registered object through the *user data* component of the object's *gsi_registration* structure.

The *user data* component points to a location that your G2 Gateway user code can both write to and read from. G2 Gateway itself neither reads from nor writes to the structure pointed to by the *user data* component.

You use the API function *gsi_set_user_data()* to set the *user data* component of a *gsi_registration* structure, as illustrated in the following example:

```
void gsi_receive_registration(item_reg)
gsi_registration item_reg;
{
   struct tag_type *tag_ptr;
/*
   Create a new tag structure, attach it to the main
   object array and call a routine to translate
   the identifying attributes stored in the
   gsi_registration into tag data for quick access
   later.
*/
   tag_ptr = (tag_type *) malloc(sizeof(tag_type));
   gsi_set_user_data(item_reg,(void*)tag_ptr);
   fill_tag_struct_using_item_reg
      (tag_ptr,gsi_handle_of(item_reg));
}
```

You use the API function *gsi_user_data_of()* to obtain the contents of the *user data* component of a *gsi_registration* structure. For information about *gsi_set_user_data()* and *gsi_user_data_of()*, see API Functions.

Note The enclosing *gsi_registration* structure is reclaimed when the item is deregistered, but the unallocated memory is the responsibility of the user. Typically this happens in the callback function *gsi_receive_deregistrations()*.

Deregistering Items Automatically

Items are deregistered automatically when:

- A GSI interface is disabled, deactivated, or deleted. All registered items associated with that GSI interface are immediately deregistered before the connection to G2 Gateway is shut down.
- An individual registered item is disabled, deactivated or deleted.
- Any identifying attribute of a GSI variable is changed. In this case, the GSI variable is deregistered and reregistered automatically, with the same handle.
- When the G2 knowledge base is reset, at which time any active items must stop receiving data.
- The workspace where the item is located is deactivated.

When registered items are deregistered, their handle numbers are sent to G2 Gateway where they are packaged into `gsi_registration` structures, and passed to the user in one or more calls to the callback function `gsi_receive_deregistrations()`. This callback deregisters more than one item in a single call. In contrast, the callback function `gsi_receive_registration()` registers only one item in a single call.

Using gsi_receive_deregistrations()

G2 Gateway calls the callback function `gsi_receive_deregistrations()` when an item is deregistered. Your bridge must delete any internal mapping of the item to prevent future confusion, because its handle may be recycled by G2.

The callback function `gsi_receive_deregistrations()` has the following syntax:

```
void gsi_receive_deregistrations(registered_items,count)
   gsi_registered_item *registered_items;
   gsi_int  count;
```

where:

registered_items is an array of `gsi_registered_item` structures.

count is a `gsi_int` value that gives the number of `gsi_registered_item` elements in the array.

`gsi_receive_deregistrations()` should unpack the array pointed to by *registered_items* and identify the items it contains. For each item in the array that is a GSI variable, it must remove the item from the data acquisition queue of the external system. If the external system has a scheduler, the external system must be notified that the items are no longer receiving data.

Deregistering Items Explicitly

You can explicitly deregister an item using the G2 system procedure g2-deregister-on-network(). For information about this procedure, see *G2 System Procedures Reference Manual*.

Context Control

Each instance of a GSI interface connecting to a G2 Gateway bridge process is referred to as a **context**. All GSI interfaces, whether of the same or of different G2 processes, can communicate with the same G2 Gateway bridge process, but use separate contexts. G2 Gateway manages the separate contexts (maximum 50) and tracks which GSI interface is served in each context.

Contexts are identified by a number. At any time, your program can call the G2 Gateway API function `gsi_current_context()` to get the current context number.

Note Outside the `gsi_run_loop()` call tree, the current context is undefined. If the current context is undefined, `gsi_current_context()` returns -1.

If you plan to support more than one context and you have global variables or structures whose scope is limited to a single context, you must create an array that provides for a duplicate set of these objects for each context you plan to support.

G2 Gateway API functions that access G2 applications — for example, to send data to the G2 application, or to make remote procedure calls to it — include a *context* argument that enables you to specify the context through which the G2 application is accessed. The specified context can be the current context or any other active context.

Remote Procedure Calls within a Context

Over all active contexts, as many as 4096 remote procedure calls can be outstanding at one time.

Within the same context, any number of calls to different remote procedures can be outstanding at the same time. In addition, any number of calls to the same remote procedure from different contexts can be outstanding at the same time, subject to the overall 4096 limit.

User Watchdog Functions

The *gsi_watchdog()* API function calls a function that you specify when the *gsi_watchdog()* function's time-out interval expires. When *gsi_watchdog()* is called, its timer begins counting down to zero from the specified time. If *gsi_watchdog()* is called again before the time-out period is reached, the timer is reset to the latest specified time. The watchdog timer can be reset or disabled at any time, whenever *gsi_watchdog()* is called.

Note The *gsi_watchdog()* function is not available on Windows platforms.

---

gsi_watchdog() returns no value, and accepts the following arguments:

user_watchdog_function, a pointer to a user-written function that G2 Gateway executes after the *timeout_interval* expires.

timeout_interval, an integer greater than or equal to zero, that specifies the time-out interval in seconds. If *timeout_interval* is set to zero (0), the G2 Gateway watchdog timer is disabled.

Suppose a call to *gsi_watchdog()* specifies a time period of 90 seconds. Another call to *gsi_watchdog()*, made 80 seconds later, also specifies a time period of 90 seconds. The second call causes the timer for the watchdog process to be reset to 90; the G2 Gateway bridge can then go for another 90 seconds before it times out, rather than for only 10 seconds.

To use the watchdog function, include calls to *gsi_watchdog()* in any of the functions in the user code, with appropriate periods for each. You may need to experiment to find a good expiration period for your application.

gsi_watchdog() is useful when you need to execute a function at a specified period of time, or when the G2 Gateway bridge stops communicating with G2 and you either want to re-establish communication or run a clean-up function and exit the G2 Gateway bridge process. If there is no communication between the G2 Gateway bridge and the G2 process within a period of time that you specify (perhaps due to the G2 Gateway bridge being hung), the watchdog process causes the G2 Gateway bridge to execute a specified function or to exit.

Thus, if communication between the processes breaks down and G2 loses control of the G2 Gateway bridge, the bridge process can exit and automatically close.

The following code illustrates the use of *gsi_watchdog():*

```
gsi_int gsi_initialize_context(str, str_len)
gsi_char *str;
gsi_int str_len;
{
   /*Set the timer initially.*/
   gsi_watchdog(my_exit_func,100);
}


void gsi_g2_poll()  /* Called every second */
{
   /* Resets timer to 100 seconds.*/
   gsi_watchdog(my_exit_func,100);
}


void gsi_pause_context()
{
   /* Disable timer. */
   gsi_watchdog(my_exit_func,0);
}


void my_exit_func()
{
   /* Called when the watchdog timer expires.*/
   exit(0);
}
```

In this example, if one of the user code functions hangs for some reason, *gsi_g2_poll()* is not called every second, its watchdog timer counts down to zero, and *my_exit_func()* is then called by the system. This watchdog function merely exits.

A whenever rule in G2 can monitor the gsi-interface-status of interface objects, so that if a connection to G2 Gateway times out and closes, G2 deactivates and reactivates the corresponding GSI interface. This cleanly restarts the G2 Gateway bridge process.

gsi_watchdog() must be called at least once for its timer to begin. *gsi_initialize_context()* is a typical function from which to call the watchdog function, because the G2 Gateway bridge calls *gsi_initialize_context()* when it starts to initialize the data objects in G2 Gateway. Another function in which you could include a call to *gsi_watchdog()* is *gsi_resume_context()*.

You most likely will want the functions *gsi_pause_context()* and *gsi_shutdown_context()* to call *gsi_watchdog()* with a *period* of 0 to disable the timer, so that neither one causes G2 Gateway to time out.

gsi_watchdog() can be used to reset the timer in *gsi_g2_poll()*, which is called each second if the poll-external-system-for-data attribute of the GSI interface has the value yes.

If you are controlling the processing loop by running G2 Gateway in one-cycle mode, include a single call to *gsi_watchdog()* after the call to *gsi_run_loop()*.

Memory Management Responsibilities of G2 Gateway User Code

For most purposes, G2 Gateway automatically manages memory for data structures and the strings and arrays that they contain. However, for some purposes, your G2 Gateway user code can or must manage memory for these items.

Managing Data Structures

Your G2 Gateway user code *is* responsible for deallocating memory for any data structures that it has allocated itself. Your user code can allocate data structures using the API functions *gsi_make_attrs()*, *gsi_make_attrs_with_items()*, *gsi_make_items()*, or *gsi_make_registered_items()*. Your user code can reclaim this memory whenever it has no further use for it, using the API functions *gsi_reclaim_attrs()*, *gsi_reclaim_attrs_with_items()*, *gsi_reclaim_items()*, or *gsi_reclaim_registered_items()*.

Your G2 Gateway user code does not need to deallocate any *gsi_registration*, *gsi_registered_item*, *gsi_item*, or *gsi_attr* structures that G2 Gateway allocates automatically, and should not attempt to do so:

- G2 Gateway automatically allocates a *gsi_registration* structure for each item that is registered with the G2 Gateway bridge. A *gsi_registration* structure lasts until the corresponding item is deregistered. Your user code cannot either allocate or deallocate *gsi_registration* structures.
- G2 Gateway automatically allocates *gsi_registered_item*, *gsi_item*, or *gsi_attr* data structures when it invokes callback functions. These data structures last only for the duration of the current invocation of the callback.

To determine whether a given data structure was created by user code or generated automatically by G2 Gateway, use the API function *gsi_owner_of()*. For information about this function, see API Functions.

Managing Arrays and Lists

API functions that set the values of arrays and lists with arrays or lists provided by your user code can be grouped into two categories:

- Functions that make copies of the arrays or lists that your user code passes to them. These include all API functions that set an array or list to elements of a *particular type*, such as `gsi_set_flt_array()` or `gsi_set_sym_list()`.
- Functions that do not make copies of the arrays or lists. There are two such functions: `gsi_set_elements()` and `gsi_set_attrs()`.

The API functions that set arrays or lists to elements of a particular type all make copies of the arrays or lists that your user code passes to them. If your user code has no further use for the elements after these API functions complete, it can deallocate the elements.

For example, the API function `gsi_set_flt_array()` makes a copy of a floating-point array that user code passes to it and writes this copy into a `gsi_item`. If this array is located in a block of memory that the user code previously allocated with `malloc()`, it can `free()` this memory after `gsi_set_flt_array()` completes if it no longer needs the memory.

In contrast, the API functions `gsi_set_elements()` and `gsi_set_attrs()` do not make copies of the arrays that your user code passes to them. These functions accept existing arrays that your user code has allocated:

- The function `gsi_set_elements()` accepts an array of `gsi_item` elements that your user code has allocated by calling `gsi_make_items()`. `gsi_set_elements()` uses this array to set the `value` component of a `gsi_item`. Your user code can call `gsi_reclaim_items()` to deallocate the array after `gsi_set_elements()` completes, it if no longer needs the array.
- The function `gsi_set_attrs()` accepts an array of `gsi_attr` elements that your user code has allocated by calling `gsi_make_attrs()`. `gsi_set_attrs()` uses this array to set the `attribute(s)` component of a `gsi_item` structure. Your user code can call `gsi_reclaim_attrs()` to deallocate these structures after `gsi_set_attrs()` completes, if it no longer needs them.

API functions that return pointers to arrays of particular data types do not make copies of the arrays. The arrays persist only as long as the data structures that contain them. If your user code needs to use the arrays after the data structures have been deallocated, it must make copies of the arrays before the data structures are deallocated.

For example, the API function

```
double *gsi_flt_array_of(item)
```

returns an array of floating-point values stored in a `gsi_item` structure specified by *item*. This array persists only as long as the `gsi_item` persists. If your user code

needs to use this array after the *gsi_item* deallocated, your user code must make a copy of the array before it is deallocated.

The API functions *gsi_elements_of()* and *gsi_attrs_of()* do not allocate any new memory. Their return value point to existing arrays or lists stored in G2 Gateway data structures. These array and lists, and their elements, persist as long as the data structures that contain them.

Reclaiming Memory

Any memory you create with C functions such as *malloc()* is your responsibility to reclaim with *free()*. G2 Gateway will not do this, even for memory you have stored as user data.

Write Buffer Management

Each context in G2 Gateway has its own write buffer of fixed size (999,999 bytes). Every G2 Gateway API that sends data to G2, such as *gsi_return_values()*, places the data in the write buffer for the specified context and then flushes the buffer out to G2.

Using and Disabling Abbreviated Function Name Aliases

The header file *gsi_main.h* defines macros for abbreviated names of G2 Gateway API functions and G2 Gateway types. For example:

```
#define NULL_TAG GSI_NULL_TAG
#define context_socket(c) gsi_context_socket(c)
```

You can define the *__GENSYM_NOALIAS__* C preprocessor flag to force the *gsi_main.h* header file to omit the abbreviated names and macros for G2 Gateway API functions and types. Omitting abbreviated names can avoid conflicts between the abbreviated names used by the user code and the external system.

Be sure to define *__GENSYM_NOALIAS__* *before* you include *gsi_main.h*. For example:

```
/* G2 Gateway */
#define __GENSYM_NOALIAS__
#include "gsi_main.h"
```

For more info on defining C preprocessor flags see Defining C Preprocessor Flags.

Note This manual uses the unabbreviated names of API and callback functions.

Using and Disabling ANSI C Prototypes for API Functions

By default, the G2 Gateway header file *gsi_main.h* defines ANSI prototypes for the API functions of GSI. These prototypes are used whenever the bridge source program is compiled with an ANSI-compliant C compiler. These compilers define a preprocessor flag named __STDC__ to indicate that the compiler is using ANSI C.

However, you can disable the ANSI prototypes and use the Kernighan and Ritchie style function declarations. To do this, you must compile your G2 Gateway code with the *__GENSYMKR__* C preprocessor flag defined or use the corresponding compile time switch.

However, you can disable the ANSI prototypes and use the Kernighan and Ritchie style function declarations. To do this, define the C preprocessor flag *__GENSYMKR__* before you include the header file *gsi_main.h*. For example:

```
/* G2 Gateway*/
#define __GENSYMKR__
#include "gsi_main.h"
```

For more info on defining C preprocessor flags see Defining C Preprocessor Flags.

4

Remote Procedure Calls

Describes how a G2 Gateway bridge and a G2 application can make remote procedure calls to each other.

Introduction

Remote procedure calls provide a powerful and flexible means of communication between a G2 and a G2 Gateway bridge. Through remote procedure calls:

- A G2 procedure can invoke user-written C functions in a G2 Gateway bridge. The user-written functions are known as **G2 Gateway local functions**.
- A G2 Gateway bridge can invoke user-written G2 procedures or methods of G2 objects.

Kinds of Data that G2 Can Pass to G2 Gateway

When a G2 procedure invokes a G2 Gateway local function, it can pass the following kinds of data to the local function:

- Variable or parameter, or simple values such as *float* or *int*, whether directly computed or as the value of a G2 variable or parameter.
- References to items, including both items that inherit from the object class and those that do not.
- Copies of items, including both items that inherit from the object class and those that do not.

Kinds of Data that G2 Gateway Can Pass to G2

When a G2 Gateway bridge invokes a G2 procedure or method, it can pass the following kinds of data to the procedure or method:

- Variable or simple values whose types map to G2, such *double* or C integer types.
- References to items all classes, including both items that inherit from the object class and those that do not.
- Copies of data structures, which G2 uses to create instances of existing G2 classes. These can be either:
  - Data structures that your G2 Gateway user code allocates by calling the API functions *gsi_make_item()*, *gsi_make_items()*, *gsi_make_array()*, *gsi_make_attrs_with_items()*, or *gsi_make_attrs()*. These become genuine items when they reach G2, no different from items local to that G2.
  - Data structures that G2 Gateway previously allocated to represent a copy of an object that it received from G2 through a remote procedure call.

Making Remote Procedure Calls from G2 to the G2 Gateway Bridge

To enable G2 to make remote procedure calls to G2 Gateway local functions:

1. Write the G2 Gateway local functions that G2 can call or start, making certain that you include the required arguments.
2. Declare the G2 Gateway local functions in your bridge user code, using the API function *gsi_rpc_declare_local()* within the *gsi_set_up()* callback function.

   For information about how to write and declare local functions, see Writing a G2 Gateway Local Function to be Called by G2.
3. In G2, declare each G2 Gateway local function as a remote procedure. For information about how to do this, see Declaring the G2 Gateway Local Function in G2.
4. Write G2 procedures that start or call the G2 Gateway local functions.

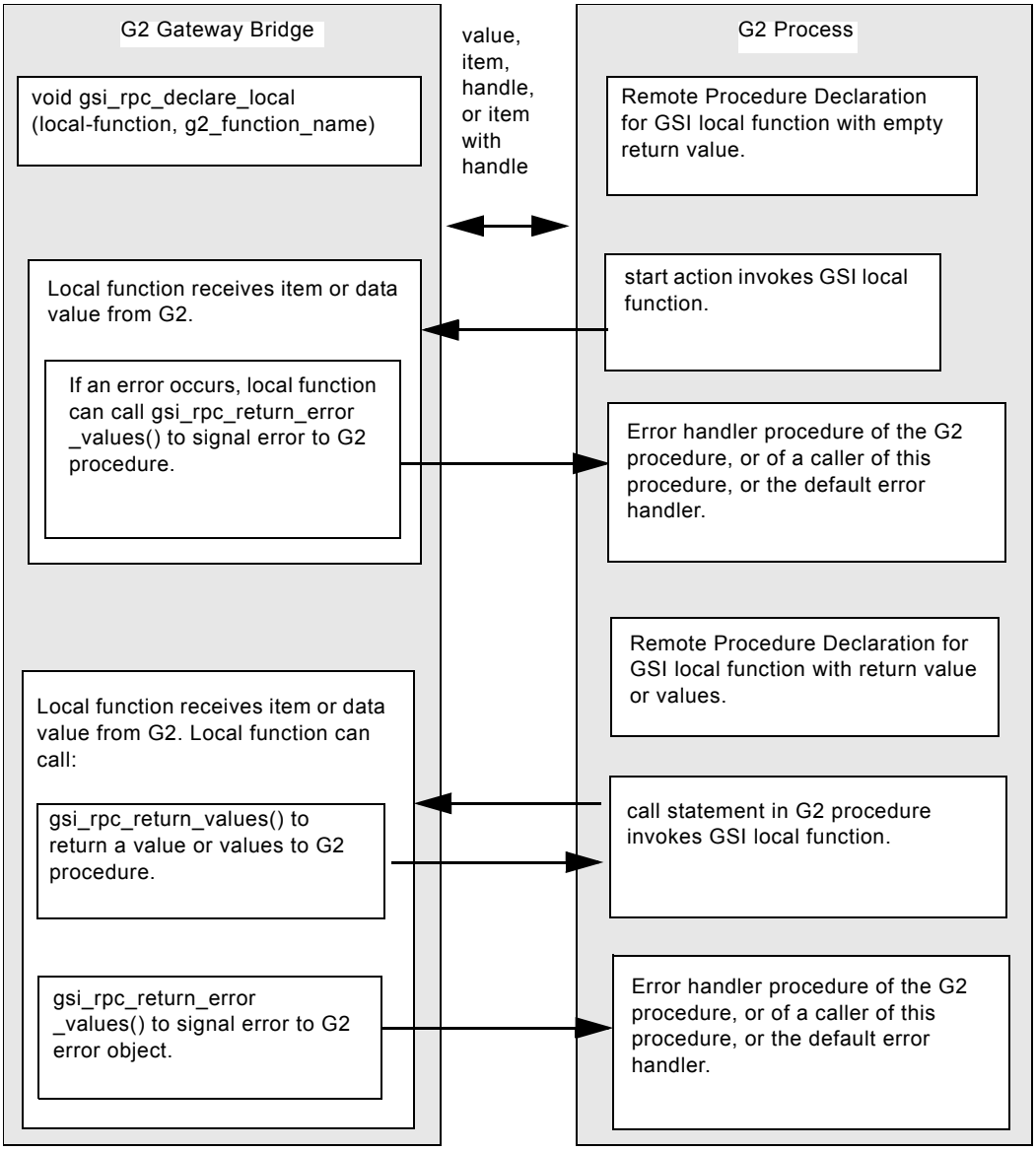
   G2 procedures start G2 Gateway local functions that do not return data to G2.

   G2 procedures call G2 Gateway local functions that return data to the G2 procedures. You can write these G2 procedures to use any data returned by the G2 Gateway local functions.

   Local functions that G2 invokes with either call or start can return error information to G2, if you code them to call the API function *gsi_rpc_return_error_values()*.

The following figure summarizes how G2 can invoke G2 Gateway local functions as remote procedures.

Invoking G2 Gateway Local Functions from G2



Note G2 can make remote procedure calls only to user-defined functions (G2 Gateway local functions). G2 cannot make remote procedure calls to any of the standard functions provided with G2 Gateway.

Writing a G2 Gateway Local Function to be Called by G2

Each G2 Gateway local function that you want to call from G2 must have the arguments: *rpc_arguments*, *count*, and *call_identifier*. It can optionally have a *procedure_user_data* argument. In the description below, arguments enclosed in square brackets [ ] are optional.

The syntax is:

```
void local_function (procedure_user_data, rpc_arguments,
    count, call_identifier)
    [gsi_procedure_user_data_type procedure_user_data;]
    gsi_item *rpc_arguments;
    gsi_int count;
    gsi_call_identifier_type call_identifier;
```

where:

local_function is the unique name of the G2 Gateway local function.

procedure_user_data is user data that G2 associates with the call to G2 Gateway. For information about the use of the *procedure_user_data* argument, see Call Identifiers and Procedure User Data. This argument is enabled only if the compile time switch `GSI_USER_DATA_FOR_CALLBACKS` is set. This switch is set automatically if you compile your G2 Gateway application with the `GSI_USE_USER_DATA_FOR_CALLBACKS` C preprocessor flag defined or you use the corresponding compile time switch. For information about these flags and options, see Preprocessor Flags and Runtime Options.

rpc_arguments is an array of `gsi_item`. Items passed from G2 to G2 Gateway are stored as elements of this array.

count is an integer that indicates the number of `gsi_item` structures in the array.

call_identifier is an integer that G2 generates to identify a particular remote procedure call to a G2 Gateway local function, within the current context. The API functions `gsi_rpc_return_values()` and `gsi_rpc_return_error_values()` reference *call_identifier* to indicate which outstanding remote procedure call within a specified context to return values to in G2. If the G2 Gateway local function is invoked by a start action in G2, the *call_identifier* argument of the local function is set to `CALL_HANDLE_OF_START`.

You must declare each receiver function in your header file. As a convenience, you can use the macro `declare_gsi_rpc_local_fn` to create the appropriate prototype declaration. The syntax is:

```
specifier declare_gsi_rpc_local_fn(local_function_name);
```

For example:

```
static declare_gsi_rpc_local_fn(my_local_function);
```

Note If you are using a C compiler that supports ANSI C, these *declare-* macros generate prototype declarations for the callbacks; otherwise, Kernighan and Ritchie style declarations are generated.

The return value of a G2 Gateway local function must be declared *void*.

Returning Values to G2 Through a G2 Gateway Local Function

When a G2 Gateway local function is invoked by a G2 call action, it can call the API function *gsi_rpc_return_values()* to return values to G2.

Note If the G2 Gateway local function is invoked by a start action in G2, the local function should not attempt to return values to G2, because G2 is not expecting it to return any values. However, the local function can call *gsi_rpc_return_error_values()* to signal an error to G2.

The first argument to *gsi_rpc_return_values()* is an array of GSI items (type *gsi_item **). To provide this argument, you can either allocate your own GSI items or reuse the array of GSI items pointed to by the *rpc_arguments* argument of the local function. For more information about *gsi_rpc_return_values()*, see API Functions.

If you reuse the item array received by the local function in *rpc_arguments* to return items to G2, do not change the values of these items before you finish reading them, and do not try to add items to *rpc_arguments* before returning it to G2. However, you do not need to return all the items that the G2 Gateway local function received in *rpc_arguments*.

If you allocate your own GSI items as return values, you must either reclaim them when you no longer need them, or allocate them once in *gsi_set_up()* and reuse them repeatedly. If you allocate them in *gsi_set_up()*, the arguments persist until the bridge process terminates.

The following example illustrates a G2 Gateway local function named *addnums* that adds any number of integers and returns their sum to G2:

```
void addnums(arguments, count, call_identifier)
gsi_item *arguments;
gsi_int count;
gsi_call_identifier_type call_identifier;
{
   gsi_int i, sum = 0;
   gsi_item return_value;
   gsi_item *return_value_pointer;

   return_value_pointer = gsi_make_items(1);
   return_value = *return_value_pointer;

   for (i = 0; i < count; i++, arguments++)
      sum += gsi_int_of(arguments);

   gsi_set_int(return_value,sum);
   gsi_rpc_return_values(&return_value, 1,
      call_identifier,gsi_current_context());
   gsi_reclaim_items(return_value_pointer);
}
```

In this example, the function *addnums* calls the following API functions:

- *gsi_make_items()*, which allocates the item pointed to by *return_value_pointer*. G2 Gateway automatically allocates and reclaims data structures for items that the bridge receives from G2. The *gsi_make_items()* function is included in this example to illustrate how to allocate data structures to represent values that originate in the bridge or in an external system and were not previously sent to the bridge by G2.
- *gsi_int_of()*, which gets the integer values of the items pointed to by *arguments*
- *gsi_set_int()*, which sets the integer value of *return_value* to the sum of the integer values in the items pointed to by *arguments*.
- *gsi_rpc_return_values()*, which returns the item to the calling procedure in G2.
- *gsi_reclaim_items()*, which frees the memory allocated by the call to *gsi_make_items()*.

Setting Behavior for Writing to G2 Lists or Arrays

Two internal G2 Gateway flags determine the behavior of *gsi_rpc_return_values()* when it is used to return values to a G2 list or array:

- *item_append_flag*.
  When this flag is set (turned on), *gsi_return_values()* appends the contents of a *gsi_item* with a list or array type to values in an existing G2 list or array. You can turn this flag on and off using the function *gsi_set_item_append_flag()*.
- *update_items_in_list_or_array_flag*.
  When this flag is set (turned on), the attribute values of items in a G2 list or array are updated with attribute values of the corresponding items in a list or array returned to G2 by *gsi_return_values()*. You can turn this flag on and off using the function *gsi_set_update_items_in_list_or_array_flag()*.

The default behavior when either of these flags is set is to overwrite the contents of the G2 list or array.

Declaring the Local Function in Your G2 Gateway User Code

After you write a G2 Gateway local function, you must declare that function in your G2 Gateway user code, so that G2 Gateway knows which function to execute when G2 calls the function.

To declare G2 Gateway local functions, use the API function *gsi_rpc_declare_local()*. Call this API function from *gsi_set_up()* or from a function called from *gsi_set_up()*. In the description below, arguments enclosed in square brackets [ ] are optional. The syntax is:

```
void gsi_rpc_declare_local(local_function, procedure_user_data,
    g2_function_name)
    gsi_rpc_local_fn_type *local_function;
    [gsi_procedure_user_data_type procedure_user_data;]
    char  *g2_function_name;
```

where:

local_function is a pointer to the G2 Gateway local function.

procedure_user_data is user data associated with the call that G2 makes to the local function. For information about the use of the *procedure_user_data* argument, see Call Identifiers and Procedure User Data. This argument is enabled only if the compile time switch *GSI_USER_DATA_FOR_CALLBACKS* is set. This switch is set automatically if you compile your G2 Gateway application with the *GSI_USE_USER_DATA_FOR_CALLBACKS* C preprocessor flag defined or you use the corresponding compile time switch. For information about these flags and options, see Preprocessor Flags and Runtime Options.

g2_function_name is a string that gives the name of the G2 Gateway local function. The string must match exactly the string specified in the name-in-remote-system attribute of the remote procedure declaration in G2.

For example, the following call to *gsi_rpc_declare_local()* declares a function named *receive_and_return_copy()* as a G2 Gateway local function. In G2, the name-in-remote-system attribute of the remote procedure declaration that invokes this procedure is receive-and-return-item-copy.

```
gsi_rpc_declare_local(receive_and_return_copy,
    "RECEIVE-AND-RETURN-ITEM-COPY");
```

Note It is not necessary to declare G2 Gateway local functions more than once. For this reason, it is good practice to invoke *gsi_rpc_declare_local()* in the *gsi_set_up()* callback function, which is called only once during the life of the G2 Gateway process. For information about *gsi_set_up()*, see *gsi_set_up*.

Declaring the G2 Gateway Local Function in G2

You must declare the G2 Gateway local function as a remote procedure within your G2 knowledge base, so that G2 can call or start the function as a remote procedure. The declaration tells G2 the name of the function, the number and type of arguments that it requires, and the number and type of values, if any, that it returns to G2.

To declare the G2 Gateway local function as a remote procedure that you can invoke from your G2 application:

1. Select KB Workspace > New Definition > remote-procedure-declaration.
2. In the G2 text editor window that appears, create the remote procedure declaration. The syntax is:

   declare remote *remote-procedure-name* (*argument-type(s)*) = (*return-type(s)*)

   where:

   remote-procedure-name specifies the name of the G2 Gateway local function, as it is known in G2. This is the name used in G2 to call or start the local function. By default, the name you specify here will be duplicated as a string in upper case letters in the name-in-remote-system attribute in the attribute table of this remote procedure declaration.

   argument-type(s) are the G2 data types of the arguments passed to the local function. Argument types can be simple data types (float, integer, truth-value, symbol, or text), compound data types (sequence and structure), abstract data types (item-or-value, value, quantity), or items. For

more information about how to specify *argument-type(s)*, see Grammar for G2 Remote Procedure Argument Declarations.

return-type(s) are G2 data types of the values returned by the function. Return values can be item classes, as handles or objects, or data types of float, integer, truth-value, symbol, or text.

For example:

```
declare remote addnums(integer, integer) =
    (integer)
```

In this example, the name of the G2 Gateway local function declared as a remote procedure in G2 is addnums. The two arguments and the return value of the remote procedure are declared as integer values. The local function must call the G2 Gateway API function `gsi_rpc_return_values()` to return the integer value to the calling procedure in G2.

For more information about declaring a G2 Gateway local function as a remote procedure, see the discussion of the G2-to-G2 interface in the *G2 Reference Manual*.

3 Display the attribute table of the remote procedure declaration to set the name-in-remote-system attribute to a string that specifies the name of the local function in G2 Gateway. For example:

ADDNUMS, a remote-procedure-declaration	
Notes	OK
Authors	jfb (7 Jun 1997 8:42 a.m.)
Change log	0 entries
Item configuration	none
declare remote addnums(integer, integer) = (integer)	
Name in remote system	"ADDNUMS"

The name-in-remote-system string must be identical to the *g2_function_name* argument of the `gsi_rpc_declare_local()` function called in the G2 Gateway user code to declare the local function. You must enclose the string in double quotation marks (“”).

For detailed information about how to declare a remote procedure, see the *G2 Reference Manual*.

Caution If you start the remote procedure, be careful not to modify or delete any G2 item that you are passing through the remote procedure call until the item has been successfully passed to G2 Gateway. If you modify the item before it is fully passed, it may be passed with some of your modifications. If you delete the item before it is passed, the item may not be passed successfully, with unpredictable results.

To avoid this problem, call the remote procedure (rather than start it), and call *gsi_rpc_return_values()* in your bridge to send an acknowledgment to G2 indicating that the item has been received. When G2 receives the acknowledgment, it is safe to modify or delete the item. To send the acknowledgment, you can use a call of the form:

```
gsi_rpc_return_values(NULL_PTR, 0, call_identifier,
   gsi_current_context());
```

Grammar for G2 Remote Procedure Argument Declarations

G2 remote procedure declarations enable you to specify the following information about remote procedure calls:

- The number and type of the arguments passed to the local function in G2 Gateway.
- The number and type of the values returned to G2 by the local function.

Specifying the Data Type of Arguments in the Remote Procedure Call

For the data types of arguments to remote procedure calls, you can specify the following G2 data types:

- Simple data types (float, integer, truth-value, symbol, or text)
- Compound data types (sequence and structure)
- Abstract data types (item-or-value, value, quantity)
- Items

A G2 application can send G2 items with their attributes to the G2 Gateway bridge. G2 can pass items through arguments of any type except the simple data types. Note that:

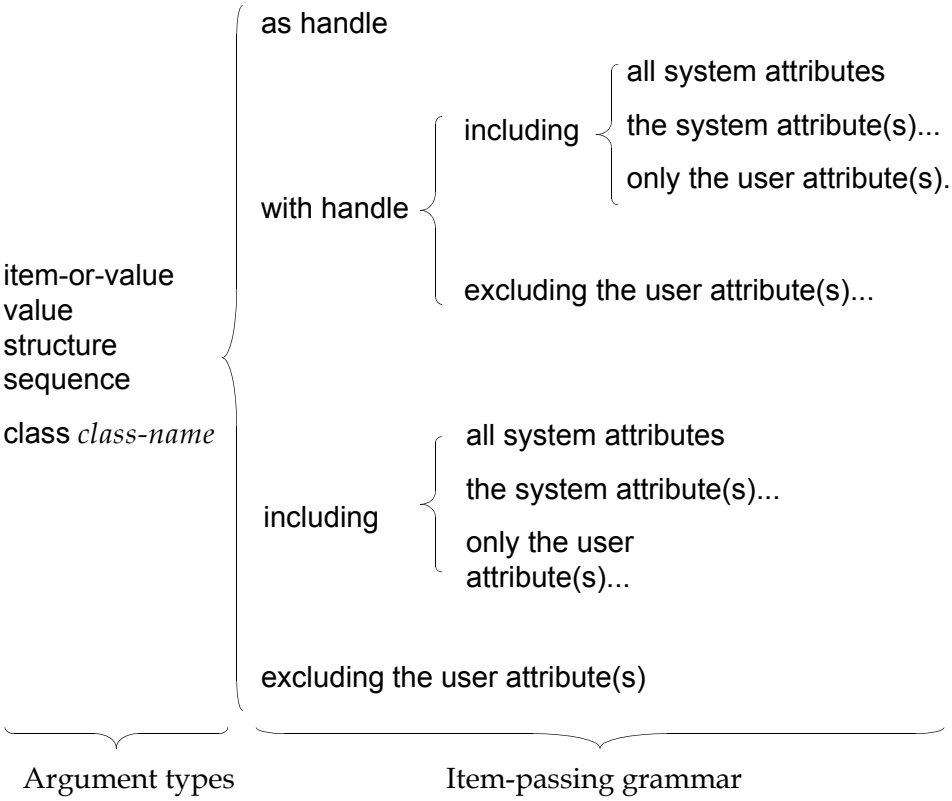
- An item-or-value can be an item, a value, a sequence, or a structure.
- A value can be a sequence, a structure, or any other value.
- A sequence or structure can contain items, any value types, or any other sequence or structure.

When you declare an argument of any data type through which you can pass items, you can include item-passing grammar that enables you to specify:

- Whether to pass a copy of the item, only a handle that refers to the item, or a copy with a handle.
- If you are passing a copy, which attributes of the copy to pass.

The following figure illustrates the item-passing grammar that you can use in G2 remote procedure declarations:

Item-Passing Grammar for G2 Remote Procedure Declarations



If you do not specify any item-passing grammar, the item is passed by default as a copy, with no handle, with all of its user-defined attributes, and with none of its system attributes.

The following table lists the elements of item-passing grammar and describes their meanings:

Elements of Item-Passing Grammar

Grammar	Meaning
as handle	Pass only a reference to the item. Do not pass a copy of the item, or any attributes of the item. **Note**: When you pass a sequence or a structure using as handle, the sequence or structure itself is passed as a copy, and each item contained in the sequence or structure is received by G2 Gateway as a *gsi_item* with the G2 Gateway type tag *GSI_HANDLE_TAG*. A sequence or a structure that is contained within the attribute of another sequence or a structure is processed in exactly the same way as the outer, containing sequence or a structure: the embedded sequence or structure is passed as a copy, and any items that it contains are passed as items that have the G2 Gateway type tag *GSI_HANDLE_TAG*.
with handle	Pass both a copy of the item as of the time of the call and a reference to it.

Elements of Item-Passing Grammar

Grammar	Meaning
including	Specifies the attribute or attributes to pass when you pass a copy of an item. The following grammar enables you to specify which attributes are passed: • all system attributes Pass only the system attributes of the item. By default, no system attributes are passed. For information on system attributes see the *G2 Reference Manual*. • the system attribute(s) Pass only the system attribute or attributes that you specify by name. • only the user attribute(s) Pass only the user attributes that you specify by name. By default, all user attributes are passed.
excluding the user attributes	Specifies the user -defined attributes that are not passed. By default, all user attributes are passed.

Note The item-passing grammar with handle can be used only following class *classname*, and cannot be used to pass sequence or structure types.

For example, the following remote procedure declaration passes an item-or-value object with no system attributes and all user-defined attribute except temperature, pressure, and volume:

```
declare remote tank-data-function(item-or-value
    excluding the user attributes: temperature, pressure, volume) = (integer)
```

For more information about G2 remote procedure declarations, see the *G2 Reference Manual*.

Passing Attributes of Structures and Sequences

All attributes of any structure or sequence are passed with the structure or sequence; you cannot select which attributes to pass. However, for those attributes of the structure or sequence that are items in G2, you can specify which attributes of the items in the structure or sequence are passed with those item. To do this, you use the item-passing grammar for remote procedure declarations.

For example, the following remote procedure declaration specifies that all system attributes of each item in a sequence are passed to G2 Gateway with that sequence:

```
declare remote tanks-array-function(sequence
    including all system attributes) = (integer)
```

Invoking the G2 Gateway Local Function from G2

G2 can make a remote procedure call to any of your G2 Gateway local functions, either by a call statement or by a start action. G2 supports up to 4096 simultaneous outstanding calls.

Invoking a G2 Gateway Local Function that Returns Values to G2

If the G2 Gateway local function in the bridge user code is to return values to G2, it should be invoked from G2 by a call procedure statement. The call statement can be used only in G2 procedures.

Caution Do not use the start action to invoke a G2 Gateway local function that needs to return a value to G2.

After G2 executes the call statement within a procedure, the procedure waits indefinitely for the function accessed by the remote procedure call to complete and return a value. When the calling procedure receives the return value, it resumes execution. If G2 Gateway returns an error through *gsi_return_timed_values()*, G2 looks for an error handler. If G2 Gateway closes the connection over which the remote procedure call was made, G2 aborts the calling procedure.

Note If the remote procedure call is one branch of a do in parallel [until one completes] statement, and the other branch statement finishes first, G2 may abort the remote procedure call but continue to process the calling procedure. If that happens, data is not returned from the remote procedure call.

The call statement in G2 has the same priority as the procedure that contains it. This priority has no effect on calls when they are handled by the bridge process.

The syntax is:

[*x*, *y* ...] = call *remote-procedure-name* (*arguments*) across *gsi-interface-object*

where:

x, *y* are names of G2 objects (such as GSI variables) in which G2 places the values returned by the called G2 Gateway function. The number and order of the objects [*x*, *y*...] must match the number and order of the values returned. If any objects are returned, they will be transient within G2. To prevent a memory leak, you must either delete them or make them permanent.

remote-procedure-name is the name of the remote procedure declaration specifying the local function called by your G2 Gateway bridge.

arguments are the supplied argument(s) used by the called function. Separate the arguments with commas.

gsi-interface-object is the name of the GSI interface used by G2 to connect to the bridge.

An example is:

```
sum = call addnums (integer1, integer2) across interface-10
```

In this example, sum, integer1, and integer2 are local variables in the G2 procedure, addnums is the name of the remote procedure declaration that specifies the G2 Gateway local function in the bridge that is called by G2, and interface-10 is the name of the GSI interface used by G2 to access the bridge. When addnums completes, it returns a value to sum.

Note If you use a G2 call statement to call a G2 Gateway local function that does not return values, you should include in the local function a call to the API function *gsi_rpc_return_values()*, with a 0-length argument list.

Invoking a G2 Gateway Local Function that Does Not Return Values to G2

If the G2 Gateway local function in the G2 Gateway user code does not return values to G2, use a G2 start action to invoke the function. You can use start actions in both rules and procedures.

When G2 uses a start action to invoke a G2 Gateway local function, G2 continues to execute the rule or procedure that contains the start action, without waiting for the started function to complete. This is true even if the function does not exist or if an error occurs in starting the function. If an error occurs, it can be recorded on the Operator's Logbook.

Use this syntax to start a local G2 Gateway function:

start *remote-procedure-name* (*arguments*) across *gsi-interface-object(s)*

where:

remote-procedure-name is the name of the remote procedure declaration that specifies the local G2 Gateway function called by G2.

arguments are the arguments passed to the G2 Gateway procedure. Use commas to separate arguments.

gsi-interface-object is the name of the GSI interface used by G2 to connect to the bridge, or a list or array of GSI interface objects across which you can broadcast the start action. The list or array can contain g2-to-g2-data-interface and/or gsi-interface objects in any combination.

For example:

```
start sendnums (integer1, integer2) across interface-10
```

where integer1 and integer2 are variables in the G2 knowledge base, sendnums() is the name of the remote procedure declaration for the function in the bridge that is called by G2, and interface-10 is the name of the GSI interface used by G2 to access the bridge. sendnums() does not return a value to G2.

Alternatively:

```
start rem-procs (10) across interface-array
```

which starts rem-procs with the argument 10 across every interface in interface-array.

Enhancing Performance when Using the start Action

A broadcast start can execute several times faster than a series of single starts. This improvement requires changes that are not available in versions of G2 and G2 Gateway prior to 5.1. It also requires that a connection not use message interleaving, as described in "

To maximize the performance of a broadcast start:

- Specify only interfaces to G2 or G2 Gateway Version 5.1 or higher.
- Use only interfaces that do not use message interleaving.

If either of these rules is violated, the broadcast start will execute correctly, but its performance advantage will not be fully realized.

Passing a Varying Number of Arguments to the Same G2 Gateway Local Function

Your G2 application may need to make remote procedure calls with varying numbers of arguments to the same G2 Gateway local function. Every G2 Gateway local function has a *count* argument indicating the number of arguments that it receives from G2 in any particular call.

If your G2 application needs make calls to the same local function with varying numbers of arguments, you can either:

- Make a single G2 remote procedure declaration that passes a varying number of arguments to the G2 Gateway local function. To do this, use the all remaining grammar to allow the remote procedure declaration to accept a varying number of arguments. Declare the all remaining arguments to be of the type quantity, value, or item-or-value.

  For example, the following remote procedure declaration accepts an integer argument and a text argument, followed any number of arguments that are declared item-or-value:

  ```
  declare remote my-rpc (integer, text, all remaining item-or-value) = (integer)
  ```

- Make several different G2 remote procedure declarations, each of which passes a different, but fixed, number of arguments to the G2 Gateway local function.

  In each G2 remote procedure declaration, specify a different *remote-procedure-name* value and a different number of arguments, but specify the same string in the name-in-remote-system attribute.

  For example, if your G2 Gateway user code includes a local function that can find the sum of any number of values received as arguments, you can create several remote procedure declarations for this G2 Gateway local function. The remote procedure declarations specify different numbers of arguments, but they all specify the same string in their name-in-remote-system attributes. The G2 Gateway local function can find the sum of the arguments passed to it by G2 through any of these remote procedure declarations.

How a Local Function Can Process Argument Arrays Received from G2

In G2 Gateway, a G2 item is represented by a `gsi_item` structure. The user-defined attributes of the G2 item are represented by an array of `gsi_attr` structures; each element of this array represents one attribute. Each element has the same data type as the G2 item attribute that it represents.

To access `gsi_item` or `gsi_attr` structures, your user code must call API functions provided with G2 Gateway. For example, it can call `gsi_attrs_of()` to get the attributes of a `gsi_item` structure, `gsi_attr_name_of()` to get the name of a `gsi_attr` structure, and other API functions to get similar kinds of information. For information about these API functions, see API Functions..

Note Your user code cannot access `gsi_item` or `gsi_attr` structures directly. It must use the API functions provided with G2 Gateway.

To process each element of the array that is received as *arguments*, the G2 Gateway local function can:

- Find out whether the element corresponds to a G2 item, using the API function `gsi_is_item()`.
- If the element corresponds to a G2 item:
  - Invoke `gsi_attr_count_of()`, which returns the number of attributes in the item.
  - Invoke `gsi_attrs_of()`, which returns an array containing the attributes of the item. You can use the value returned by `gsi_attr_count_of()` as the size of this array.
  - Loop over the array of attributes, using `gsi_is_item()` to determine whether each subattribute corresponds to a G2 item, and then process the attribute accordingly.

    For example, if `gsi_is_item` is true of an attribute, you can use `gsi_attrs_of()` to return its attributes to an array, and then loop over the elements of this array to process them as your application requires. If the attribute is a constant value, the G2 Gateway local function can evaluate and use the attribute value as your application requires.
- If the element is a not a G2 item, the G2 Gateway local function can evaluate and use the value as your application requires.

Making Remote Procedure Calls from a G2 Gateway Bridge to G2

Your G2 Gateway bridge user code can make remote procedure calls to user-defined G2 procedures and to methods of user-defined G2 classes.

To enable your G2 Gateway bridge to make remote procedure calls to a G2 KB:

1. Write the G2 procedures or G2 object methods that the G2 Gateway bridge can call.

   No special configuration is required in G2 for these procedures or methods.
2. In your G2 Gateway user code, declare the G2 procedures or methods as remote procedures, by calling `gsi_rpc_declare_remote()` or `gsi_rpc_declare_remote_with_error_handler_and_user_data()`within the `gsi_initialize_context()` callback function.

For each procedure or method that you declare, you must create a global variable called a **function handle**. The type of a function handle variable must be *gsi_function_handle_type*. The function handle holds a pointer to your receiver function (see below).

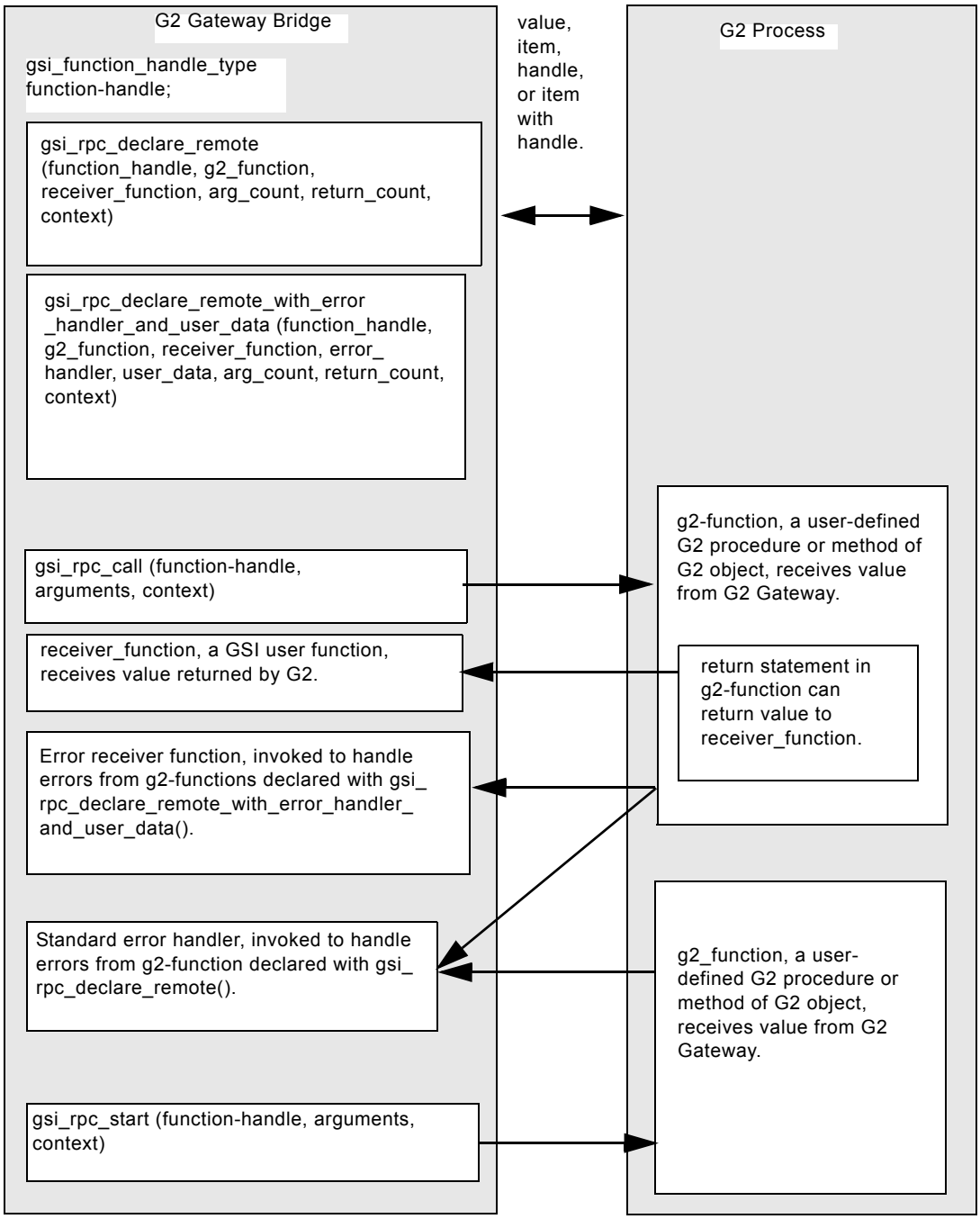
Your G2 Gateway user code can use the API function *gsi_rpc_start()* to invoke a G2 procedure or method that does not return values to the bridge, or *gsi_rpc_call()* or *gsi_rpc_call_with_count()* to invoke a G2 procedure or method that returns values to the bridge.

3 If a G2 procedure or method that your bridge invokes as a remote procedure is to return values to the G2 Gateway bridge, you must write a function in your G2 Gateway user code that can receive these return values from G2.

Such a function is called a **receiver function**.

The following figure summarizes how a G2 Gateway bridge can invoke G2 procedures and methods:

Invoking G2 Procedures or Methods from a G2 Gateway Bridge



Writing the G2 Procedure or Method to be Invoked by G2 Gateway

To write G2 procedures or methods to be invoked by G2 Gateway, you can use the same syntax that you use to write any other G2 procedures or methods. For information about how to write G2 procedures and methods, see the *G2 Reference Manual*.

The following example illustrates a G2 procedure that you can declare as a remote procedure in the bridge:

```
display-new-alarm(new-alarm: class alarm)
begin
    transfer new-alarm to the workspace of the item
    superior to this workspace at (alarm-x-pos,
    alarm-y-pos);
end
```

The following sections illustrate how to declare a G2 procedure or method as a remote procedure in the bridge.

Declaring the Remote Procedure in the Bridge

Recall that local functions are cross-context and can therefore be declared in `gsi_set_up()` but remote declarations are context-specific and are better declared in `gsi_initialize_context()`.

You must declare G2 procedures and methods to which the bridge makes remote procedure calls, using the API function `gsi_rpc_declare_remote()` or `gsi_rpc_declare_remote_with_error_handle_and_user_data()`. The first argument, *function_handle*, points to a global variable that enables the bridge to identify the remote procedure or method. In your G2 Gateway user code, you must create a function handle for each G2 procedure or method that you declare as a remote procedure.

Creating a Handle for the Remote Procedure

The G2 Gateway bridge process needs a `gsi_function_handle_type` global variable, or handle, to identify each G2 procedure or method that it invokes as a remote procedure. For each handle that you need, include a declaration of the handle in your user code before you use the handle in declaring or invoking the remote procedure. The syntax is:

`gsi_function_handle_type` *handle;*

where *handle* is the name by which the user code refers to the G2 procedure or method.

For example, if the name that you want to use in your user code to refer to the G2 procedure is *display_new_alarm*, you can declare the handle as:

```
gsi_function_handle_type display_new_alarm;
```

If you want to declare a G2 procedure as a remote procedure for use in more than one context, you can declare an array of *gsi_function_handle_type*, and use the number of the current context to index the array. This takes advantage of the fact that context numbers are contiguous integers, suitable for such array indices.

Using gsi_rpc_declare_remote() and gsi_rpc_declare_remote_with_error_handler_and_user_data()

To declare the G2 procedure or method as a remote procedure, include a call to the G2 Gateway API function *gsi_rpc_declare_remote()* or *gsi_rpc_declare_remote_with_error_handler_and_user_data()* once for every context in which the remote procedure is used. To do this, include the calls to these API functions in *gsi_initialize_context()*, which is called each time a new context is established.

The syntax of *gsi_rpc_declare_remote()* is:

```
void gsi_rpc_declare_remote(function_handle,
    g2_function_name,receiver_function, procedure_user_data,
    argument_count,return_count, context_number)
    gsi_function_handle_type *function_handle;
    gsi_char *g2_function_name;
    gsi_rpc_receiver_fn_type *receiver_function;
    [gsi_procedure_user_data_type procedure_user_data;]
    gsi_int argument_count;
    gsi_int return_count;
    gsi_int context_number;
```

where:

function_handle is a pointer to a global variable used in the G2 Gateway user code to identify the G2 procedure or method.

g2_function_name is the name of the G2 procedure or method as it is known in G2. This string must be enclosed in double quotation marks (“”), and must match the format of the procedure as it appears in G2.

If *g2_function_name* refers to a method of a G2 object, you can specify the method either by its name alone (generically), or by its name prefixed with the name of its class followed by two colons (directly). For example, you can specify the method fill of a G2 class flask as "*FILL*" (generically) or as "*FLASK::FILL*" (directly). For more information about how to invoke G2 methods, see the *G2 Reference Manual*. For more information about how to invoke a method, see Invoking Methods of G2 Items from G2 Gateway.

receiver_function is a pointer to the receiver function that receives values returned by the G2 procedure to the bridge, or *NULL_PTR* if no values are returned.

procedure_user_data is user data associated with this remote procedure call to G2. Procedure user data can be associated with the call only if the compile time switch *GSI_USER_DATA_FOR_CALLBACKS* is set. This switch is set automatically if you compile your G2 Gateway application with the *GSI_USE_USER_DATA_FOR_CALLBACKS* C preprocessor flag defined or you use the corresponding compile time switch. For information about these flags and options, see Preprocessor Flags and Runtime Options.

argument_count is the number of arguments passed to the remote procedure.

return_count is the number of values returned by the remote procedure to the bridge.

context_number is the context used by this function. The context identifies one particular connection to G2.

An example of a call to *gsi_rpc_declare_remote()* is:

```
gsi_rpc_declare_remote(&display_new_alarm,
    "DISPLAY-NEW-ALARM", NULL_PTR, 1, 0,
    gsi_current_context());
```

The G2 procedure is known in G2 as display-new-alarm. In your bridge user code, you can refer to the procedure using the handle variable *display_new_alarm*.

The API function *gsi_rpc_declare_remote_with_error_handler_and_user_data()* is similar to *gsi_rpc_declare_remote()*, but in addition it allows user data to be associated with this remote procedure call, and specifies an error receiver function that receives error values returned by the G2 procedure. The G2 procedure can signal error values to the error receiver function if it is invoked with *gsi_rpc_call()*, but not if it is invoked with *gsi_rpc_start()*.

Invoking Methods of G2 Items from G2 Gateway

When you use *gsi_rpc_call()* or *gsi_rpc_start()* to invoke a method of a G2 item, the first argument included in the list of arguments (*gsi_item***arguments*) passed to the G2 method must be the G2 item whose method is invoked.

This first argument passed to the G2 method can be either:

- a *gsi_item* with *GSI_HANDLE_TAG* for its *type tag* component, to reference the G2 item by handle. This call invokes the G2 method but does not pass attribute data to the G2 item.
- a *gsi_item* with a valid G2 class name for its *class name* component. This call causes G2 to create a new G2 item of the specified class, and passes attribute data to the specified method of that item.

Defining a Function to Receive Values Returned by G2

When the bridge process invokes a G2 procedure, it does not wait for the G2 procedure to complete and return values before continuing with processing. To make it possible for G2 to return values to the bridge, you must create a function in your G2 Gateway user code known as a **receiver function** to receive values from G2.

When the G2 procedure called by G2 Gateway has completed execution, G2 sends its return value to G2 Gateway, which causes G2 Gateway to invoke the receiver function. The receiver function then receives the return value.

The receiver function can perform any operations necessary for the application, including evaluation of the data returned by G2.

The syntax for a receiver function is:

```
void receiver_function (procedure_user_data, arguments,
   count, call_identifier)
   gsi_procedure_user_data_type procedure_user_data;
   gsi_item *arguments;
   gsi_int  count;
   gsi_call_identifier_type call_identifier;
```

where:

receiver_function is the unique name of the receiver function.

procedure_user_data is procedure user data associated with the call that G2 makes to the receiver function. The receiver function can receive procedure user data only if the compile time switch `GSI_USER_DATA_FOR_CALLBACKS` is set. To set this switch, you must compile your G2 Gateway code with the GSI_USE_USER_DATA_FOR_CALLBACKS C preprocessor flag defined or use the corresponding compile time switch. For information about these flags and options, see Preprocessor Flags and Runtime Options.

arguments is an array of `gsi_item`, which contains the data values that G2 is returning to the bridge process.

count is an integer specifying the number of values in the *arguments* array.

call_identifier is a user-specified call identifier value that G2 Gateway associated with its call to the G2 procedure and that G2 associates with its return call to the receiver function. The receiver function can receive a call identifier only if the compile time switch `GSI_USER_DATA_FOR_CALLBACKS` is set. To set this switch, you must compile your G2 Gateway code with the GSI_USE_USER_DATA_FOR_CALLBACKS C preprocessor flag defined or use the corresponding compile time switch. For information about these flags and options, see Preprocessor Flags and Runtime Options.

Note You can enable *procedure_user_data* and *call_identifier* using the compile time switch *GSI_USER_DATA_FOR_CALLBACKS*. This switch is set automatically if you compile your G2 Gateway application with the *GSI_USE_USER_DATA_FOR_CALLBACKS* C preprocessor flag defined or you use the corresponding compile time switch. For information about preprocessor flags and runtime options, see Preprocessor Flags and Runtime Options.

You must declare each receiver function in your header file. As a convenience, you can use the macro *declare_gsi_rpc_receiver_fn* to create the appropriate prototype declaration. The syntax is:

```
specifier declare_gsi_rpc_receiver_fn(receiver_function_name);
```

For example:

```
static declare_gsi_rpc_receiver_fn(my_receiver_function);
```

Note If you are using a C compiler that supports ANSI C, these *declare-* macros generate prototype declarations for the callbacks; otherwise, Kernighan and Ritchie style declarations are generated.

Defining a Function to Receive Error Values Returned by G2

When you declare a G2 procedure as a remote procedure that can be invoked by the G2 Gateway bridge, you can specify an error receiver callback function in the bridge to which G2 can signal error values in the case of an error. To do this, you must:

- Use the API function *gsi_rpc_declare_remote_with_error_handler_and_user_data()* to declare the G2 procedure as a remote procedure. For information about this function, see *gsi_rpc_declare_remote_with_error_handler_and_user_data*.
- Write an error receiver callback function in your G2 Gateway user code to which the remote G2 procedure can signal error values.

The error receiver function can perform any operations necessary for the application, including examination of the error data returned by G2. The error receiver function can signal an error back to G2 by call the API function *gsi_rpc_return_error_values()*.

The syntax for an error receiver function is:

```
void error_handler (arguments)
   gsi_item *arguments;
```

where:

arguments can be either:

- a *gsi_item* representing an error object in G2.
- a *symbolic-expression* and a *text-expression*, similar to the arguments of the signal G2 procedure statement.

These arguments are identical in meaning to the *error_arguments* in a call to *gsi_rpc_return_error_values()*. For more information about these arguments, see *gsi_rpc_return_error_values*.

For information about how to write error receiver functions, see Callback Functions.

Invoking the Remote G2 Procedure

After you declare remote procedures with *gsi_rpc_declare_remote()*, you can invoke them using the API functions *gsi_rpc_call()*, *gsi_rpc_call_with_count()*, and *gsi_rpc_start()*.

A G2 Gateway function containing a call to *gsi_rpc_call()*, *gsi_rpc_call_with_count()*, or *gsi_rpc_start()* continues after it executes the call, and does not wait for the remote procedure in G2 to complete.

A remote G2 procedure invoked by *gsi_rpc_start()* does not return values, so your G2 Gateway user code does not need to include a corresponding receiver function.

Calling a G2 Procedure that Returns Values to the Bridge

Use *gsi_rpc_call()* or *gsi_rpc_call_with_count()* if the procedure in G2 is to return values to the bridge. For detailed information about *gsi_rpc_call()* and *gsi_rpc_call_with_count()*, see API Functions.

gsi_rpc_call() and *gsi_rpc_call_with_count()* return immediately and do not wait for return arguments. The remote procedure in G2 is executed as soon as G2 receives the remote procedure call from G2 Gateway.

An example of a call to *gsi_rpc_call()* is:

```
gsi_rpc_call(display_new_alarm, alarm_args,
   gsi_current_context());
```

In this example, the handle of the remote procedure passed is *display_new_alarm*, and the pointer to the array of arguments is called *alarm_args*.

Your bridge user code must include a receiver function to accept the values returned to G2 Gateway by the remote procedure in G2. You specify the receiver function in the remote procedure declaration of the G2 procedure. For information about how to write receiver functions, see Defining a Function to Receive Values Returned by G2.

Note The receiver function is called only within the *gsi_run_loop()* call tree. If you are running your G2 Gateway bridge in one-cycle mode, you must call *gsi_run_loop()* to ensure that the receiver function is called.

Starting a G2 Procedure that Does Not Return Values to the Bridge

Use *gsi_rpc_start()* if the remote G2 procedure does not return values to the bridge.

An example of a call to *gsi_rpc_start()* is:

```
gsi_rpc_start(start_timer, timer_args,
   gsi_current_context());
```

In this example, the handle of the remote G2 procedure is *start_timer*, and the pointer to the array of arguments is called *timer_args*.

Passing Items from a G2 Gateway Bridge to G2

A G2 Gateway bridge typically sends data to a G2 application at the request of an external system. The bridge process can invoke a G2 Gateway user function to send an item and its attribute values to G2. The user function must do the following:

- Assemble the data to be passed to G2 into an array of *gsi_item* structures. The *gsi_item* structures must correspond to existing class-definitions in G2. To do this, your G2 Gateway user code must:
  - Allocate an array of the *gsi_item* structures that you want to pass to G2. To do this, invoke some combination of the API functions *gsi_make_items()*, *gsi_make_attrs()*, or *gsi_make_attrs_with_items()*.
  - Invoke *gsi_set_class_name()* to set the *class name* component of each *gsi_item* structure in the array that you allocated. Set the class name of each *gsi_item* to the name of an existing item definition in G2.

    gsi_set_class_name() does *not* reset the data type of the *gsi_item* If the *gsi_item* structure was previously used, its G2 Gateway type tag may not be appropriate for the class to which you are setting it. In this case, you must reset the type of the *gsi_item* to a G2 Gateway data type that is appropriate for the class. For a list of the API functions that set the types of G2 Gateway structures, see *gsi_set_type*. For information about the G2 Gateway type tags that correspond to G2 data types, see G2 Data Types and G2 Gateway Type Tags.
  - Populate the array with the items to be passed.

- Invoke *gsi_rpc_call()*, *gsi_rpc_call_with_count()*, or *gsi_rpc_start()* to pass the array of items to G2. If you have set *gsi_use_data_for_callbacks*, a call to any of these functions should specify the *call_identifier* value used in the call to your receiver function.
- Reclaim the items that you passed to G2. To do this, invoke the API functions *gsi_reclaim_items()*, *gsi_reclaim_attrs()*, or *gsi_reclaim_attrs_with_items()*, depending on how the items were created.

When G2 receives a *gsi_item* structure, it creates an item of the class specified by *gsi_set_class_name()*, in exactly the same way that a create action creates an item. Attributes for which G2 Gateway does not send values are given the default values in the specified G2 item definition.

Returning G2 Items from G2 Gateway Back to G2

To enable a G2 Gateway bridge to pass items back to a G2 that originated there, the API functions *gsi_return_values()*, *gsi_return_attrs()*, *gsi_return_timed_values()*, and *gsi_return_timed_attrs()* accept *gsi_item* in their first argument. The *gsi_item* must represent an item that G2 passed to the bridge through a remote procedure call declared in G2 with the with handle grammar.

Thus, these API functions can be invoked with the following arguments:

```
void gsi_return_values(gsi_items, count, context_number)
void gsi_return_attrs(registered_item, attributes, count,
    context_number)
void gsi_return_timed_values(registered_items, count,
    context_number)
void gsi_return_timed_attrs(registered_item, attributes, count,
    context_number)
```

where *gsi_item* or *gsi_items* refers to items passed to the bridge by G2 through remote procedure calls declared using the with handle grammar.

Note These API functions can still be invoked with a *gsi_registered_item* as their first argument, as in previous releases.

Attributes Passed with Items

The functions *gsi_return_values()*, *gsi_return_attrs()*, *gsi_return_timed_values()*, and *gsi_return_timed_attrs()* can return the following attributes of an item (represented by a *gsi_item* structure) to G2:

- User-defined attributes, including attributes that are class-qualified.

  To exclude certain user-defined attributes, use *gsi_set_rpc_remote_return_exclude_user_attrs*.

- System-defined attributes exported to the bridge by G2.

  To specify the system-defined attributes to include, use *gsi_set_rpc_remote_return_include_system_attrs*. To specify the system-defined attributes to exclude, use *gsi_set_rpc_remote_return_include_all_system_attrs_except*.

- A *gsi_attr* marked with an index value that specifies the location in a G2 item-array into which the item in the *gsi_attr* is to be placed.

  You can set the index value of a particular element of the attribute array by calling *gsi_set_attr_array_index()*. Array indexes start at 0. For example, a *gsi_attr* within a *gsi_item* of class float-array whose index is *3* refers to the fourth element of that array in G2.

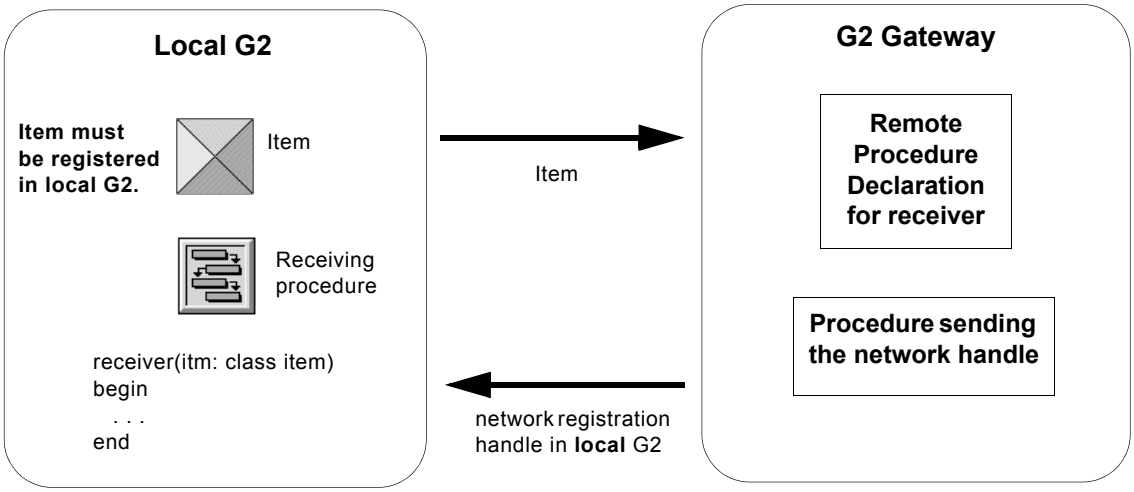
Passing Network Handles as the Class in RPCs

You can pass the network handle of an item as an argument to an RPC in a G2 Gateway bridge, where the receiving procedure in the local G2 expects an item, and G2 attempts to replace the handle with the item before calling the procedure. If G2 does not find an item with that network handle, or if the handle is not of the class the procedure is expecting, it signals a type-mismatch error to the caller.

Note To call a G2 procedure with a network handle in order to rendezvous with an item, the procedure must declare its argument type to be a class of item; the procedure cannot declare it to be an item-or-value.

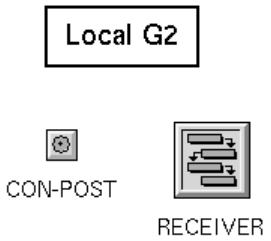
Note To use this feature, you must register the item in the local G2 and pass the local network handle as the argument to the RPC. For example, you might register a number of items in the local G2 and pass a list of network handles to the G2 Gateway bridge, which can then be used to call a procedure remotely in the originating G2 where item rendezvous can now occur.

This figure illustrates how item rendezvous occurs when passing network handles:



Example of Passing Handles as the Class

For example, in the local G2, suppose you have an item named con-post that you want to pass as the argument to a procedure named receiver, which you are calling from a G2 Gateway bridge:



Here is the receiver procedure in the local G2, which takes an item as its argument and simply posts the name of the item to the Message Board. Notice that the procedure argument is declared to be a class of item, not item-or-value.

```
receiver(i: class item)
    begin
    post "received an item named [the name of i]";
end
```

In the G2 Gateway bridge, you would define a GSI interface and declare the remote procedure. Note that the remote procedure declaration takes as its

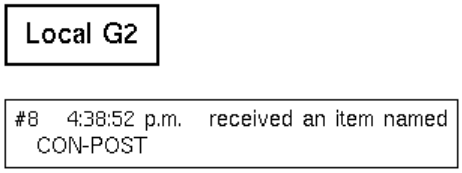
argument a value, which is the argument type that is being passed to the RPC in the remote G2, namely, a network handle.

In the G2 Gateway bridge, you can make a remote procedure call to receiver, passing the network handle as the argument, in this case, the integer 1. Note that this integer is the network handle of the item registration in the local G2, which you must generate locally and pass to the G2 Gateway bridge.

Here is the send-handle procedure, which makes the remote procedure call, passing the network handle as the argument, instead of the item:

```
send-handle(handle: integer)
    begin
    call receiver(handle) across interface;
end
```

Calling receiver across the GSI interface replaces the network handle with the item, which posts the name of the item in the Message Board:

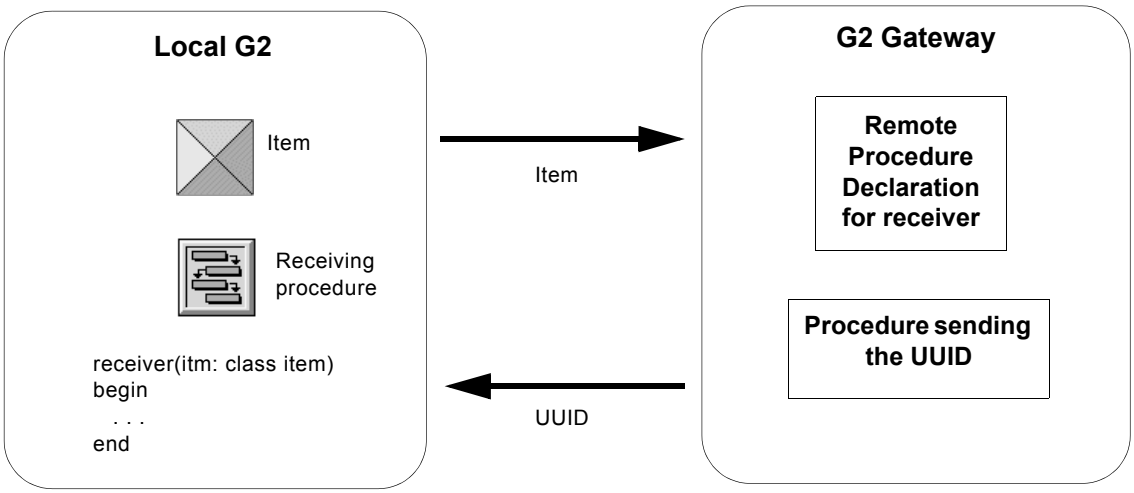


Passing UUIDs Referring to Items in RPCs

You can pass the text of the UUID of an item as an argument to an RPC in a G2 Bridge, where the receiving procedure in the local G2 expects an item, and G2 attempts to replace the UUID with the item before calling the procedure. If G2 does not find an item with that UUID, or if the UUID is not of the class the procedure is expecting, it signals a type-mismatch error to the caller.

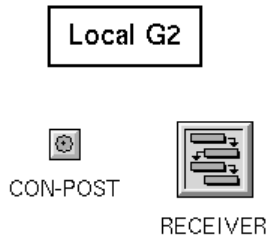
Note To call a G2 procedure with a UUID in order to rendezvous with an item, the procedure must declare its argument type to be a class of item; the procedure cannot declare it to be an item-or-value.

This figure illustrates how item rendezvous occurs when passing UUIDs:



Example of Passing UUIDs Referring to Items

For example, in the local G2, suppose you have an item named con-post that you want to pass as the argument to a procedure named receiver, which you are calling from a G2 Gateway bridge:



Here is the receiver procedure in the local G2, which takes an item as its argument and simply posts the name of the item to the Message Board. Notice that the procedure argument is declared to be a class of item, not item-or-value.

```
receiver(i: class item)
    begin
    post "received an item named [the name of i]";
end
```

In the G2 Gateway bridge, you can make a remote procedure call to receiver, passing the text of the UUID as the argument. Note that the remote procedure

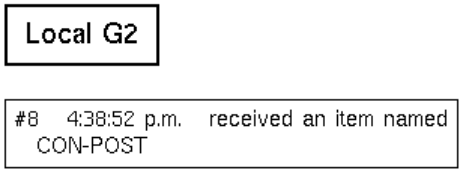
declaration takes as its argument a value, which is the argument type that is being passed to the RPC in the G2 Gateway bridge, namely, a UUID.

Note You can also pass the UUID in compressed format; however, note that you cannot see the value of the UUID in compressed format like you can the text format.

Here is the send-uuid procedure, which makes the remote procedure call, passing the UUID as the argument, instead of the item:

```
send-uuid(uuid: text)
    begin
    call receiver(uuid) across interface;
end
```

Calling receiver across the GSI interface replaces the UUID with the item, which posts the name of the item in the Message Board in the local G2:



Developing a Bridge Using Only Remote Procedure Calls

To build your G2 Gateway application, relying on only RPCs to handle events or transfer data between G2 and the bridge, the minimum that you must include in your application is as follows:

- In G2:
  - Create and configure one GSI interface for each connection that G2 needs to establish to the bridge.
  - Declare all of your G2 Gateway local functions (functions that G2 will invoke as remote) inside the callback function *gsi_set_up()*.
  - Declare the G2 Gateway local functions as remote procedures that can be invoked by G2.
- In G2 Gateway:
  - Include all of the G2 Gateway callback functions in your G2 Gateway user code. If you do not modify any of the callback functions for your own use, you must still include them in their *stub* form (unmodified, except to return *GSI_ACCEPT* or *GSI_REJECT*).

- Declare all of your remote procedures (G2 procedures invoked by the bridge) in `gsi_initialize_context()`.
- Declare a global variable of the type `functon_handle_type` to identify each G2 procedure that G2 Gateway invokes as a remote procedure.

Call Identifiers and Procedure User Data

G2 Gateway enables remote procedure calls between G2 and G2 Gateway to include:

- **Procedure user data** arguments, which contain values that identify particular remote procedure declarations in the G2 Gateway user code.
- **Call identifier** arguments, which contain values that distinguish individual remote procedure calls from each other.

To enable the use of these arguments, you must compile your G2 Gateway code with the `gsi_use_user_data_for_callbacks` C preprocessor flag defined or use the corresponding compile time switch. For information about the G2 Gateway preprocessor flags and runtime options, see Preprocessor Flags and Runtime Options.

Procedure User Data for Remote Procedure Calls

You can associate user data with calls to particular G2 procedures by including a *procedure_user_data* argument in the calls. The procedure user data is returned to G2 Gateway in the first arguments of G2 Gateway local functions, receiver functions and error receiver callback functions. The *procedure_user_data* argument of a remote procedure call can be data of any type.

You can include a user data argument in local functions and receiver functions only if the compile time switch `gsi_user_data_for_callbacks` is set. This switch is set automatically if you compile your G2 Gateway application with the `gsi_use_user_data_for_callbacks` C preprocessor flag defined or you use the corresponding compile time switch. For information about these flags and options, see Preprocessor Flags and Runtime Options.

Declare a *procedure_user_data* argument as follows:

```
gsi_procedure_user_data_type procedure_user_data
```

Procedure User Data in Receiver Functions and Error Receiver Functions

As the first argument of receiver functions and error receiver functions, *procedure-user-data* represents a value that was specified by the GSI programmer and associated with a particular remote procedure declaration, using `gsi_rpc_declare_remote_with_error_handler_and_user_data()`.

Every call that the G2 Gateway bridge makes to a G2 procedure using this declaration has the specified procedure user data value associated with it; the procedure user data is, in effect, a label identifying a particular remote procedure declaration. G2 never examines or uses the procedure user data, but associates it with any return call that it makes to the receiver function or error receiver function in the G2 Gateway bridge. The procedure user data then enables the receiver function or error receiver function to know which G2 Gateway remote procedure declaration was used to invoke the G2 procedure that is now sending a response to the receiver function or error receiver function.

Procedure User Data in Local Functions

As the first argument of local functions, procedure user data represents a value that was specified by the G2 Gateway programmer and associated with a particular local procedure declaration, using `gsi_rpc_declare_local()`.

When the local function is invoked by G2, it receives the procedure user data value specified in the corresponding local procedure declaration. Each local procedure declaration specifies a different *g2-function-name*, which must be identical to the name-in-remote-system attribute of the G2 remote procedure declaration through which G2 is invoking the local function.

Using Procedure User Data

For some purposes, you can use `gsi_procedure_user_data_type` arguments to eliminate the need for separate local functions or receiver functions. For example, you can include several calls to `gsi_rpc_declare_local()` in your user code, with each call specifying the same user-written G2 Gateway local function, but with different G2 names and `gsi_procedure_user_data_type` arguments. Thus, the single local function in your bridge appears to G2 as several different remote procedures, each with a different name. When G2 calls the function under one of its names, it passes a value of whatever type to the `gsi_procedure_user_data_type` argument local function. Your G2 Gateway user code can customize its behavior based on the `gsi_procedure_user_data_type` value that it receives, in a `case` statement or by other means.

Similarly, a single receiver function with a `gsi_procedure_user_data_type` argument can handle return data from a variety of different G2 procedures invoked from the bridge. Without the use of a `gsi_procedure_user_data_type` arguments, several different receiver functions might be required to handle different types of data returned by different G2 procedures.

Declaring a Local Function with a Procedure User Data Argument

When you declare a G2 Gateway local function, you specify the procedure user data argument in the call to `gsi_rpc_declare_local()`:

```
void gsi_rpc_declare_local
    (local_function, procedure_user_data, g2_function_name)
```

Any G2 procedure that invokes the local function sends the procedure user data associated with the invocation to the first argument (the *procedure_user_data* argument) of the local function.

Declaring a G2 Procedure as a Remote Function with a Procedure User Data Argument

When you declare a G2 procedure as a remote procedure that can be invoked by the G2 Gateway bridge, you specify the *procedure_user_data* argument in the call to `gsi_rpc_declare_remote()`:

```
void gsi_rpc_declare_remote
    (function_handle, g2_function_name, receiver_function,
    procedure_user_data, argument_count, return_count,
    context_number)
```

A G2 procedure called by the G2 Gateway bridge through `gsi_rpc_call()` or `gsi_rpc_call_with_count()` can return data of any type to the first argument (the *procedure_user_data* argument) of the receiver function in the G2 Gateway bridge.

Writing Receiver Functions and Error Receiver Callbacks with Procedure User Data Arguments

You can include a procedure user data argument in G2 Gateway receiver functions and error receiver callback functions. For information about the argument syntax of these functions, see RPC Support Callback Functions.

Call Identifiers for Remote Procedure Calls

G2 Gateway supports the use of call identifiers both in calls from G2 Gateway to G2, and in calls from G2 to G2 Gateway.

If a G2 Gateway bridge invokes the same G2 procedure repeatedly, there is no guarantee that the G2 procedure will return the results of the separate invocations to the bridge in the order in which the bridge made them. Thus, the bridge needs a way to distinguish separate calls from each other. Similarly, G2 may make repeated calls to local function in a G2 Gateway bridge, and need to distinguish these calls from each other.

To make it possible for G2 Gateway to distinguish the values returned to the bridge by different invocations of a remote G2 procedure, `gsi_rpc_call()` and `gsi_rpc_call_with_count()` can include a call identifier argument that you supply. For example:

```
void gsi_rpc_call(function_handle, arguments,
    call_identifier, context_number)
```

The G2 procedure can send the original *call_identifier* value specified in the call to `gsi_rpc_call()` or `gsi_rpc_call_with_count()` to a receiver function in the G2 Gateway bridge.The required syntax for a receiver function is:

```
void receiver_function (procedure_user_data,arguments,count,
    call_identifier)
```

where *call_identifier* is a `gsi_call_identifier_type` value returned from G2 that is identical to the value returned by the call to `gsi_rpc_call()` or `gsi_rpc_call_with_count()` that invoked the G2 procedure.

In calls that G2 makes to a G2 Gateway local function, the call identifier is a value that G2 generates automatically to distinguish each call from other calls that it may make to the same local function. G2 passes this value to the *call_identifier* argument of the local function. The local function should return this value to G2 in calls that it makes to `gsi_rpc_return_error_values()` or `gsi_rpc_return_values()`, but it can also use this value to distinguish between simultaneous calls to the same local function.

5

Error Handling

Describes how G2 Gateway handles errors by default, and how you can customize error handling in your G2 Gateway bridge.

Introduction

G2 Gateway provides a default error handler that responds to a variety of system-defined errors automatically, as they occur. It also enables you to signal user-defined errors from your user code, and to create a customized error handling procedure to perform any specialized processing of errors that your application requires.

Default Error Handling

When an API function or callback detects an error, the default error handler:

- Sends information about the error to the G2 Gateway application's standard output device (*stdout*).

  Your G2 Gateway application can suppress the output of error information. For information about how to do this, see Sending Error Information to Standard Output.

- If the error is fatal, shuts down the context (if known) in which the error occurred.

  If the error is non-fatal, the default error handler allows the context to continue running.

  For a complete list of standard error conditions, see Appendix C, G2 Gateway Error Messages.

- If you have not installed a customized error handler, returns control to the point in your user code that called the function that detected the error.

- If you have installed a customized error handler procedure, calls this customized procedure.

See the table in the section called Error Handling in Continuous and One-Cycle Modes for details.

Sending Error Information to Standard Output

When G2 Gateway detects an error condition in a G2 Gateway internal operation, callback function, or API function, it prints information about the error on the G2 Gateway application's standard output (*stdout*) device. This information includes the error number, the message associated with the error, and the name of the function that produced the error.

For a list of the standard errors signalled by G2 Gateway, see Appendix C, G2 Gateway Error Messages.

You can suppress the output of error messages to standard output by setting the *GSI_SUPPRESS_OUTPUT* option. To do this, include a call to the following API function in your *gsi_set_up()* callback function:

```
gsi_set_option(GSI_SUPPRESS_OUTPUT);
```

Shutting Down the Context Where the Error Occurred

After printing the error information about a fatal error, G2 Gateway unconditionally shuts down the context in which the error occurred and passes control back to `gsi_run_loop()`.

If an error occurs in an API function called from outside the `gsi_run_loop()` call tree, any context that the API function operated on is shut down. Some API functions accept a context number argument to specify the particular context on which they operate. If the API function does not specify a particular context, it is of the sort that does not require that the context be shut down when an error occurs.

Customized Error Handling

G2 Gateway enables you to extend the default error-handling of your G2 Gateway bridge process by:

- Signalling non-standard user errors from your user code, using the API function `gsi_signal_error()`. The default error handler handles user-defined errors in the same way that it handles standard errors.
- Creating a customized error handler procedure to respond to errors in specialized ways required by your G2 Gateway bridge process. The customized error handler responds to both standard and user-defined errors.

The customized error handler is called by the default error handler, and works in conjunction with, not in place of, the default error handler.

The default error handler calls the customized error handler after the data is sent to standard output and, in the case of a fatal error, after the current context has been shut down.

Signalling Customized Error Conditions

Your user code can signal its own errors and make use of the internal G2 Gateway error handling system by calling `gsi_signal_error()`. You can use values above 1024 for your user code errors.

When `gsi_signal_error()` signals a customized error, the G2 Gateway error handler responds to it in the same way that it responds to errors generated by G2 Gateway internals.

For information about `gsi_signal_error()`, see API Functions.

Note The API functions *gsi_signal_handler()* handles C and UNIX signals, and not signals raised by your user code by calling *gsi_signal_error()*. The term *signal* has a different sense in the two cases.

Writing a Customized Error Handler

When an API function reports an error, the default error handler automatically invokes your user error handler procedure, if you have installed one.

As arguments to the customized error handler, the default error handler passes the number of the context where the error occurred, the error number, and the error message that G2 Gateway associates with the error.

Your error handler procedure must use the following syntax:

```
void *handler_function(error_context, error_code, error_message)
   gsi_int error_context;
   gsi_int error_code;
   gsi_char *error_message;
```

where:

error_context is the context number of the context in which the error occurred.

error_code is an integer that identifies the error condition.

error_message is the text of the G2 Gateway message associated with the error number.

Installing a Customized Error Handler

You must install your customized error handler procedure by calling the API function *gsi_install_error_handler()*.

Because you install an error handler only once, it is good practice to invoke *gsi_install_error_handler()* in the *gsi_set_up()* callback function.

The error-handling procedure that you install is called automatically by G2 Gateway's internal error handler when an error occurs.

Checking the Global Error Flag

A global error flag is set by each API function when it finishes executing. If an error occurs during the execution of an API function, G2 Gateway sets the global error flag and saves the message associated with the error. The value to which the API function sets the flag specifies the particular error that occurred during execution of the function, or specifies that no error occurred.

To read the global error flag, your error handler function can call *gsi_last_error_message()*. This function returns the value of the error flag, which is a number that identifies the error. Use *gsi_last_error_message()* rather than *gsi_error_message()*, which can return formatting templates rather than the simple text of the error message.

Your user code can pass the error number returned by *gsi_last_error()* to the API function *gsi_error_message()*, which returns a string containing the message associated with a specified error code.

You can clear the global error flag by calling the API function *gsi_clear_last_error()*, which sets the last error number to zero.

Error Handling in Continuous and One-Cycle Modes

G2 Gateway performs both default and customized error handling differently in one-cycle and continuous modes.

The following table summarizes the default and customized error-handling behavior of G2 Gateway in continuous and one-cycle modes:

Error Handling in Continuous and One-Cycle Modes

	Default Error Handling	Customized Error Handling
Continuous Mode	**1** Send error information to *stdout*. **2** Shut down context where the error occurred. Not all errors shut down the context. For a list of the errors that do, see Errors that Shut Down a Context. **3** Pass control to bottom of *gsi_run_loop()*, which continues to iterate over any contexts that remain open.	**1** Send error information to *stdout*. **2** Shut down the context where the error occurred. Not all errors shut down the context. For a list of the errors that do, see Errors that Shut Down a Context. **3** Invoke customized error handler. **4** Pass control to bottom of *gsi_run_loop()*, which continues to iterate over any contexts that remain open.
One-Cycle Mode	**1** Send error information to *stdout*. **2** Shut down context where error occurred. Not all errors shut down the context. For a list of the errors that do, see Errors that Shut Down a Context. **3** If the error is detected within *gsi_run_loop()*, return control to code that called *gsi_run_loop()*. If the error is detected by an API function outside *gsi_run_loop()*, return control to the code that invoked this API function.	**1** Send error information to *stdout*. **2** Shut down the context where the error occurred. Not all errors shut down the context. For a list of the errors that do, see Errors that Shut Down a Context. **3** Invoke customized error handler. **4** If the error is detected within *gsi_run_loop()*, return control to code that called *gsi_run_loop()*. If the error is detected by an API function outside *gsi_run_loop()*, return control to code that called this API function.

Errors that Shut Down a Context

The following table lists errors that always shut down the context in which they occur:

Errors that Always Shut Down a Context

Value	Error Symbol	Text of Error
38	*GSI_CONNECTION_LOST*	"*Network layer reports connection was lost or ICP protocol error occurred: error-message*"
57	*GSI_ICP_MESSAGE_TOO_LONG*	"*ICP message series too long -- please call Gensym customer support*"
58	*GSI_ICP_MESSAGE_OUT_OF_SYNCH_CASE_2*	"*Protocol out-of-synch (case 2)*"
59	*GSI_MAXIMUM_TEXT_STRING_LENGTH_ERROR*	"*Attempting to allocate number-of-elements element string, which is beyond the established maximum of maximum-elements.*"
60	*GSI_EXTEND_CURRENT_TEXT_STRING_ERROR*	"*Trying to write a string longer than 1MB!*"
63	*GSI_CIRCULARITY_NOT_SUPPORTED*	"*Self referencing items may not yet be sent to G2 - sorry*" **Note**: This error occurs only when G2 Gateway 5.0 is communicating with a G2 4.0.
67	*GSI_UNKNOWN_TYPE_TAG*	"*GSI structure contains unknown type tag.*"
71	*GSI_MAXIMUM_CONTEXTS_EXCEEDED*	"*Connection rejected - GSI bridge context limit maximum-contexts exceeded.*"
72	*GSI_CONNECTION_DENIED*	"*Connection denied - the G2 at protocol-host-port has disallowed connections from GSI*"

Errors that Always Shut Down a Context

Value	Error Symbol	Text of Error
74	*GSI_ERROR_IN_CONNECT*	"*Error during connection attempt: error-message*"
79	*GSI_UNKNOWN_CALLING_PROCEDURE_INDEX*	"*Unknown calling procedure index.*"

The following table lists errors that can shut down a context, depending on the circumstances that gave rise to the error.

Errors that Shut Down a Context in Some Circumstances

Value	Error Symbol	Text of Error
16	*GSI_INCOMPATIBLE_TYPE*	"*Type mismatch - value of type data-type passed to this function*"
30	*GSI_INCOMPATIBLE_STRUCTURE*	"*Received null pointer argument, or a structure type incompatible with requested operation*"
68	*GSI_INVALID_HISTORY_TYPE*	"*GSI found an invalid value type for this history.*"

6

Troubleshooting Guidelines

Describes how to identify problems in your G2 Gateway bridge user code.

Introduction

This chapter provides troubleshooting suggestions for your G2 Gateway application, and concludes with a procedure for submitting any problems you may have with your code, the G2 Gateway bridge, or G2 to Gensym's Customer Support personnel.

This chapter lists problems for the following general topics:

- Connectivity
- Data collection and transmission
- Initializing
- Object definition
- Remote Procedure Calls (RPCs)

Connectivity

This section covers problems with the interface between the G2 and the G2 Gateway processes, the Intelligent Communications Protocol (ICP), or network connections.

Problem: You have created a GSI interface, started the bridge process, and started the G2 process, but nothing happens.

Solutions:

- The workspace on which the GSI interface is located may be inactive (disabled). Make sure that the workspace is enabled.
- The GSI interface is not named. An unnamed GSI interface cannot support a connection between a G2 Gateway bridge process and a G2 knowledge base. Specify one or more names for the GSI interface in its names attributes.
- The GSI interface has a different specification for its gsi-connection-configuration attribute than the one it should have to locate the bridge process you want to use. For example, you may have specified an incorrect port number or machine name. Check the value specified for this attribute and change it if necessary.
- If the bridge process was started after the G2 process, the bridge may not be able to connect to the G2 process without your disabling, then re-enabling the GSI interface used to connect to the bridge. The GSI interface only checks whether the bridge process is running when the GSI interface becomes enabled, which is usually at the same time that you start the G2 process. By manually toggling the GSI interface, you allow the GSI interface to determine whether the bridge process is running, and, if so, connect to it.

Problem: The operator's logbook displays the error message "*Cannot establish ICP connection.*"

Solutions:

- Is there an ICP connection between the machines on which the G2 and the G2 Gateway processes are running? Send out a `ping` to the machine on which the bridge process is running. If it does not respond or does not respond before the interface times out, there may be network problems that are interfering with communication.
- Make sure that you have specified the correct location of the bridge process for the gsi-connection-configuration attribute of the GSI interface. If you edit the attribute while the processes are still running, toggle the GSI interface so that it will check to see if the bridge process is running and, if so, connect to it.

Problem: The bridge process could not establish a listener at the TCP/IP port and exits, or it establishes a listener but doesn't connect.

Solutions:

- The port number or object name is unavailable. Try using a different port or object name.
- The two arguments *argv* and *argc* are not passed correctly from *main()* to *gsi_start()*. You must make sure that, however you define the two arguments in *main()*, when you pass the arguments on to *gsi_start()*, the arguments are exactly what you expect *gsi_start()* to receive.
- If the bridge process was started after the G2 process, the bridge may not be able to connect to the G2 process without your disabling, then re-enabling the GSI interface used to connect to the bridge. The GSI interface only checks whether the bridge process is running when the GSI interface becomes enabled, which is usually at the same time that you start the G2 process. By manually disabling and re-enabling the GSI interface, you allow the GSI interface to determine whether the bridge process is running, and, if so, to connect to it.
- On some platforms the operating system takes a while to release a socket that was retained by a recently killed process. Either try a different port number or wait a while. The duration is platform-dependent and ranges from a few seconds to a few minutes.
- Ask your network administrator to check the physical connection between the machines on which G2 and G2 Gateway are running.

Problem: The GSI interface times out.

Solutions:

- You may be running in an environment with heavy network traffic. Increase the interface-timeout-period attribute of the GSI interface until the interface no longer times out.
- G2 Gateway may be attempting to define at one time a large number of objects for which you have specified default update intervals. You can arrange to stagger definition requests over time by placing groups of the objects you want to be defined onto different subworkspaces, then enabling the subworkspaces one at a time, pausing in between each enable operation until all objects on that workspace are defined.

Problem: The GSI interface's status code changes to -1 or -2.

Solutions:

- The -1 status code indicates that the ICP connection has timed out and possibly may be gone. For this case, there may be no need for your code to take action in response, because the status code will change to a 2 (`OK`) or -2 (failed) after several seconds or so have elapsed. If it does not, you can toggle the GSI interface by disabling, then re-enabling it.
- The -2 status code indicates that the ICP connection is gone and cannot be recovered. You may be able to establish a new connection without restarting the bridge. Try to restart the bridge process, followed by restarting the G2 process. If the problem persists, report it to Gensym's Customer Support staff.

Problem: The *ICP out of synch* error message appears.

Solutions:

- You may be using an outdated `gsi_main.h` file, or linking against an old `gsi_main` object. Gensym recommends that you use the `gsi_main.h` file that was included with the G2 Gateway distribution kit.
- Memory may be corrupted due to a user programming error.
- Too many arguments are specified when you are passing arguments in a remote procedure call (RPC). Check that you are passing the correct number of RPC arguments.

Data Collection and Transmission

This section covers problems with the collection and transmission of data from the bridge to the knowledge base, or with G2 setting values in the external system.

Problem: The GSI interface is okay but the G2 process is very slow (or eventually exits).

Solution: You may be inundating G2 Gateway with more requests for values than it can return to G2 in a cycle. G2 may slow down as more and more requests wait to be processed. Stagger the update intervals for your variables or decrease the frequency with which G2 sets values in the external system.

Problem: The bridge cannot return attribute values to G2.

Solution: The names of attributes whose values are returned from the external system through a vector must be specified in your user code as symbols. (Refer to your *G2 Reference Manual* for the correct symbol format.) Frequently, this simply means using upper case letters for attribute names in your bridge code instead of lower case letters.

Problem: A GSI variable is not receiving any values.

Solutions:

- The item may not be registered. Call the system procedure g2-get-network-handle-from-item() to determine if the item is registered. For information about this procedure, see the *G2 System Procedures Reference Manual*.
- The gsi-interface-name specified for the GSI variable is incorrect or missing. Edit the GSI variable's gsi-interface-name attribute, specifying the unique name for the GSI interface used by the variable.
- You may be passing back the *NULL_TAG* data type to the variable. Make sure that you return the value as *NULL_TAG* only if you do not want to update the value of the variable.
- Set a higher priority for the priority-of-data-service attribute in the Data Server Parameters system table, or reduce the processing load of your G2 application.

  If you have specified a low priority for the priority-of-data-service attribute in the Data Server Parameters system table, and your G2 application has a heavy processing load, *gsi_g2_poll()* may never be called. This results in the GSI variables not receiving values if your G2 Gateway application uses *gsi_g2_poll()* to get values from an external system and return them to the GSI variables in G2.

Problem: The bridge is returning the wrong value to a GSI variable.

Solutions:

- Make sure that the data type corresponds to the one expected for the variable. Make sure that your code performs any necessary type conversions prior to transmission of the value.
- Signed integer values that occupy more than 29 bits, or unsigned integer values that occupy more than 30 bits are not accepted by G2. Limit the size of your integer values to no more than 29 bits for a signed integer, or 30 bits for an unsigned integer. Alternatively, you can cast the integer value as a float, or specify the variable as a quantitative variable. Quantitative variables automatically convert integers to floating point numbers if they are too large for G2 to accept as integers.

Problem: An error message appears on the Operator's Logbook stating that a data type mismatch has occurred.

Solution: The value returned to a GSI variable is of the wrong data type. Make sure that the data type corresponds to the one expected for the variable. Make sure that your code performs any necessary type conversions prior to transmission of the value.

Problem: The variable is not updated in accordance with the default update interval for a GSI variable.

Solutions:

- You have not placed the proper code in *gsi_get_data()* to retrieve the requested value and return it to G2. (This assumes that external-system-has-a-scheduler is No.) Change *gsi_get_data()* to include code to return the value to G2.
- The external-system-has-a-scheduler attribute for the GSI interface used by the variable is specified as yes, but you have not yet implemented the external system's scheduler. Either edit the attribute to turn off this feature, or implement the scheduler.

Problem: A value set by G2 in the external system is not echoing back to the corresponding GSI variable in the knowledge base.

Solution: Values set by G2 in the external system are not echoed back to the corresponding variables automatically. You must include a call to *gsi_return_values()* in your user code for each set value that you want returned to update a variable in your knowledge base.

Problem: Unsolicited data is not returned to GSI variables.

Solution: To make sure that unsolicited data is returned to G2, the value of both the external-system-has-a-scheduler attribute and the poll-external-system-for-data attribute of the GSI interface must be yes.

Problem: A rule of the form, "whenever X fails to receive a value", is not evaluated when a bridge does not return a value to a variable.

Solution: The reason for this problem is that the *timeout-for-inference-completion* of *Inference-Engine-Parameters* does not apply to variables whose data server attribute is *gsi-data-service*. A bridge may use as much time as it needs to return a value. There is no timeout for data acquisition from a bridge.

To determine if a variable is no longer receiving values, you can use a combination of the variable's *validity-interval*, *default-update-interval*, and a rule of the form "whenever x loses its value".

Item Registration

This section covers problems that apply to the registration of items for the G2 and the bridge processes at startup time, or the redefinition of variables when they are changed.

Problem: The function *gsi_receive_registration()* is not called to define objects.

Solutions:

- You have not specified default update intervals for your GSI variables, or your G2 process is not calling *gsi_set_data()* or *gsi_get_data()* by setting values in the external system or requesting values explicitly. If the update interval is set to none, you must either change the update interval to a positive length of time, or make explicit data requests.
- The GSI interface name specified for the variables is wrong or missing. Change to the unique name of the GSI interface, or supply it if it is missing.

Problem: A GSI variable is registered more than once.

Solution: A GSI variable is registered every time to change the list of its identifying attributes. If you use a procedure in G2 to fill in the identifying attributes for a GSI interface, make sure that the GSI interface is disabled, undefined, or differently named until you have finished listing the identifying attributes. If the GSI interface is enabled when the procedure fills in the values for the attributes, the GSI variable is registered each time an identifying attribute is added to the list, resulting in a call to *gsi_receive_registration()* each time.

Remote Procedure Calls (G2-to-G2 Gateway)

This section covers problems that apply to RPCs made to the bridge from the G2 process.

Problem: An RPC is not being invoked or is not being called.

Solutions:

- You may have specified the wrong number of arguments. Check and if necessary adjust the number of arguments.
- The *p_type* value for a parameter of the remote function does not match the type specified in the G2 RPC declaration. Make sure that the *p_type* specification matches the type given in the RPC declaration.
- RPC is started instead of being called. Use a call statement in a procedure on a workspace, or in place of a start action.

Reporting Problems to Gensym

Use the procedure explained in this section for reporting any problems you have with your G2 Gateway bridge or G2.

To report a problem to Gensym:

1. Identify the problem and/or its symptoms.
2. Try to recreate the problem.
3. Assemble and be prepared to provide the following information:
   - Your name, your company's name, and (optionally) your location within the company.
   - Your telephone number and extension, your facsimile number, and the best times for Gensym personnel to contact you using either number.
   - The version numbers of both G2 and G2 Gateway.
   - The name and version number of the G2 Gateway bridge product you are using (if any).
   - A problem description that includes its symptoms, a detailed synopsis of what you were doing when the problem occurred, and the severity of the problem. Include the exact text of any G2 Gateway internal errors and G2 log book errors associated with the problem.
4. Report your problem to Gensym's Customer Support staff.

Provide all of the information you gathered in step 3 of this procedure to the customer support representative. You may want to include a section of the relevant code.

For contact information, see Customer Support Services.

Part II

Reference

Chapter 7: G2 Gateway Data Structures

Describes how G2 Gateway data structures store information that is useful to your application, and how your G2 Gateway user code can access this information.

Chapter 8: Callback Functions

Describes the callback functions that you complete to implement your G2 Gateway user code.

Chapter 9: API Functions

Describes the capabilities and syntax of the API functions supported by G2 Gateway.

Chapter 10: Preprocessor Flags and Runtime Options

Describes C preprocessor macros and runtime options that you can use to modify the behavior of your G2 Gateway bridge.

Chapter 11: Building and Running a G2 Gateway Bridge

Describes how to compile, link, and run a G2 Gateway bridge executable image, and how to start and stop a G2 Gateway bridge process from within a G2 procedure.

7

G2 Gateway Data Structures

Describes how G2 Gateway data structures store information that is useful to your application, and how your G2 Gateway user code can access this information.

Introduction

G2 Gateway creates data structures to store information that it receives from G2. Your G2 Gateway user code can allocate the same kinds of data structures to store information that the G2 Gateway bridge receives from an external system.

Each structure includes different **components** to receive different kinds of information, such as item handles, attribute values, or history information. Some data structures also have components that contain other data structures.

Your user code can access the components of data structures by calling API functions provided with G2 Gateway. Each API function is designed to access a particular component of one or more data structures.

For fewer upgrade problems and less implementation-dependent code, G2 Gateway does not permit direct access to G2 Gateway data structures. Your user code can access the data structures only by invoking the API functions provided for this purpose.

Note The text of messages that G2 Gateway receives from a GSI Message Server is not maintained in a data structure, nor is it available after the callback *gsi_receive_message()* exits.

Summary of G2 Gateway Data Structures

G2 Gateway provides *void** pointers to its internal data structures:

- *gsi_registration*
- *gsi_registered_item*
- *gsi_item*
- *gsi_attr*
- *gsi_symbol*

The following table lists the G2 Gateway internal data structures referenced by these pointers and describes how G2 Gateway uses each structure:

G2 Gateway Data Structures

G2 Gateway Data Structure	Description
gsi_registration	Created by G2 Gateway when G2 registers a data-served GSI variable or an item handle that it passes through a remote procedure call. Remains in existence until the variable or item handle is deregistered. Stores the handle, data type, and six identifying attributes of the variable being registered. Can also store user data that you choose to associate with this registered item.
gsi_registered_item	Created by G2 Gateway each time G2 asks G2 Gateway to get a new value for a GSI variable from an external system, or when G2 asks G2 Gateway to write the value of a GSI variable to a data point in an external system. Remains in existence only during the read or write operation. Stores the handle, current attribute values, and other information associated with the registered item. You can create this data structure in your bridge code by using API functions.
gsi_item	Created by G2 Gateway each time G2 requests G2 Gateway to read from or write to a data-served GSI variable, or passes an item handle or object to G2 Gateway through a remote procedure call. Stores information associated with the variable or object, such as its value. You can create this data structure in your bridge code by using API functions.

G2 Gateway Data Structures

G2 Gateway Data Structure	Description
gsi_attr	Created by G2 Gateway to represent an attribute of an object that G2 Gateway receives from G2. Stores information associated with the attribute. You can create this data structure in your bridge code by using API functions.
gsi_symbol	Created by G2 Gateway to represent a symbol that G2 Gateway receives from G2. Stores information associated with the symbol. You can create this data structure in your bridge code by using API functions.

Using Get and Set Functions for Data Structures

Application-specific information can be stored in GSI data structures. The information includes the following user data: registrations and items, contexts, symbols, local and remote procedures, and remote calls.

The following table correlates types of data structures, data types, and set and get functions:

Type of Structure	C Data Type	Get Function	Set Function
Registrations and items	*gsi_item_user_data_type*	*gsi_user_data_type*	*gsi_set_user_data*
Contexts	*gsi_context_user_data_type*	*gsi_context_user_data*	*gsi_set_context_user_data* *gsi_initiate_connection_with_user_data*
Symbols	*gsi_symbol_user_data_type*	*gsi_symbol_user_data*	*gsi_set_symbol_user_data*
Local procedures	*gsi_procedure_user_data_type*	*local functions*	*gsi_rpc_declare_local*
Remote procedures	*gsi_procedure_user_data_type*	*receiver functions*	*gsi_rpc_declare_local* *gsi_rpc_declare_remote_with_error_handler_and_user_data*
Local calls	*gsi_call_identifier_type*	*local functions*	*none*
Remote calls	*gsi_call_identifier_type*	*receiver function*	*gsi_rpc_call* *gsi_rpc_call_with_count*

Data structures are created by G2 Gateway for a callback function and can be accessed by subsequent callbacks without the use of a lookup table. During a

callback function, G2 Gateway supplies registration so that a function can store user data in a data structure.

Referencing Data Structures in Your User Code

Your user code can use *gsi_registration*, *gsi_registered_item*, *gsi_item*, *gsi_attr*, and *gsi_symbol* to declare instances of the G2 Gateway internal data structures.

For example, the callback function

```
void gsi_receive_registration(item_registration)
```

receives one argument, an instance of a *gsi_registration* structure, which contains information about a registered item. You declare this argument as follows:

```
gsi_registration item_registration;
```

where:

item_registration is declared to be an instance of *gsi_registration*.

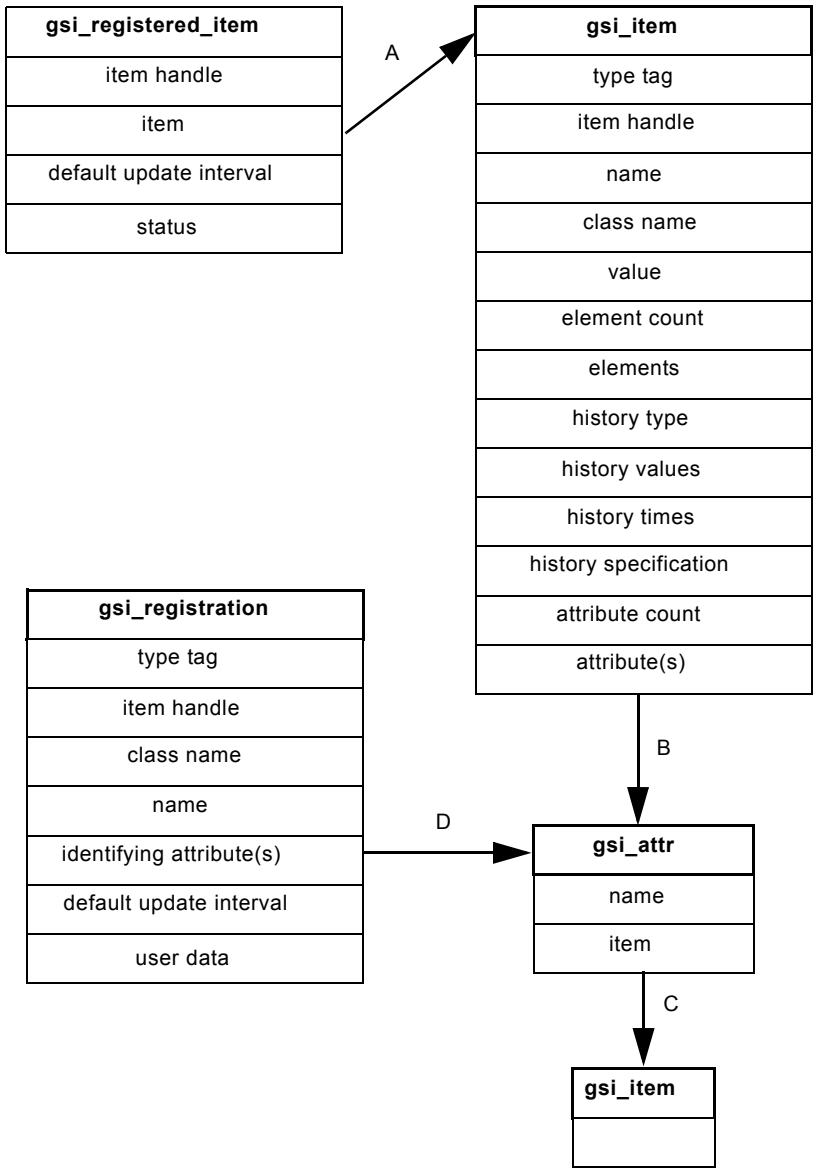
Accessing Data Structures through Other Data Structures

Some data structures have components that contain other data structures. This feature enables you to access one data structure through another, using API functions provided for this purpose.

For example, the API function *gsi_item_of_registered_item()* takes a *gsi_registered_item* structure as an argument and returns the *gsi_item* structure associated with the *gsi_registered_item*.

The following figure illustrates the G2 Gateway data structures, their components, and the API functions you can use to access data structures through their enclosing structures:

G2 Gateway Data Structures and Their Components



A gsi_item gsi_item_of_registered_item(gsi_registered_item)
B gsi_attr *gsi_attrs_of(gsi_item)
C gsi_item gsi_item_of_attr(gsi_attr)
D gsi_attr gsi_identifying_attr_of(gsi_registration,attribute_index)

Type Tags of G2 Gateway Data Structures

In GSI Version 3.1 and later versions, G2 Gateway assigns data type tags to all *gsi_registration* and *gsi_item* data structures to indicate the data type of their values. The data type tags enable the API functions to perform type-checking and to handle the structures appropriately to their type.

Setting Type Tags

API functions that set the *value* component of *gsi_item* structures also reset the type tags of these structures. For example, the function *gsi_set_int()* function sets the type tag of a *gsi_item* structure to *GSI_INTEGER_TAG* before setting the *value* component of the structure to an integer value.

Setting the Type to Null

The API function *gsi_set_type()* can set the type of a *gsi_item* structure to any non-list or non-array data type. *gsi_set_type()* specifies a default value of the appropriate type for the *gsi_item* structure.

gsi_set_type() is useful mainly for setting the type of a *gsi_registered_item* to null. You may want to do this if you are sending an array of *gsi_registered_item* structures back to G2 to update a set of GSI variables, but there are certain GSI variables that do not need to be updated. You can set the *gsi_registered_item* structures that correspond to these GSI variable to null, using *gsi_set_type()*.

Caution A G2 Gateway error results if your user code attempts return a *gsi_registered_item* to G2 with a null type and a *status* component value of 0 (OK). You can use the API function *gsi_set_status()* to set the *status* component of a *gsi_registered_item* structure. For information about the values to which you can set the status of *gsi_registered_item* structures, see Using the Gsi-Variable-Status Attribute.

G2 Gateway Data Structures and Functions for Data Transfer Operations

G2 Gateway uses different combinations of data structures and functions to perform different kinds of data transfer operations. The following table lists the data structures that G2 Gateway creates to support different data transfer operations:

G2 Operations and G2 Gateway Data Structures

G2 Operation	Data Structures that G2 Gateway Creates to Support this Operation
Ask G2 Gateway to set the value of a data point in an external system, using the value of a GSI variable in G2.	*gsi_registration* *gsi_registered_item* *gsi_item*
Ask G2 Gateway to update a GSI variable in G2.	*gsi_registration* *gsi_registered_item* *gsi_item*
Pass a copy of a G2 object to G2 Gateway as an argument to a remote procedure in G2 Gateway.	*gsi_item* (for the object) *gsi_attr* (for each attribute of the object) *gsi_item* (for each attribute of the object)
Pass a simple value to G2 Gateway as an argument to a remote procedure in G2 Gateway.	*gsi_item*
Pass the handle of a G2 item as an argument to a remote procedure in G2 Gateway.	*gsi_registration* *gsi_item* (the *type tag* of this structure is *GSI_HANDLE_TAG*)

The following sections describe the combinations of data structures and functions calls that support different kinds of data exchange between a G2 Gateway bridge and a G2 application.

Setting the Value of an External Data Point

The following steps summarize the data structures and functions that enable a G2 application to update the value of a data point in an external system using the value of a GSI variable, and to echo the value back to the GSI variable.

1. The set action in G2 sends the value of a GSI variable to the G2 Gateway bridge.
2. G2 Gateway creates a *gsi_registration* structure to store the information received from G2.
3. G2 Gateway calls the callback *gsi_receive_registration()*, passing to it the *gsi_registration* structure.
4. G2 Gateway creates *gsi_registered_item* and *gsi_item* structures. The *value* component of the *gsi_item* structure contains the value of the GSI variable.
5. G2 Gateway calls the callback *gsi_set_data()*, passing to it an array of one or more *gsi_registered_item* structures, which represent G2's requests to set values in the external system.
6. In *gsi_set_data()*, you can use API functions such as *gsi_int_of()* or *gsi_sym_of()* to access the value of the GSI variable. These functions can take the *gsi_registered_item* structure as their argument. They return the *value* component of the *gsi_item* structure, which stores the value of the GSI variable.
7. Also in *gsi_set_data()*, you add code that sets the value of the external data point with the value of the GSI variable, using the value returned from the *gsi_item* structure.
8. When the value of the external data point has been set, *gsi_set_data()* can call the API function *gsi_return_values()*, passing to it the *gsi_registered_item* structure that represents the GSI variable. *gsi_return_values()* echoes the value back to the GSI variable in G2.

Updating the Value of a GSI Variable

The following steps summarize the data structures and functions that enable a G2 application to update the value of a GSI variable using the value of a data point in an external system.

1. An update action or a collect data procedure statement in G2 requests G2 Gateway to send a value to a GSI variable in G2.
2. G2 Gateway creates a *gsi_registration* structure to store information associated with the request from G2.
3. G2 Gateway calls *gsi_receive_registration()*, passing to it the *gsi_registration* structure.

4 G2 Gateway creates *gsi_registered_item* and *gsi_item* structures.

5 G2 Gateway calls the callback *gsi_get_data()*, passing to it an array of one or more *gsi_registered_item* structures, which represent G2's requests for update values.

6 In *gsi_get_data()*, you add code to get the value of the data point in the external system.

7 Also in *gsi_get_data()*, you use API functions such as *gsi_set_int()* or *gsi_set_sym()* to set the *value* component of the *gsi_item* structure, using the value obtained from the external system. These API functions can take the *gsi_registered_item* structure as their argument.

8 Also in *gsi_get_data()*, you add a call to the API function *gsi_return_values()*, passing to it the *gsi_registered_item* structure. *gsi_return_values()* returns the value of the external data point to the GSI variable in G2.

Receiving Unsolicited Updates of GSI Variables

The following steps summarize the data structures and functions that enable a G2 application to receive unsolicited updates to the values of GSI variables., when the bridge obtains this data by polling the external system. The GSI variables are updated with the values of data points in an external system.

1 When G2 is started, it registers all data-served GSI variables.

In order for the GSI variables to receive unsolicited updates, you must set the poll-external-system-for-data attribute of your GSI interface object to yes.

2 G2 Gateway creates a *gsi_registration* structure for each registered GSI variable when the variable is activated.

3 G2 Gateway calls *gsi_receive_registration()* and passes to it the *gsi_registration* structures.

4 G2 Gateway calls the callback *gsi_g2_poll()*, to which you add the code that gets values for GSI variables from the external system.

5 Your *gsi_g2_poll()* calls *gsi_make_registered_items()* to allocate an array of *gsi_registered_item* structures. These structures represent the update values to be sent to GSI variables in G2.

6 Your *gsi_g2_poll()* calls API functions such as *gsi_set_int()* and *gsi_set_sym()* to set the values of the *gsi_registered_item* structures in the array, using the values obtained from the external system.

7 Your *gsi_g2_poll()* calls *gsi_return_values()*, passing to it the array of *gsi_registered_item* structures.

8 *gsi_return_values()* returns values to the last-recorded-value attribute of GSI variables.

The following steps summarize the data structures and functions that enable a G2 application to receive unsolicited updates to the values of GSI variables, when the external system sends data values to the bridge without having been polled by the bridge. The GSI variables are updated with the values of data points in an external system.

1. When G2 is started, it registers all data-served GSI variables.
2. G2 Gateway creates a *gsi_registration* structure for each registered GSI variable.
3. G2 Gateway calls *gsi_receive_registration()* and passes to it the *gsi_registration* structures.
4. The external system sends a new value for a registered GSI variable to the bridge.
5. The G2 Gateway user code calls *gsi_make_registered_items()* to allocate an array of *gsi_registered_item* structures. These structures represent the update values to be sent to GSI variables in G2.
6. The G2 Gateway user code calls API functions such as *gsi_set_int()* and *gsi_set_sym()* to set the values of the *gsi_registered_item* structures in the array, using the values obtained from the external system.
7. The G2 Gateway user code calls *gsi_return_values()*, passing to it the array of *gsi_registered_item* structures.
8. *gsi_return_values()* returns values to the last-recorded-value attribute of GSI variables.

Passing Objects through Remote Procedure Calls

The following steps summarize the data structures and functions that enable a G2 application to pass a copy of an object to G2 Gateway as an argument to a remote procedure call to a user-defined local function in the bridge.

1. G2 calls a remote procedure in G2 Gateway, passing a G2 object as an argument to the procedure.
2. G2 Gateway creates a *gsi_item* structure to represent the G2 object, and an array of *gsi_attr* structures that represent any attributes that G2 Gateway passes with the object.
3. G2 Gateway invokes the local function that G2 called as a remote procedure. G2 Gateway passes to the local function an array of one or more *gsi_item* structures, which includes the G2 object passed to G2 Gateway.
4. The local function can call *gsi_attrs_of()*, passing to it the *gsi_item* structure that represents the G2 object. *gsi_attrs_of()* returns the array of *gsi_attr* structures that represent the attributes of the object.

5 The local function can call API functions to get and set components of the *gsi_item* and *gsi_attr* structures.

   For example, it can call *gsi_int_of()* to get the *value* component of a *gsi_item* structure, and *gsi_set_int()* to set the *value* component. It can call *gsi_attr_name_of()* to get the *name* component of a *gsi_attr* structure, and *gsi_set_attr_name()* to change the *name* component. For information about these API functions, see API Functions.

6 The local function can perform any other operations required by your application.

7 The local function can call *gsi_rpc_return_values()*, which returns data associated with the object to G2. The local function must pass to *gsi_rpc_return_values()* an array of one or more *gsi_item* structures, which can include the structure representing the G2 object.

8 G2 creates a G2 object based on the *gsi_item* structure that it receives through *gsi_rpc_return_values()*.

   The *class name* component of the *gsi_item* must specify the name of an existing class definition in G2; otherwise, G2 cannot create an object based on the *gsi_item*, and an error occurs.

Passing Items as Handles

The following steps summarize the data structures and functions that enable a G2 application to pass an item handle to G2 Gateway through a remote procedure call. An item handle is a value that identifies a particular G2 item.

1 G2 calls a local function in G2 Gateway that is declared in G2 as a remote procedure. The remote procedure declaration in G2 must use the as handle grammar.

   The item handle is passed to G2 Gateway through this remote procedure call. For information about how to declare a remote procedure to send items handles to and receive them from G2 Gateway, see the *G2 Reference Manual*.

2 G2 Gateway creates a *gsi_registration* structure to store information associated with the item handle.

3 G2 Gateway calls *gsi_receive_registration()*, passing to it the *gsi_registration* structure.

4 G2 Gateway creates a *gsi_item* structure to represent the item handle.

5 G2 Gateway invokes the local function that G2 called as a remote procedure, passing to it an array of one or more *gsi_item* structures, which can include the item handle.

6 The local function performs any operations required by your application.

7 The local function calls *gsi_rpc_return_values()*, passing to it an array of one or more *gsi_item* structures, which can include your item handle. *gsi_rpc_return_values()* returns the item handle to the remote procedure call in G2.

Note Passing item handles rather than actual items provides a way for G2 to reference the items rather than making copies of them. This can reduce processing time, save space in your KB and prevent unnecessary traffic across your network.

Allocating and Reclaiming G2 Gateway Data Structures

G2 Gateway allocates an appropriate data structure automatically when G2 registers a variable or object, or when G2 Gateway receives an object passed to it from G2.

Your G2 Gateway user code must allocate a data structure explicitly only when:

- Your G2 Gateway bridge needs to send a variable or object to G2 that it has not previously received from G2.
- You declare a variable to be a pointer to a data structure in a callback function.

  In callback functions, a G2 Gateway data structure is not allocated automatically when you declare a variable to be a data structure. Instead, you must first allocate a G2 Gateway data structure explicitly, by calling the API function that allocates a data structure of the type that you want to use. Then you can assign the structure that you allocated to the declared variable.

To allocate data structures, use the following API functions: *gsi_make_attrs()*, *gsi_make_attrs_with_items()*, *gsi_make_items()*, and *gsi_make_registered_items()*.

These API functions return arrays of one or more structures. For this reason, if you allocate a *gsi_attr* structure as follows:

```
gsi_attr *my_att-array = gsi_make_attrs(1);
```

you must:

- Declare any variable that points to this *gsi_attr* data structure.
- Access this variable by specifying the zeroth element of the array, as for example:

```
gsi_attr my_att = my_attr_array[0];
```

Caution If you are allocating G2 Gateway data structures dynamically, in a part of your user code that can be executed more than once in the lifetime of the G2 Gateway bridge process, be sure to reclaim the data structures as soon as you no longer need them. You can reclaim data structures using the following API functions: `gsi_reclaim_attrs()`, `gsi_reclaim_attrs_with_items()`, `gsi_reclaim_items()`, `gsi_reclaim_registered_items()`.

For information about the API functions, see API Functions.

gsi_registration Data Structures

G2 Gateway creates a `gsi_registration` structure when G2 registers a GSI variable. This happens the first time when G2 requests G2 Gateway to read from or write to the GSI variable. G2 Gateway also creates a `gsi_registration` structure when G2 registers an item handle that it passes to G2 Gateway through a remote procedure call.

The `gsi_registration` structure stores information associated with the registered item, such as its handle and a list of its identifying attributes. G2 Gateway uses this information when it responds to all subsequent requests from G2 to read from or write to the variable.

A `gsi_registration` structure remains in existence until the item that it represents is deregistered. For information about how items are deregistered, see Deregistering Items Automatically.

Registering a GSI Variable or Item Handle

G2 Gateway calls the callback function `gsi_receive_registration()` when G2 registers a GSI variable or item handle with the G2 Gateway bridge. G2 Gateway passes the `gsi_registration` structure for the variable or handle to the callback.

You can use `gsi_receive_registration()` to perform tasks such as initializing an external data point, allocating memory, or returning the variable's network handle to an attribute of the variable for some future use.

For more information about `gsi_receive_registration()`, see Callback Functions.

Getting a gsi_registration Structure

The following API function returns the `gsi_registration` structure associated with a given handle and context:

`gsi_registration gsi_registration_of(`*item_handle*, *context*`)`

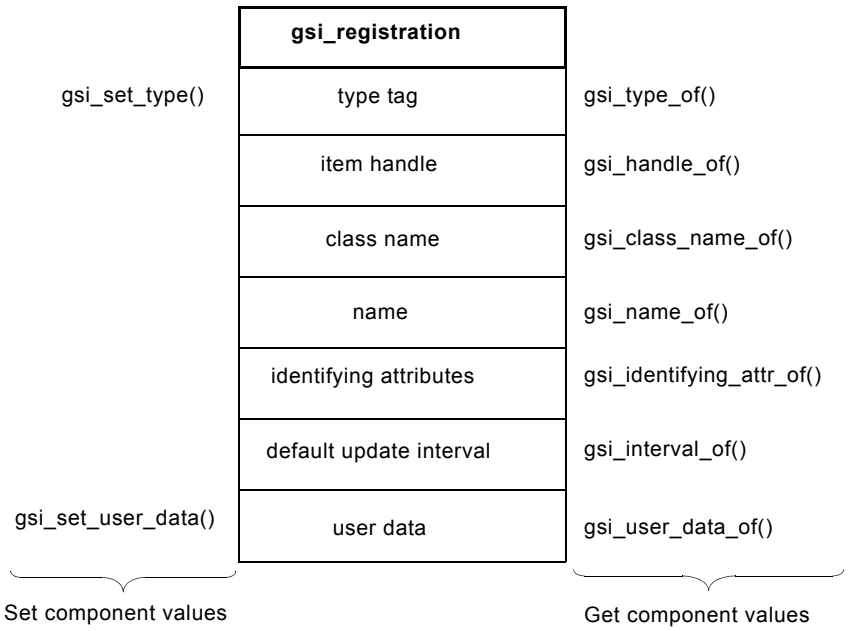
For more information about `gsi_registration_of`(), see API Functions.

Note You do not need to use an API function call to create a *gsi_registration* data structure.

Accessing Components of a gsi_registration Structure

The following figure illustrates the components of a *gsi_registration* structure that you can access with API functions.

Components of gsi_registration Structures Accessed through API Functions



The following table describes the components of a *gsi_registration* structure that your user code can access through API functions:

Components of a gsi_registration Structure

Component	Functions for Accessing
type tag One of: *GSI_FLOAT64_TAG*, *GSI_INTEGER_TAG*, *GSI_LOGICAL_TAG*, *GSI_SYMBOL_TAG*, *GSI_STRING_TAG*, *GSI_FLOAT64_ARRAY_TAG*, *GSI_INTEGER_ARRAY_TAG*, *GSI_LOGICAL_ARRAY_TAG*, *GSI_STRING_ARRAY_TAG*, *GSI_SYMBOL_ARRAY_TAG*, *GSI_ITEM_ARRAY_TAG*, *GSI_VALUE_ARRAY_TAG*, *GSI_INTEGER_LIST_TAG*, *GSI_SYMBOL_LIST_TAG*, *GSI_STRING_LIST_TAG*, *GSI_LOGICAL_LIST_TAG*, *GSI_FLOAT64_LIST_TAG*, *GSI_ITEM_LIST_TAG*, *GSI_VALUE_LIST_TAG*	*gsi_int gsi_type_of (registration)*
item handle A *gsi_int* value, used by G2 Gateway functions to reference the registered G2 item within a particular context.	*gsi_int gsi_handle_of (registration)*
class name The class name of the registered G2 item.	*gsi_symbol gsi_class_name_of (registration)*

Components of a gsi_registration Structure

Component	Functions for Accessing
name The name of the registered G2 item. Can be either a GSI variable registered for data service, or an item handle passed to G2 Gateway as an argument of a remote procedure call.	*gsi_char *gsi_name_of* (*registration*)
identifying attributes A list of the values of the identifying attributes of the registered GSI variable. These values together uniquely identify each variable that receives G2 Gateway data service through this GSI interface object. Can be from 1 to 6 simple attributes. The identifying attributes stored in the *identifying attributes* component do not include the names of the attributes.	*gsi_attr gsi_identifying_ attr_of* (*registration, attribute_index*)

Components of a gsi_registration Structure

Component	Functions for Accessing
default update interval The default update interval of the GSI variable that is registered for data service. **Note**: There is no default update interval for items passed as handles through remote procedure calls.	*double gsi_interval_of (registration)*
user data Reserved for use by your G2 Gateway user code. The user data must be of type *gsi_item_user_data_type*. You can use this component to associate any application-specific information with the *gsi_registration*, such as data that the G2 Gateway bridge process receives from a PLC or other external device. G2 Gateway itself never reads from nor writes to the *user data* component.	*void gsi_set_user_data (registration, user_data)* *gsi_item_user_data_type gsi_user_data_of (registration)*

gsi_registered_item Data Structures

gsi_registered_item contains a structure representing a GSI variable that G2 has registered across a G2-to-G2 Gateway connection. A GSI variable is an instance of a G2 variable class (a logical-, quantitative-, float-, integer-, symbolic-, or text-variable) or subclass that includes gsi-data-service as one of its direct superior classes.

G2 Gateway creates a *gsi_registered_item* structure when G2 asks G2 Gateway to get a new value for a GSI variable from an external system, or when it asks G2 Gateway to write the value of a GSI variable to a data point in an external system. A *gsi_registered_item* structure remains in existence for the duration of the read or write operation.

Returning Values to a GSI Variable

The following API functions take *gsi_registered_item* structures as arguments and return values to the corresponding GSI variables in G2:

```
void gsi_return_timed_values(registered_items, count, context)
void gsi_return_values(registered_items, count, context)
```

Setting Arguments of GSI Variables

The following API functions return values to registered GSI variables and set one or more of their attributes:

```
void gsi_return_attrs(registered_item, attributes, count, context)
void gsi_return_timed_attrs(registered_item, attributes,
   count, context)
```

Callbacks that Access gsi_registered_item Structures

The following callback functions access *gsi_registered_item* structures. In each of the following callbacks, *registered_items* is an array of one or more *gsi_registered_item* structures:

```
void gsi_get_data(registered_items, count)
void gsi_set_data(registered_items, count)
void gsi_receive_deregistrations(registered_items, count)
```

For information about these functions, see Callback Functions.

Allocating and Reclaiming gsi_registered_item Structures

Your G2 Gateway user code must allocate a *gsi_registered_item* structure in order to transfer an item to G2 when the item to be transferred did not originate in G2. In this case, there is no automatically created *gsi_registered_item* structure to represent the item.

The following API function allocates an array of *gsi_registered_item* structures:

```
gsi_registered_item *gsi_make_registered_items(count)
```

You can then pass the array of *gsi_registered_item* structures to the API function *gsi_return_values()*, which sends the structures to G2.

The following API function reclaims *gsi_registered_item* structures:

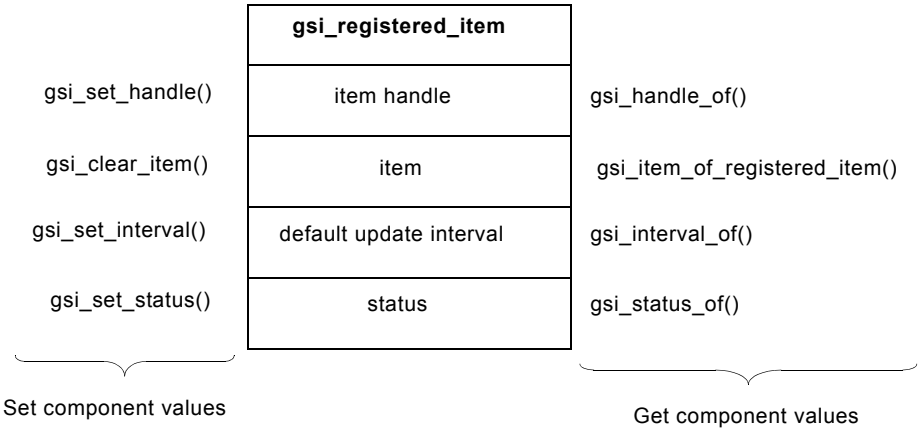
```
void gsi_reclaim_registered_items(registered_items)
```

For information about allocating and reclaiming *gsi_registered_item* structures, see Passing Items as Handles.

Accessing Components of a gsi_registered_item Structure

The following figure illustrates the components of a *gsi_registered_item* structure that you can access with API functions.

Components of gsi_registered_item Structures Accessed through API Functions



The following table lists the components of the *gsi_registered_item* structure that your user code can access through API functions:

Components of gsi_registered_item Structures

Component	API Functions for Accessing
item handle A *gsi_int* value, which refers to an object in G2 that has been registered for G2 Gateway data service through the G2 Gateway callback function *gsi_receive_registration()*. To get the *gsi_registration* associated with this registered item, you can pass the handle value returned by *gsi_handle_of()* to the *gsi_registration_of()* API function, which returns the *gsi_registration*.	*gsi_int gsi_handle_of (registered_item)* *void gsi_set_handle (registered_item, handle_value)*
item *gsi_item* contained in this *gsi_registered_item*. Some API functions that access components of a *gsi_item* structure can take as an argument a *gsi_registered_item* structure that points to the *gsi_item*. Other API functions can access the *gsi_item* only directly, and require the *gsi_item* itself as an argument. To find out whether an API function can access a component of a *gsi_item* structure directly, see the description of that function in API Functions.	*gsi_item gsi_item_of_registered_item(registered_item)* *void gsi_clear_item (registered_item)*

Components of gsi_registered_item Structures

Component	API Functions for Accessing
default update interval Current default update interval, corresponding to the default-update-interval attribute of a G2 variable.	*double gsi_interval_of (registered_item)* *void gsi_set_interval (registered_item, interval)*
status A status code, which G2 Gateway sends to the gsi-variable-status attribute of the GSI variable in G2 that corresponds to this *gsi_registered_item*. G2 Gateway initially sets *status* to 0, indicating no error. Your user code can set *status* to a positive or negative integer that reflects the current status of the external data point to which the GSI variable is mapped. When G2 receives the registered item, it updates the gsi-variable-status attribute of the corresponding GSI variable.	*gsi_int gsi_status_of (registered_item)* *void gsi_set_status (registered_item, status)*

gsi_item Structures

gsi_item points to a G2 Gateway data structure that represents an item or a value. G2 Gateway creates a *gsi_item* structure for each data-served variable, object, item handle, or simple value that G2 passes to G2 Gateway.

Verifying that an Item is an Item

The following API function determines whether *item* is a member of the G2 class item:

```
gsi_int gsi_is_item(item)
```

gsi_item Structures as Arguments of Remote Procedure Calls

The following API functions accept *gsi_item* structures as arguments:

void gsi_rpc_call(function_handle, gsi_item_arguments, context)

void gsi_rpc_start(function_handle, gsi_item_arguments, context)

void gsi_rpc_return_values
(gsi_item_arguments, count, call_handle, context)

Copying Contents of a gsi_item Structure

void gsi_simple_content_copy(destination_item, source_item)

API Functions that Return gsi_item Structures

The following API functions return *gsi_item* structures:

gsi_item gsi_item_of_attr(gsi_attr)

gsi_item gsi_item_of_registered_item(registered_item)

API Functions that Allocate and Reclaim gsi_item Structures

The following API functions allocate or reclaim *gsi_item* structures:

*gsi_item *gsi_make_items(count)*

*gsi_attr *gsi_make_attrs_with_items(count)*

void gsi_reclaim_items(items)

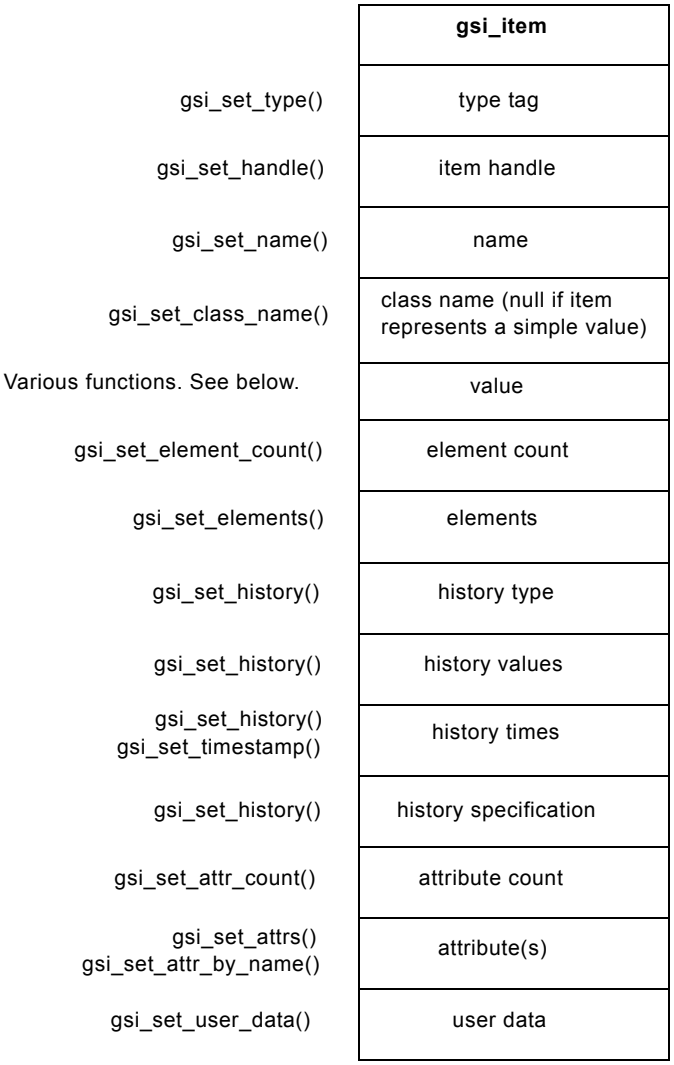
Returning gsi_item Values and Attributes to G2

The API functions that return the values and attribute values of a *gsi_item* structure to a GSI variable all take as an argument the *gsi_registered_item* that contains the *gsi_item*, rather than the *gsi_item* itself. These functions are: *gsi_return_values()*, *gsi_return_timed_values()*, *gsi_return_timed_attrs()*, and *gsi_return_attrs()*. For information about these functions, see API Functions.

Components of a gsi_item Structure

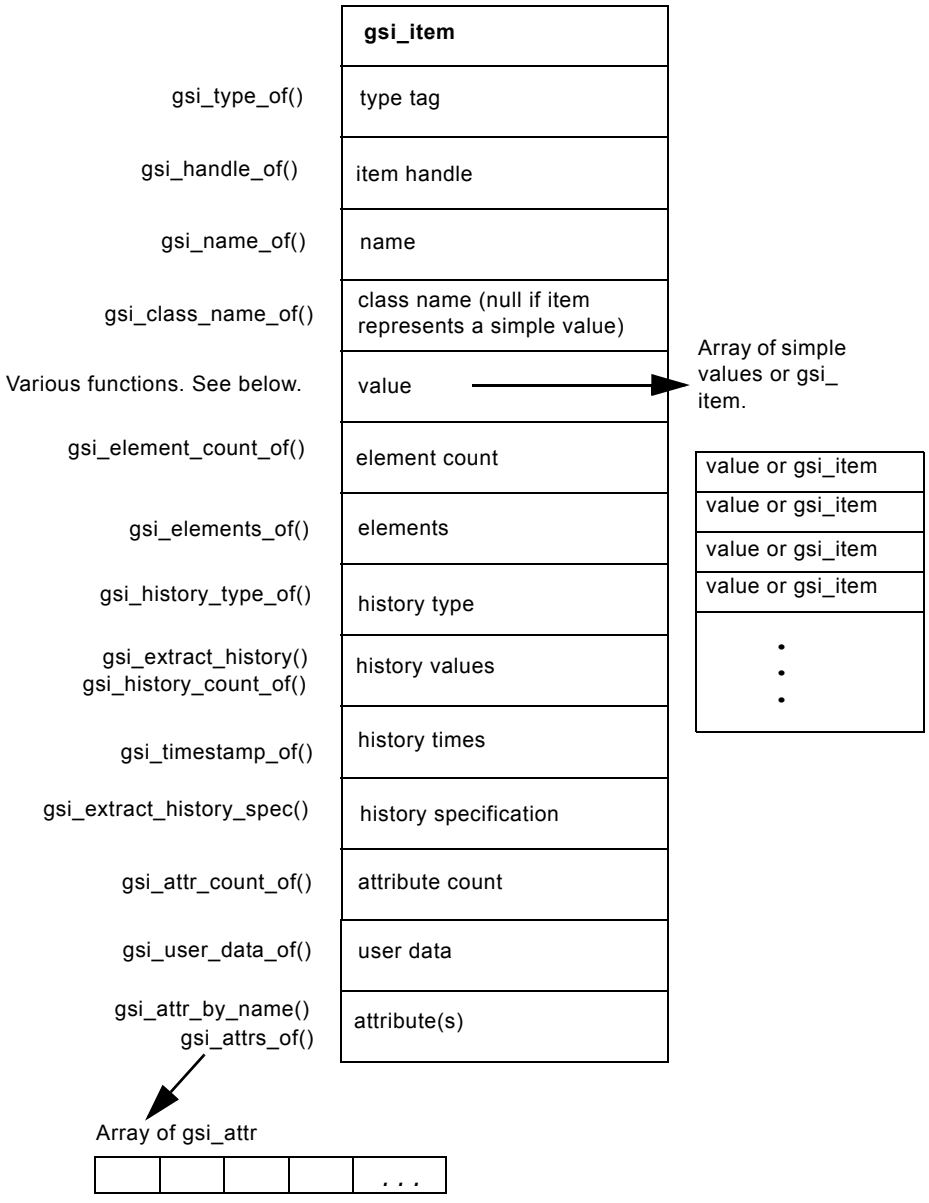
The following figure illustrates the API functions that set values of components of *gsi_item* structures:

API Functions that Set Values of gsi_item Components



The following figure illustrates the API functions that get the values of components of a *gsi_item* structure:

API Functions that Get Values of gsi_item and gsi_attr Components



The following table describes the components of the *gsi_item* structure that your user code can access through API functions:

Components of gsi_item Structures

Component	API Functions for Accessing
type tag When representing an item (class name component has a value), one of: *GSI_NULL_TAG*, *GSI_INTEGER_ARRAY_TAG*, *GSI_SYMBOL_ARRAY_TAG*, *GSI_STRING_ARRAY_TAG*, *GSI_LOGICAL_ARRAY_TAG*, *GSI_FLOAT64_ARRAY_TAG*, *GSI_ITEM_ARRAY_TAG*, *GSI_VALUE_ARRAY_TAG*, *GSI_INTEGER_LIST_TAG*, *GSI_SYMBOL_LIST_TAG*, *GSI_STRING_LIST_TAG*, *GSI_LOGICAL_LIST_TAG*, *GSI_FLOAT64_LIST_TAG*, *GSI_ITEM_LIST_TAG*, *or* *GSI_VALUE_LIST_TAG* When representing a value (class name component is null), one of: *GSI_NULL_TAG*, *GSI_HANDLE_TAG* *(RPC arguments only)*, *GSI_INTEGER_TAG*, *GSI_SYMBOL_TAG*, *GSI_STRING_TAG*, *GSI_LOGICAL_TAG*, *GSI_FLOAT64_TAG*	Set the *type tag* component: *void gsi_set_type(item)* *void gsi_clear_item(item)* Get the *type tag* component: *gsi_int gsi_type_of(item)*

Components of gsi_item Structures

Component	API Functions for Accessing
item handle A *gsi_int* value, which refers to some object in G2 that has been registered through the API function *gsi_receive_registration()*. G2 generates handles for the following objects: • A GSI variable, when it is registered for data service. • An object passed to G2 Gateway through a remote procedure call declared with the as handle grammar. • An object registered by the G2 system procedure g2-register-on-network().	*gsi_int gsi_handle_of(item)* *void gsi_set_handle(item, handle_value)*
name The name of the GSI variable registered for data service, or of an object passed to G2 Gateway through a remote procedure call.	*gsi_char *gsi_name_of(item)* *void gsi_set_name(item, name)* (*gsi_item*)
class name Name of the G2 class. If the *gsi_item* structure represents a simple value in G2, this component is null. In this case, only the *type tag*, *value*, *element count*, and *elements* components of this *gsi_item* are meaningful.	*gsi_symbol gsi_class_name_of (item)* *gsi_int gsi_is_item(item)* *void gsi_set_class_name (item, class_name)*

Components of gsi_item Structures

Component	API Functions for Accessing
value One or more values.	Set the *value* component: *void gsi_set_flt(item, float_value)* *void gsi_set_flt_array(item, doubles_array, count)* *void gsi_set_flt_list(item, doubles_array, count)* *void gsi_set_int(item, integer_value)* *void gsi_set_int_array(item, integers_array, count)* *void gsi_set_int_list(item, integers_array, count)* *void gsi_set_log(item, truth_value)* *void gsi_set_log_array(item, truth_values_array, count)* *void gsi_set_log_list(item, truth_values_array, count)* *void gsi_set_str(item, text_value)* *void gsi_set_str_array(item, text_values_array, count)* *void gsi_set_str_list(item, text_values_array, count)* *void gsi_set_sym(item, symbol_value)* *void gsi_set_sym_array(item, symbol_values_array, count)* *void gsi_set_sym_list(item, symbol_values_array, count)* Clear memory associated with the *value* component: *void gsi_clear_item(item)*

Components of gsi_item Structures

Component	API Functions for Accessing
value	Get the *value* component: *double *gsi_flt_array_of(item)* *double *gsi_flt_list_of(item)* *double gsi_flt_of(item)* *gsi_int *gsi_int_array_of(item)* *gsi_int *gsi_int_list_of(item)* *gsi_int gsi_int_of(item)* *gsi_int *gsi_log_array_of(item)* *gsi_int *gsi_log_list_of(item)* *gsi_int gsi_log_of(item)* *gsi_char **gsi_str_array_of(item)* *gsi_char **gsi_str_list_of(item)* *gsi_char *gsi_str_of(item)* *gsi_char **gsi_sym_array_of(item)* *gsi_char **gsi_sym_list_of(item)* *gsi_symbol *gsi_sym_of(item)*
element count Number of elements in the *value* component.	*gsi_int gsi_element_count_of (item)* *void gsi_set_element_count (item, count)* *void gsi_clear_item(item)*
elements The elements in the *value* component.	*gsi_item *gsi_elements_of (item)* *void gsi_set_elements (item, elements_array, count, type_tag)*
history type Type of the item's associated history data, if any. This component is meaningful only when the *class name* component of this *gsi_item* has a value that names a subclass of variable or parameter in G2.	*gsi_int gsi_history_type_of (item)* *void gsi_clear_item(item)*

Components of gsi_item Structures

Component	API Functions for Accessing
history values C array that contains the item's associated history data, if any. This component is meaningful only when the *class name* component of this *gsi_item* has a value that names a subclass of variable or parameter in G2.	*gsi_int gsi_extract_history (item, values_address, timestamps_address, type_address)* *gsi_int gsi_history_count_of (item)* *void gsi_set_history (item, values, timestamps, count, type,maximum_count, maximum_age, min_interval)* *void gsi_clear_item(item)*
history times Array of floating-point timestamp values, if any. This component is meaningful only when the *class name* component of this *gsi_item* has a value that names a subclass of variable or parameter in G2.	*void gsi_set_history (item, values, timestamps, count, type, maximum_count, maximum_age, min_interval)* *void gsi_set_timestamp(item, timestamp_value)* *double gsi_timestamp_of(item)* *void gsi_clear_item(item)*
history specification Maximum number of history data values to retain, the maximum age of history data values, and the minimum interval between timestamps, which correspond to those specified in the history-keeping-specification attribute of a G2 variable or parameter. This component is meaningful only when the *class name* component of this *gsi_item* has a value that names a subclass of variable or parameter in G2.	*gsi_int gsi_extract_history (item, values_address, timestamps_address, type_address)* *gsi_int gsi_extract_history_spec (item, maximum_count_address, maximum_age_address, minimum_interval_address)* *void gsi_set_history (item, values, timestamps, count, type, maximum_count, maximum_age, min_interval)* *void gsi_clear_item(item)*

Components of gsi_item Structures

Component	API Functions for Accessing
attribute count Number of attributes in the *attributes* component. This component is meaningful only when the *class name* component of this *gsi_item* has a value.	*gsi_int gsi_attr_count_of (item)* *void gsi_set_attr_count (item,count)* *void gsi_clear_item (registered_item)*
user data Reserved for use by your G2 Gateway user code. Can point to any 32-bit entity, including memory allocated by a user-written function. You can use this component to associate any application-specific information with the *gsi_registration*, such as data that the G2 Gateway bridge process receives from a PLC or other external device. G2 Gateway itself never reads from nor writes to the *user data* component.	*void gsi_set_user_data (registration, user_data)* *gsi_int gsi_user_data_of (registration)*

Components of gsi_item Structures

Component	API Functions for Accessing
attribute(s) An array of *gsi_attr* structures, representing the attributes of this class of item. This component is meaningful only when the *class name* component of this *gsi_item* has a value.	*gsi_attr *gsi_attrs_of(item)* *gsi_attr *gsi_attr_by_name (item,search_name)* *void gsi_set_attrs (item, new_attributes, count)* *void gsi_set_attr_by_name (destination_item, search_name, source_item)* *void gsi_set_item_of_attr (attribute, source_item)* *void gsi_clear_item(item)* *void gsi_return_attrs (registered_item, attributes, count, context)* *void gsi_return_timed_attrs (registered_item, attributes, count, context)*

gsi_attr Structures

A gsi_attr points to an internal G2 Gateway structure that represents an attribute of an item. The attribute can contain an attribute value or can contain a *gsi_item* structure that represents a G2 object embedded in an attribute of another G2 object.

API Functions that Return gsi_attr Structures

The following API functions return *gsi_attr* structures:

gsi_attr gsi_identifying_attr_of(registration, identifying_attribute_index)

*gsi_attr *gsi_attr_by_name(item-or-attribute, search-name)*

*gsi_attr *gsi_attrs_of(item-or-attribute)*

API Functions that Allocate and Reclaim gsi_attr Structures

The following API functions allocate or reclaim *gsi_attr* structures:

```
gsi_attr *gsi_make_attrs(count)
gsi_attr *gsi_make_attrs_with_items(count)
void gsi_reclaim_attrs(attributes)
void gsi_reclaim_attrs_with_items(attributes)
```

For information about allocating and reclaiming *gsi_attr* and other G2 Gateway data structures, see the following section.

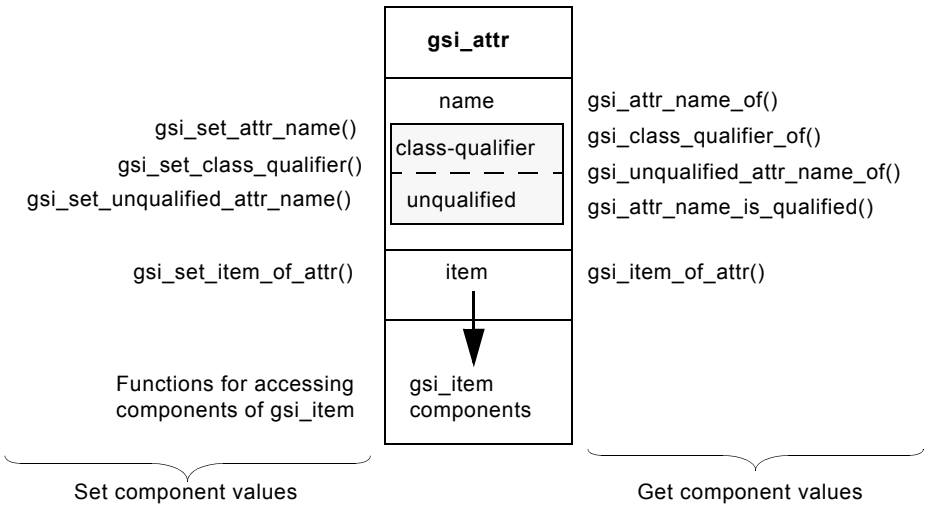
Components of a gsi_attr Structure

A *gsi_attr* structure has a user-accessible *name* component and an *item* component that can contain a *gsi_item* structure. This *gsi_item* structure represents an G2 object that is embedded in the G2 object attribute represented by the *gsi_attr* structure.

If the *item* component contains a *gsi_item* structure, the *gsi_attr* structure can be used as an argument of API functions that access components of *gsi_item* structures. In this case, the API functions access the components of the *gsi_item* structure indirectly, through the *gsi_attr* structure that points to the *gsi_item* structure.

The following figure illustrates the components of a *gsi_attr* structure that you can access through API functions.

API Functions that Set and Get Values of gsi_attr Components



The following table describes the components of the *gsi_attr* structure that your user code can access through API functions.

Components of gsi_attr Structures

Component	API Functions for Accessing
name The name of this *gsi_attr* structure, which represents the name of an attribute of a G2 object.	*gsi_char *gsi_attr_name_of(attribute)* *gsi_char *gsi_class_qualifier_of (attribute)* *gsi_char *gsi_unqualified_attr_name_of (attribute)* *gsi_int gsi_attr_name_is_qualified (attribute)* *void gsi_set_attr_name(attribute, attribute_name)* *void gsi_set_class_qualifier (attribute, attribute_name)* *void gsi_set_unqualified_attr_name (attribute, attribute_name)*
item A *gsi_item* structure that represents a G2 item embedded in the G2 attribute that this *gsi_attr* represents.	*gsi_item gsi_item_of_attr(attribute)* *void gsi_set_item_of_attr(attribute, source_item)*

gsi_symbol Structures

The *gsi_symbol* data structure helps improve performance because it is more efficient to compare symbols than to compare text strings. Symbols are like text strings, but they are stored differently. The system stores all symbols in a table. When creating a symbol given a text string, the system checks for an existing symbol having the same name as the given text string. If the system finds a symbol with the same name, it always uses the symbol.

When the *GSI_NEW_SYMBOL_API* runtime option is in effect, a symbol is a (void *) pointer that does not change through the lifetime of the G2 Gateway process. Calling *gsi_make_symbol* with a text string having the same contents as in an earlier call always returns exactly the same result.

Symbols and text strings cannot be used interchangeably. For example, using *gsi_set_sym* on a text attribute will fail and using *gsi_set_str* on a symbol attribute will fail.

Note If *gsi_use_new_symbol_api* is *not* in effect, the *gsi_make_symbol* and *gsi_symbol_name* functions return a result as a *gsi_char** type, which is simply a copy of the argument string.

API Functions that Return gsi_symbol Structures

The following API functions return *gsi_symbol* structures:

gsi_attr_name_of()

gsi_class_name_of()

gsi_class_qualifier_of()

gsi_make_symbol()

gsi_name_of()

**gsi_sym_array_of()*

**gsi_sym_list_of()*

gsi_sym_of()

gsi_unqualified_attr_name_of()

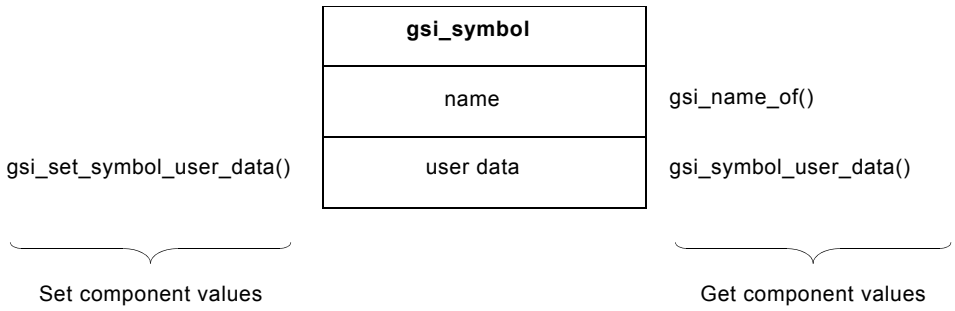
gsi_symbol_user_data()

An API Function that Allocates a gsi_symbol Structure

The API function *gsi_set_symbol_user_data()* allocates a *gsi_symbol* structure.

Accessing Components of a gsi_symbol Structure

The following figure illustrates the components of a *gsi_symbol* structure that you can access with API functions:



Note Because you cannot set the name of a symbol, create a new symbol using *gsi_make_symbol*.

The following table lists the components of the *gsi_symbol* structure that your user code can access through API functions:

Components of gsi_symbol Structures

Component	API Functions for Accessing
name A symbol giving the name of the specified data structure. The symbol persists only as long as the data structure with which it is associated exists. If your user code needs to keep the symbol for longer than the life-span of the data structure, it must copy the symbol into memory that it has allocated itself.	*gsi_symbol gsi_name_of (item_attr_or_reg)*
user data Reserved for use by your G2 Gateway user code. Can point to any 32-bit entity, including memory allocated by a user-written function. You can use this component to associate any application-specific information with the *gsi_symbol*. G2 Gateway itself never reads from nor writes to the *user data* component.	*void gsi_set_symbol_user_data (symbol, symbol_user_data)* *gsi_symbol_user_data_type gsi_symbol_user_data (symbol)*

8

Callback Functions

Describes the callback functions that you complete to implement your G2 Gateway user code.

Introduction

Callback functions form the basis of your G2 Gateway user code. G2 Gateway invokes callback functions *automatically*, when network events on a connection to a G2 KB occur. G2 Gateway invokes each callback to respond to one particular kind of event, such as the activation of a GSI interface or a request by G2 for a value for a GSI variable.

You complete the code of stub versions of callback functions provided with G2 Gateway and provide the code for callback functions not provided with G2 Gateway in stub form to make them respond to these network events in the ways required by your application.

G2 Gateway invokes callback functions only while your G2 Gateway bridge process is executing under the control of `gsi_run_loop()`, the API function that establishes the main event-processing loop of your G2 Gateway bridge process. For information about `gsi_run_loop()`, see API Functions.

Caution Do not attempt to invoke callback functions directly from your G2 Gateway user code. G2 Gateway automatically invokes all callbacks at the appropriate times.

Standard Callback Functions

There are a total of 23 standard callback functions, 12 of which existed in GSI 4.1 and 11 of which were added in G2 Gateway 5.0. For compatibility with GSI 4.1, there are different rules for using the 4.1 functions than for using the 5.0 functions.

Callbacks that Existed in GSI 4.1	Callbacks Added in G2 Gateway 5.0
gsi_set_up() *gsi_get_tcp_port()* *gsi_initialize_context()* *gsi_pause_context()* *gsi_resume_context()* *gsi_shutdown_context()* *gsi_receive_registration()* *gsi_receive_deregistrations()* *gsi_g2_poll()* *gsi_get_data()* *gsi_set_data()* *gsi_receive_message()*	*gsi_close_fd()* *gsi_error_handler()* *gsi_missing_procedure_handler()* *gsi_open_fd()* *gsi_read_callback()* *gsi_run_state_change()* *gsi_write_callback()* *gsi_start_context()* *gsi_reset_context()* *gsi_writing_fd()* *gsi_not_writing_fd()*

Using Standard Callback Functions

The way in which you use the standard callback functions differs depending on whether:

- You are using the GSI 4.1 standard callback functions.
- You are using the standard callback functions introduced in G2 Gateway 5.0.
- Your code is linked to G2 statically.
- Your code is linked using dynamic libraries (DLLs).

The following sections provide instructions for each.

Using GSI 4.1 Callbacks with G2 Gateway Linked Statically

The GSI 4.1 callback functions have required argument signatures and required names, you cannot rename these functions.

G2 Gateway provides the argument signature for the GSI 4.1 callback functions. The only requirement for using these callbacks is that a definition for each must be included in your user code. The definition can be only a stub if you do not intend to use the callback. See the section, Using Stub Versions of GSI 4.1 Callbacks, for more information.

Using GSI 4.1 Callbacks with G2 Gateway Linked Dynamically

The GSI 4.1 callback functions have required argument signatures and required names. You cannot rename these functions.

To use GSI 4.1 callback functions when you link G2 Gateway using DLLs:

1. Include a definition for each callback in your user code.

   The definition can be only a stub if you do not intend to use the callback. See the section, Using Stub Versions of GSI 4.1 Callbacks, for more information.

2. Use the standard *gsi_main.h* header file.

   The *gsi_main.h* file contains a prototype declaration for each GSI 4.1 callback function.

3. Set the C preprocessor flag *GSI_USE_DLL* when you compile your application.

   See Defining C Preprocessor Flags for instructions on defining a C preprocessor flag when you compile your G2 Gateway application.

4. Include a call to the *GSI_SET_OPTIONS_FROM_COMPILE()* before the call to *gsi_start()* in your *main()* function.

The *gsi_main.c* file includes a call to *GSI_SET_OPTIONS_FROM_COMPILE()*. If you are using the standard *gsi_main.c* file, you do not need to do anything for this step.

Setting the compile flag *GSI_USE_DLL* and including a call to *GSI_SET_OPTIONS_FROM_COMPILE()* in your main() function results in G2 Gateway calling the *gsi_initialize_callbacks()* API function for each GSI 4.1 callback function.

Using Stub Versions of GSI 4.1 Callbacks

For each callback function that existed in GSI 4.1, you must include a definition. The definition can be only a stub if you do not intend to use the callback.

G2 Gateway provides an uncompleted **stub** version of each of the GSI 4.1 standard named callback functions, in a file named *skeleton.c*. You can copy the *skeleton.c* file to make your own source file, in which you can complete the code of the stub functions.

In order for your G2 Gateway user code to link successfully, it must include *all* the callback functions in *skeleton.c*. If you do not intend to use a particular callback function in the *skeleton.c* file, your user code must nevertheless include the stub version of that callback.

Note The *skeleton.c* file contains two utility functions, *gsi_show_callback()* and *gsi_show_registered_items()*, that are invoked by several of the stub callbacks in the file. These functions are provided for convenience only; callbacks are not required to use them. *gsi_show_callback()* and *gsi_show_registered_items()* are not themselves callback functions and are not part of the G2 Gateway library of API functions.

Using G2 Gateway 5.0 Callbacks with G2 Linked Statically or Dynamically

The callback functions introduced in G2 Gateway 5.0 have required argument signatures; however, the names can be user defined. The procedure to use 5.0 callback functions is the same regardless of whether you link with G2 Gateway statically or using dynamic libraries (DLLs).

To declare and install G2 Gateway Version 5.0 callback functions:

1 Declare each callback function by calling the appropriate macro from your header file.

G2 Gateway provides you with macros that create the appropriate prototype for each 5.0 callback function. The format for the macros are:

specifier declare_callback_name(callback_name);

For example:

```
extern declare_gsi_error_handler(gsi_error_handler);
```

Note If you are using a C compiler that supports ANSI C, these macros generate prototype declarations for the callbacks; otherwise, Kernighan and Ritchie style declarations are generated.

2 Install each callback function by calling the appropriate install-macro before the call to *gsi_start()* in your *main()* function.

G2 Gateway provides you with install-macros that install each 5.0 callback function. The format for the macros are:

specifier install_callback_name(callback_name);

For example:

```
static gsi_install_error_handler(gsi_error_handler);
```

The install-macro expands into a call to *gsi_initialize_callbacks()*, with the correct argument list.

Note If you do not use the install-macros, you can initialize each 5.0 callback function individually using the gsi_initialize_callbacks() API function. This method for initializing the callback functions is not recommended.

3 Write the function.

Note that you can change the name of the callback function to something else. For example, you can use a function named *my_error_handler()* using the following syntax:

```
extern declare_gsi_error_handler(my_error_handler);
static gsi_install_error_handler(my_error_handler);
```

Using Stub Versions of G2 Gateway 5.0 Callbacks

Unlike GSI 4.1 callback functions, you do not need a stub for the 5.0 callbacks. There is no action that you must take for the G2 Gateway callback functions that you do not intend to use.

Calling Other Functions from Callbacks

Your callback functions can make calls to other functions, including:

- **G2 Gateway API functions**, which are defined in the G2 Gateway object libraries provided by Gensym. You make calls to these built-in functions inside your user code functions to perform various GSI-related tasks, such as returning data to G2.

  Your callback functions can call any API functions except *gsi_start()*, *gsi_run_loop()*, and *gsi_pause()*. These functions can be called only outside the extent of any callback function.

  For more information about specific G2 Gateway API functions, see API Functions.

- User-written functions in your G2 Gateway user code.
- API functions for accessing the external system to which your G2 Gateway bridge is connected or any third-party utility libraries that you need.
- **Remote procedure calls (RPCs)**, which are local functions in your G2 Gateway bridge user code that can be called from G2. You can also set up your bridge user code to call procedures found in your G2 knowledge base.

  For more information about RPCs, see Remote Procedure Calls.

Values Returned by Callback Functions

gsi_get_tcp_port() and *gsi_initialize_context()* are the only G2 Gateway callback functions that can return a value. The return value of all other callback functions is declared *void*.

Groups of Functionally Related Callback Functions

Callback functions fall into several functionally related groups, as described in the following sections.

Application Initialization

The following functions support the initial setup of a G2 Gateway application:

gsi_get_tcp_port()
gsi_set_up()

Both of these callbacks are executed by the API function *gsi_start()*. For a general discussion about how to use these functions, see Using gsi_start().

Connection Management

G2 Gateway calls the following functions when connections are activated or deactivated, or when a connected G2 process is paused or resumed:

gsi_initialize_context()
gsi_pause_context()
gsi_resume_context()
gsi_shutdown_context()

For information about how to use these functions, see Managing a Connection between G2 and a G2 Gateway Bridge.

Flow Control

G2 Gateway invokes the following functions when there is change in its ability to read or write on an open connection, or when a file descriptor is opened or closed:

gsi_read_callback()
gsi_write_callback()
gsi_open_fd()
gsi_close_fd()

Item Registration and Deregistration

G2 Gateway calls the following functions to register and deregister items with the G2 Gateway bridge process:

```
gsi_receive_registration()
gsi_receive_deregistrations()
```

For information about how G2 Gateway registers items, see Registering and Deregistering Items.

Data Service

The following functions support data-service for GSI variables in G2:

```
gsi_get_data()          gsi_set_data()
gsi_g2_poll()
```

For a general discussion about how to use these functions, see Implementing Data Service in G2 Gateway.

Error Handling

G2 Gateway calls the following functions to handle errors on active contexts, or to respond to a remote procedure call to a local function that has not been declared in the G2 Gateway user code:

```
gsi_error_handler()        gsi_missing_procedure_handler()
```

For information about how to signal errors for your G2 Gateway code and write customized error-handling procedures, see Error Handling.

Message Passing

G2 Gateway calls the following function to receive text messages that G2 sends using the G2 inform action:

```
gsi_receive_message()
```

For a general discussion of message passing, see Message Passing.

Run State Change

G2 Gateway calls the following callback whenever the flow of control enters or leaves G2 Gateway:

gsi_run_state_change()

Standard Callbacks

This section presents basic reference information about each of the standard named callback functions. The functions are presented in alphabetical order.

gsi_close_fd

Called whenever a file descriptor for network I/O is closed.

Synopsis

void gsi_close_fd(gsi_int descriptor)

Argument	Description
descriptor	The file descriptor.

Description

gsi_close_fd() is called whenever a file descriptor for network I/O is closed.

gsi_open_fd() and *gsi_close_fd()* enable a bridge to wait until there is network activity to which it must respond. Use of these callbacks, rather than the API function *gsi_pause()*, is recommended if the bridge is handling non-blocking I/O outside of the control of G2 Gateway. In this case, a mask containing a 1 bit for each of the currently open file descriptors should be maintained, using *gsi_open_fd()* and *gsi_close_fd()* as well as the non-G2 Gateway file descriptors through which the bridge is performing the non-blocking I/O. This mask, and possibly also a timeout, should be passed to the select() system function, to perform the waiting.

Before you can use the *gsi_close_fd()* callback, you must declare and install it. For instructions, see Using G2 Gateway 5.0 Callbacks with G2 Linked Statically or Dynamically.

gsi_error_handler

Called when errors occur in active contexts.

Synopsis

void gsi_error_handler(context, code, message)

Argument	Description
gsi_int context	The context in which the error occurs.
gsi_int code	The error code associated with the error.
*gsi_char *message*	The message associated with the error.

Description

The *gsi_error_handler()* callback is invoked when an error occurs in any active *context*. It receives argument values specifying the context in which the error occurred, and the error code and message associated with that error.

You can complete the code of *gsi_error_handler()* to handle the errors in the way that your application requires.

Before you can use the *gsi_error_handler()* callback, you must declare and install it. For instructions, see Using G2 Gateway 5.0 Callbacks with G2 Linked Statically or Dynamically.

gsi_g2_poll

Seek data or perform other user-specified actions, repeatedly.

Synopsis

```
void gsi_g2_poll()
```

Description

Use *gsi_g2_poll()* to perform any task that must be done repeatedly, such as polling an external system for data. A typical implementation of *gsi_g2_poll()* checks which registered GSI variables need updating, retrieves any data values available, packages the values into data structures, and sends these data structures to G2 via one of these API functions:

- *gsi_return_values()*
- *gsi_return_timed_values()*
- *gsi_return_attrs()*
- *gsi_return_timed_attrs()*

G2 Gateway calls *gsi_g2_poll()* if the poll-external-system-for-data attribute of the GSI interface is set to yes.

How often G2 Gateway calls *gsi_g2_poll()* is determined by the setting of the scheduler-mode attribute of the Timing Parameters system table in the current KB of the connected G2 process:

- If scheduler-mode is set to as fast as possible, G2 Gateway calls *gsi_g2_poll()* as often as possible.
- If scheduler-mode is set to real time, G2 Gateway calls *gsi_g2_poll()* approximately once per second.

  The real time interval can vary, depending on the processing load of G2, and on the value assigned to the priority-of-data-service attribute, which is found under Data Server Parameters in the system tables of your KB.

Caution If you have specified a low priority in priority-of-data-service and your G2 application has a heavy processing load, *gsi_g2_poll()* may never be called.

- If scheduler-mode is set to simulated time, G2 calls *gsi_g2_poll()* at the end of each scheduler tick.

If any tasks performed through *gsi_g2_poll()* take longer than 1 second, G2 Gateway queues the accumulated *gsi_g2_poll()* calls for future execution.

gsi_g2_poll() is stopped automatically when the G2 KB to which it is connected is paused, and it resumes when the G2 KB is resumed.

For more information about the G2 system tables, see the *G2 Reference Manual*.

Example

The following *gsi_g2_poll()* callback updates the value of the data-value attribute of a G2 object if the value for that attribute in the bridge has changed since the last time when *gsi_g2_poll()* was executed.

Note This attribute is named "*DATA-VALUE*" in G2 Gateway.

A *gsi_g2_poll()* callback has no arguments through which it can receive existing data structures. For this reason, *gsi_g2_poll()* must allocate any data structures that it needs in order to send values back to G2. This *gsi_g2_poll()* callback allocates the following data structures:

- A *gsi_registered_item*, to which *gsi_g2_poll()* assigns the handle of an object in G2.
- A *gsi_attr* with an item.

  The item allocated with this *gsi_attr* is used by this *gsi_g2_poll()* to carry a value back to the data-value attribute of the object in G2.

This example assumes that the G2 Gateway user code includes the following global variables:

- *my_stored_object_handle*, which contains the handle of the object in G2. This variable can receive the handle either within the *gsi_receive_registrations()* callback, or from a user-defined G2 Gateway function. The *gsi_g2_poll()* callback assigns the handle in *my_stored_object_handle* to the *gsi_registered_item* that it allocates.
- *new_object_value*, in which the user code stores the current value for the object. The user code obtains values for this variable from the external system.
- *old_object_value*, in which the user code saves for future use the value with which it updates the data-value attribute of the object in G2.

```
gsi_int new_object_value = 0;
gsi_int old_object_value = 0;
gsi_int my_stored_object_handle = 0;

. . .

/* gsi_g2_poll() callback function */

void gsi_g2_poll()
```

```
{
   gsi_registered_item *object;
   gsi_attr *ret_attr;

/* Include code here that gets a new value from external
   system and stores it in new_object_value. If the new
   value is equal to the old value, this callback returns
   without updating the attribute. */

if(new_object_value == old_object_value)
      return;

/* If the new attribute value is not equal to the old
   value, update the attribute in the G2 object with the
   new value. */

/* Allocate memory for the local object and attribute */

   object = gsi_make_registered_items(1);
   ret_attr = gsi_make_attrs_with_items(1);

/* Set the handle of the local object to the handle of
   the G2 object to which the value will be returned. */

   gsi_set_handle(object[0], my_stored_object_handle);

/* Set the object_index attribute name and value.
   Note: Must enable object status return (if it is
   disabled)in order to get the index back to the object */

   gsi_set_attr_name(ret_attr[0], "DATA-VALUE");
   gsi_set_int(ret_attr[0], new_object_value);

   gsi_return_attrs(object[0], ret_attr, 1,
      gsi_current_context());

/* Release the allocated memory */

   gsi_reclaim_registered_items(object);
   gsi_reclaim_attrs_with_items(ret_attr);

   return;
} /* End of gsi_g2_poll() */
```

gsi_get_data

Respond to a request from G2 for a value for a data-served GSI variable.

Synopsis

void gsi_get_data(registered_items,count)

Argument	Description
*gsi_registered _item *registered_items*	An array of one or more registered items. G2 Gateway creates a *gsi_registered_item* structure each time G2 asks G2 Gateway to get a new value for a GSI variable from an external system, or to write the value of a GSI variable to a data point in an external system. The structure stores the handle, current attribute values, and other information associated with the registered item. For information about the components of *gsi_registered_item* structures and the API functions that access these components, see gsi_registered_item Data Structures.
gsi_int count	Number of registered items.

Description

G2 Gateway calls *gsi_get_data()* when it receives a request from G2 for a value for one or more data-served GSI variables. For information about the events that cause G2 to request G2 Gateway for a value for a GSI variable, see Returning Solicited Data to G2.

G2 Gateway may also call *gsi_get_data()* repeatedly, at regular intervals, depending on the settings of the retry-interval-after-timeout and timeout-for-variables attributes of the G2 Inference Engine Parameters system table. If the retry-interval-after-timeout attribute has a time interval value, G2 requests G2 Gateway for values for GSI variables immediately when the variables exceed the time interval specified in timeout-for-variables, and repeatedly thereafter at the interval specified in retry-interval-after-timeout. Setting retry-interval-after-timeout to do-not-retry prevents G2 from retrying a variable. Note that these attributes affect all variables in a KB. For more information about system engine parameters, see the *G2 Reference Manual*.

To return the data to G2, include a call inside *gsi_get_data()* to one of the following G2 Gateway API functions:

gsi_return_values()

gsi_return_timed_values()

gsi_return_attrs()

gsi_return_timed_attrs()

If you use *gsi_return_values()*, it can be convenient to pass to it the array pointed to by *registered_items*, because this array is already allocated, is of the correct size, and has handles already assigned.

Note If the G2 value of a quantitative variable is none when G2 Gateway makes a call to *gsi_get_data()*, G2 sets the data type to *gsi_null_tag*. This is an indication that there is no valid value available for the variable.

See also the discussion of G2 Gateway data service in the *G2 Reference Manual*.

Example

The following *gsi_get_data()* callback updates the value of the data-value attribute of a G2 object. This example assumes that:

- The G2 Gateway user code obtains a new value for the data-value attribute from an external system, and stores this value in a global variable named *new_object_value*.
- The item passed to this function represents a G2 object that was passed to the G2 Gateway bridge through a remote procedure call, using the as handle grammar. Thus, the object's handle value was sent to the bridge, but not its attributes.
- The G2 object has an attribute named data-value. This callback passes the new value to this attribute.

To return a value to the data-value attribute, this *gsi_get_data()* callback does the following:

1. Allocates a *gsi_registered_item* structure and assigns to it the handle of the G2 object. The handle value is the only meaningful information that this structure carries.
2. Allocates a *gsi_attr* structure and assigns to it the new value for the data-value attribute of the G2 object.
3. Calls the API function *gsi_return_attrs()* to return the new attribute value to the G2 object designated by the handle value of the *gsi_registered_item* structure.

4 Deallocates the *gsi_registered_item* and *gsi_attr* structures.

```
/* gsi_get_data() callback function */

void gsi_get_data(registered_items, count)
   gsi_registered_item *registered_items;
   gsi_int count;
{
   gsi_registered_item *object;
   gsi_attr *ret_attr;
   gsi_int n;

/* Allocate memory for the local object and attribute */

   object = gsi_make_registered_items(1);
   ret_attr = gsi_make_attrs_with_items(1);

/* Loop through registered items sent to this function. */

   for(n=0; n<count; n++)
   {
/* Set the handle of the local object to the handle of
   the G2 object to which the value will be returned */

   gsi_set_handle(object[0],
      gsi_handle_of(registered_items[n]));

/* Set the object_index attribute name and value.
   Note: Must enable object status return (if it is
   disabled) to get the index back to the object */

   gsi_set_attr_name(ret_attr[0], "DATA-VALUE");
   gsi_set_int(ret_attr[0], new_object_value);

   gsi_return_attrs(object[0], ret_attr, 1,
      gsi_current_context());
   } /* End of for loop */

   /* Release the allocated memory */
   gsi_reclaim_registered_items(object);
   gsi_reclaim_attrs_with_items(ret_attr);

   return;

} /* End of gsi_get_data() */
```

gsi_get_tcp_port

Specify a default port number that a G2 Gateway process using TCP/IP can listen on for connections from a G2 process.

Synopsis

```
gsi_int gsi_get_tcp_port()
```

Return Value	Description
gsi_int	The TCP/IP port number that G2 Gateway will listen on for connections from a G2 process.

Description

G2 Gateway calls *gsi_get_tcp_port()* at startup if no port number, or a port number of 0, is specified on the command line that invokes the G2 Gateway bridge process. *gsi_get_tcp_port()* specifies a default port number for the G2 Gateway process to listen on.

Use a *return* statement inside *gsi_get_tcp_port()* to return the value that you want to specify as the default port number. If *gsi_get_tcp_port()* returns a value of 0, G2 Gateway by default uses port number 22041, if it is available. If 22041 is not available, G2 Gateway uses the first available port number within the next 99 addresses.

If you use the *-tcpipexact* option when you activate a G2 Gateway interface, the process will exit if the port is not available. For more information, see the description of *tcpipexact*.

Note If TCP/IP is not installed, *gsi_get_tcp_port()* has no effect.

Examples

The following *gsi_get_tcp_port()* callback returns the value 0, which causes G2 Gateway to use the default port number, 22041, or the port number at the first available address:

```
#define TCPIP_PORT_NUMBER  0

...

/* gsi_get_tcp_port() callback function */
```

```
gsi_int gsi_get_tcp_port()
{
   return(TCPIP_PORT_NUMBER);
} /* End of gsi_get_tcp_port() */
```

The following *gsi_get_tcp_port()* callback returns the port number 4000 if the variable *my_machine_name* contains the string "*DEVELOPMENT*". Otherwise, it returns the port number 4001.

```
/* gsi_get_tcp_port() callback function */

gsi_int gsi_get_tcp_port()
{
   gsi_int  tcpip_starting_port_number =
      TCPIP_PORT_NUMBER;

   if(!strcmp(my_machine_name,"DEVELOPMENT"))
      tcpip_starting_port_number = 4000;
   else
      tcpip_starting_port_number = 4001;

   return(tcpip_starting_port_number);
} /*End of gsi_get_tcp_port() */
```

gsi_initialize_context

Initialize a connection between a GSI interface in G2 and G2 Gateway, or reject the connection.

Synopsis

gsi_int gsi_initialize_context(rpis, length)

Argument	Description
*gsi_char *rpis*	Points to the string specified for the remote-process-initialization-string attribute of the GSI interface that configures the connection between this G2 Gateway process and a G2 process. The string pointed to by the *rpis* argument can be any user-specified string that enables *gsi_initialize_context*() to perform tasks such as initializing data structures, opening a file for reading, starting a device, turning on debugging, obtaining a database user login and password, and opening communication with the external system. For information about how to specify the remote-process-initialization-string attribute of a GSI interface, see Remote-Process-Initialization-String Attribute.
gsi_int length	Number of characters (not including the null terminator character) in *rpis*.

Return Value	Description
gsi_int	Must be *GSI_ACCEPT* or *GSI_REJECT*. This value indicates your user code's acceptance or rejection, respectively, of the new connection. If this function returns *GSI_ACCEPT*, the connection from G2 is accepted. G2 sets the gsi-interface-status attribute of the GSI interface to 2 (the connection is active). If this function returns *GSI_REJECT*, the connection from G2 is rejected. G2 sets gsi-interface-status to –2 (the connection failed). G2 Gateway assumes that the new connection has been rejected by the user code, and immediately shuts down the just-opened context. If you are running in continuous mode, *gsi_run_loop()* processing continues. If you are running in one-cycle mode, *gsi_run_loop()* exits.

Description

G2 Gateway calls *gsi_initialize_context()* per context each time you activate a GSI interface that specifies the machine name and port number on which this G2 Gateway application is listening.

You can use this callback to perform tasks such as:

- Validating connections from G2, as for a login procedure.
- Allocating and/or initializing global tables on a per-connection basis; that is, tables which are unique to this connection.
- Declaring G2 procedures as remote procedures, so that your G2 Gateway bridge process can invoke them. These remote procedure declarations are valid only for the context through which the G2 process is connected to the G2 Gateway bridge.

An additional feature of this function is that it allows remote procedure calls and message service after making a connection but before returning a value. This is true regardless of whether *gsi_initialize_context()* eventually returns *GSI_ACCEPT* or *GSI_REJECT*.

Note To enable G2 to access local C functions in your bridge user code through remote function calls, declare these C functions in *gsi_set_up()*, using the callback function *gsi_rpc_declare_local()*.

You can use the string pointed to by the *rpis* argument to perform tasks such as initializing data structures, opening a file for reading, starting a device, turning on debugging, passing a database user login and password, or opening communication with the external system.

Note *gsi_initialize_context()* and *gsi_get_tcp_port()* are the only two G2 Gateway callback functions that return a value.

Examples

The following *gsi_initialize_context()* callback declares a G2 procedure named g2-custom-proc() as a remote procedure that can be invoked by your G2 Gateway user code within any active context:

```
gsi_int context_procedure_hdl[50];

. . .

/* gsi_initialize_context() callback function */

gsi_int gsi_initialize_context(remote_process_init_string,
   length)
   gsi_char *remote_process_init_string;
   gsi_int length;
{
   gsi_int context_number;
   context_number = gsi_current_context();

/* Call a user defined function here to perform context
   specific initializations, such as allocating memory,
   connecting to external systems, checking system
   resources, managing logfiles, and so on. */

/* Declare any remote procedure calls here (GSI to g2).
   Because remote procedure declarations are per
   context, they cannot appear in gsi_set_up(). */

/* Declare G2-CUSTOM-PROC() as a remote procedure
   that your G2 Gateway bridge can invoke. */
```

```
gsi_rpc_declare_remote(&context_procedure_hdl
   [context_number],"G2-CUSTOM-PROC",
   NULL,3,0,context_number);

/* The declaration above is available on every active
   context. Thus, 'G2-CUSTOM-PROC' can be called from
   within any active context. */

   return(GSI_ACCEPT);
} /* End of gsi_initialize_context() */
```

The following *gsi_initialize_context()* callback opens a file whose filename is specified by the remote-process-initialization-string attribute of the GSI interface that configures this connection:

```
#define MAX_CONTEXTS 50
#define MAX_FNAME 25
char file_name[MAX_CONTEXTS][MAX_FNAME];
gsi_int file_status[MAX_CONTEXTS];
FILE *file_id[MAX_CONTEXTS] = {NULL};

. . .

/* gsi_initialize_context() callback function */

gsi_int gsi_initialize_context(rpis, length)
   gsi_char *rpis;
   gsi_int length;
{
   gsi_char *msg_ptr;
   gsi_int i;
   gsi_int status;
```

```
/*
*  Open a file for reading specified by G2.
*/
   strcpy(file_name[current_context], rpis);
   printf("\nInitializing data server, file = %s for
      #%hd\n", rpis, current_context);
   file_id[current_context] = fopen(rpis, "r");
   printf("\nFile open state = %ld\n", (gsi_int)
      file_id[current_context]);
   if(file_id[current_context] == NULL) {
      printf("\nCannot open file %s in gsi_init().\n",
         rpis);
      file_status[current_context] = CLOSED;
      return(GSI_REJECT); }
   else
      file_status[current_context] = OPEN;

return(GSI_ACCEPT);

} /* End of gsi_initialize_context() */
```

gsi_missing_procedure_handler

Called whenever G2 makes a call to a G2 Gateway local function that has not been declared in the bridge by a call to *gsi_rpc_declare_local()*.

Synopsis

void gsi_missing_procedure_handler(name)

Argument	Description
*gsi_char * name*	The name of your missing procedure handler function.

Description

gsi_missing_procedure_handler() enables your G2 Gateway user code to designate a handler callback that is called whenever G2 makes a remote procedure call to a G2 Gateway function that has not been declared by a call to *gsi_rpc_declare_local()*.

You can include a call to *gsi_rpc_declare_local()* within *gsi_missing_procedure_handler()* to declare the undeclared local function. G2 Gateway then invokes the local function in response to the call from G2.

Before you can use *gsi_missing_procedure_handler()*, you must declare and install it. For instructions, see Using G2 Gateway 5.0 Callbacks with G2 Linked Statically or Dynamically.

gsi_not_writing_fd

Called whenever a specified file descriptor stops being in use by G2 Gateway for writing.

Synopsis

void gsi_not_writing_fd (gsi_int ***file_descriptor****)*

Argument	Description
gsi_int file_descriptor	A UNIX file descriptor, usually referring to a network socket (or possibly a pipe), on which input or output can occur asynchronously.

Related Functions

Function	Description
gsi_open_fd()	Called whenever a specified file descriptor is opened for network I/O.
gsi_close_fd()	Called whenever a file descriptor for network I/O is closed.
gsi_writing_fd()	Called whenever a specified file descriptor is in use by G2 Gateway for writing.
gsi_watch_fd()	Specifies a file descriptor that G2 Gateway watches for network read or error activity.
gsi_unwatch_fd()	Causes *gsi_run_loop()* to not wake up when input or output takes place on a file descriptor.
gsi_watch_fd_for_writing()	Starts G2 Gateway watching for write activity on a file descriptor.
gsi_unwatch_fd_for_writing	Stops G2 Gateway from watching for write activity on a file descriptor.
gsi_pause()	Causes the bridge process to sleep for 1 second by default, or until a network event occurs on a network connection to the bridge process.

gsi_open_fd

Called whenever a specified file descriptor is opened for network I/O.

Synopsis

void gsi_open_fd(gsi_int descriptor)

Argument	Description
descriptor	The file descriptor.

Description

gsi_open_fd() and the callback *gsi_close_fd()* enable a bridge to wait until there is network activity to which it must respond. Use of these callbacks, rather than the API function *gsi_pause()*, is recommended if the bridge is handling non-blocking I/O outside of the control of G2 Gateway. In this case, a mask containing a 1 bit for each of the currently open file descriptors should be maintained, using *gsi_open_fd()* and *gsi_close_fd()* as well as the non-G2 Gateway file descriptors through which the bridge is performing the non-blocking I/O. This mask, and possibly also a timeout, should be passed to the select() system function, to perform the waiting.

Before you can use the *gsi_open_fd()* callback, you must declare and install it. For instructions, see Using G2 Gateway 5.0 Callbacks with G2 Linked Statically or Dynamically.

gsi_pause_context

Called by G2 Gateway whenever any G2 process that is connected to this G2 Gateway process pauses its current knowledge base (KB).

Synopsis

```
void gsi_pause_context()
```

Description

gsi_pause_context() is useful for pausing any functions in your G2 Gateway bridge process that operate in cooperation with G2. You can use *gsi_pause_context()* to suspend these functions until G2 resumes operation.

For example, you can use *gsi_pause_context()* to halt unsolicited data collection from a queue in the external system, record an event in a log file, or stop the G2 Gateway watchdog timer invoked through the API function *gsi_watchdog()*. If an external system is sending data to the G2 Gateway bridge asynchronously to G2, *gsi_pause_context()* can arrange for the external system to stop sending the data.

Your code in *gsi_pause_context()* should perform the reverse of your code in *gsi_resume_context()*, which you can use to reenable G2 Gateway operations that require a running KB.

Unsolicited reporting through *gsi_g2_poll()* works in cooperation with G2, which means that calls to *gsi_g2_poll()* are stopped when the G2 process pauses its current KB.

Related Procedures

Procedure	Description
gsi_current_context()	Returns the context number of the current context.

Example

The following *gsi_pause_context()* callback illustrates where you can place statements that suspend operations performed asynchronously to G2, tell an external system that the G2 Gateway bridge process is paused, or perform any other appropriate operations as required by your application.

```
/* gsi_pause_context() callback function */

void gsi_pause_context()
{
   printf("gsi_pause_context in context %d\n",
      gsi_current_context());

/* Suspend operations here. */

} /* End of gsi_pause_context() */
```

gsi_read_callback

Called by G2 Gateway when there is a change in the state of G2 Gateway's ability to read data from G2 on an open connection.

Synopsis

void gsi_read_callback(context, state)

Argument	Description
gsi_int context	The number of the context through which G2 Gateway is trying to read data from G2.
gsi_int state	*GSI_IO_BLOCKED* (there is no data to read) or *GSI_IO_UNBLOCKED* (there is data to read) These statuses are defined in *gsi_main.h*.

Description

gsi_read_callback() is called by G2 Gateway when there is a change in the state of G2 Gateway's ability to read data from G2. That is, G2 Gateway invokes this callback when:

- It becomes impossible for G2 Gateway to read data, after a period when it was possible; for example, when G2 Gateway tries to read data from G2 but there is no longer any data to read. In this case, G2 Gateway calls *gsi_read_callback()*, passes the number of the current context to *context*, and sets *state* to *GSI_IO_BLOCKED*.
- It becomes possible for G2 Gateway to read data, after a period when it was not possible; for example, when there is once again data in G2 that the bridge can read. In this case, G2 Gateway calls *gsi_read_callback()*, passes the number of the current context to *context*, and sets *state* to *GSI_IO_UNBLOCKED*.

You can use *gsi_read_callback()* for purposes such as accumulating information about the flow of data, or to set indicators on a control panel.

Note Before you can use the *gsi_read_callback()* callback, you must declare and install it. For instructions, see Using G2 Gateway 5.0 Callbacks with G2 Linked Statically or Dynamically.

gsi_receive_deregistrations

Deregister a single registered variable.

Synopsis

void gsi_receive_deregistrations(registered_items, count)

Argument	Description
*gsi_registered_item *registered_items*	An array of one or more registered items, each of whose handle indicates a G2 variable whose G2 Gateway data service was cancelled in some way. For information about the components of *gsi_registered_item*structures and the API functions that access these components, see gsi_registered_item Data Structures.
gsi_int count	Number of elements in the array pointed to by *registered_items* .

Description

G2 Gateway calls *gsi_receive_deregistrations()*:

- Before a GSI interface is shut down. G2 Gateway calls *gsi_shutdown_context()*immediately after it calls *gsi_receive_deregistrations()*.
- When a GSI variable is disabled, or deleted, or before a GSI variable is redefined by a change to its data type or one of its identifying attributes.

A common use of this function is to free handles and any memory that was allocated for variables when they were defined, and if the external system has a scheduler, to notify the external system that the variables are no longer receiving data.

The items involved in your *gsi_receive_deregistrations()* code must not conflict with a subsequent *gsi_get_data()* call, or the reporting of unsolicited (but unscheduled) values. *gsi_receive_deregistrations()* must free any memory that was dynamically allocated for a particular item by *gsi_receive_registration()*.

For information about how G2 Gateway registers and deregisters items, see Registering and Deregistering Items.

Example

The following *gsi_receive_deregistrations()* callback deregisters G2 objects, identifying each object by its handle.

```
void gsi_receive_deregistrations(registered_items, count)
    gsi_registered_item  registered_items[];
    gsi_int count;
{
    gsi_int obj_handle;

    /* For the G2 object passed to this function */

    obj_handle = gsi_handle_of(registered_items[0]);

    /* Make sure that the object handle is valid */

    if (gsi_registration_of(obj_handle, gsi_current_context()) ==
      NULL) {
   printf("Bad handle %d\n", obj_handle);
    }
    else {

    /* For this object perform object-specific deinitializations,
       such as deallocating memory and data structures, managing
       logfiles, disconnecting the object from an external system
       or data point, and so on. These deallocations are commonly
       the reverse of any allocation performed in the
       gsi_receive_registration() callback. */

    gsi_show_registered_items("gsi_receive_deregistrations",
             registered_items, count);
    }

   /* End of gsi_receive_deregistrations() */
}
```

gsi_receive_message

Receive a message from G2, sent as a result of a G2 inform action on a GSI message server.

Synopsis

void gsi_receive_message(message, length)

Argument	Description
*gsi_char *message*	A pointer to the text string passed to the G2 Gateway bridge from G2 as a result of an inform action.
gsi_int length	Number of characters (not including the null terminator) in *message*.

Description

G2 Gateway calls *gsi_receive_message()* whenever a G2 KB executes an inform action on an item that is a GSI message server. For information about how to use GSI message servers to pass messages from G2 to G2 Gateway, see Message Passing.

gsi_receive_message() can pass the text that it receives from G2 to any external system with which this G2 Gateway bridge is communicating.

The maximum number of characters that G2 can send in a string to *gsi_receive_message()* is 999,9999, including the null terminator.

Your *gsi_receive_message()* function must be coded to accept a C string of characters that are encoded in the Gensym character set. The Gensym character set can encode characters not found in the standard ASCII character set. For information about the Gensym character set, see the *G2 Reference Manual*.

Example

The following *gsi_receive_message()* callback receives a message and returns the message to G2, through a remote procedure call to a G2 procedure. This example assumes that your user code does the following:

- Declares a G2 procedure as a remote function, by including the following call in *gsi_initialize_context()*:

```
gsi_rpc_declare_remote
  (&context_procedure_hdl[gsi_current_context()],
   G2_CUSTOM_PROC,NULL,3,0,context_number);
```

- Allocates the following variable to hold the current context number:

```
gsi_int context_procedure_hdl[50];

/* gsi_receive_message() callback function */

void gsi_receive_message(message, length)
   gsi_char *message;
   gsi_int length;
{
   gsi_item *return_args;
   char temp_buff[128];

/* Receive a message from G2, manipulate it and place into
an item array for return as an RPC argument back to G2. */

/* Modify the original message */

   sprintf(temp_buff,"%s %s","The message I received
      is:",message);

/* Allocate three items to hold the return values */
   return_args = gsi_make_items(3);

/* arg0 = the modified message
   arg1 = the new length of the message
   arg2 = a symbol identifying the message */

/* Load three values into the return item structure */
   gsi_set_str(return_args[0], temp_buff);
   gsi_set_int(return_args[1], strlen(temp_buff));
   gsi_set_sym(return_args[2], "GENSYM-MESSAGE");

/* Send the values back to the G2 procedure identified by
   context_procedure_hdl[gsi_current_context()] */

gsi_rpc_start(context_procedure_hdl
   [gsi_current_context()],return_args,
   gsi_current_context());

   gsi_reclaim_items(return_args);

} /* End of gsi_receive_message() */
```

gsi_receive_registration

Called when G2 registers an item with G2 Gateway.

Synopsis

void gsi_receive_registration(registration)

Argument	Description
gsi_registration registration	Specifies the data type of the variable being registered, the default update interval, and the six identifying attributes.

Description

G2 Gateway calls *gsi_receive_registration()* when:

- G2 tries to map a GSI variable to a data point in an external system. This happens the first time when G2 requests G2 Gateway to read from or write to the GSI variable, or immediately after any GSI variable's data type or identifying attributes are modified.
- G2 calls a local function in G2 Gateway and passes a handle for a G2 item that has not previously been registered. *gsi_receive_registration()* is called *before* the local function is invoked in G2 Gateway.

You can use *gsi_receive_registration()* to perform tasks such as initializing the external data point, allocating memory, or returning the variable's network handle to an attribute of the variable for some future use.

For more information about how items are registered, see Item Passing.

See also the description of the G2 system procedure g2-register-on-network() in the *G2 System Procedures Reference Manual*.

Example

The *gsi_receive_registration()* callback shown below does the following:

1. Stores the handle of a *gsi_registered_item* in a variable named *obj_handle*.
2. Allocates a *gsi_registered_item* named *object*, and an attribute named *ret_attr*. The callback uses these structures to return a value to the registered G2 object.

3 Assigns the handle stored in *obj_handle* to the *handle* component of the newly allocated *gsi_registered_item* structure named *object*. The *object* structure now has the same handle as the registered G2 object.

4 Assigns the name *OBJECT-HANDLE* to the *name* component of the structure *ret_attr*. This example assumes that the registered G2 object has an attribute named object-handle.

5 Assigns the value of *obj_handle* to the *ret_attr* structure.

6 Calls *gsi_return_attrs()* to return the value in the *ret_attr* structure to the object-handle attribute of the registered G2 object.

```
/* gsi_receive_registration() callback function */

void gsi_receive_registration(item_registration)
   gsi_registration item_registration;
{
   gsi_registered_item *object;
   gsi_attr *ret_attr;
   gsi_int obj_handle;

   /* Get the item handle of this item */
   obj_handle = gsi_handle_of(item_registration);

/* Store this object handle for later use */
   my_stored_object_handle = obj_handle;

/* Allocate memory for the local object and attribute */
   object = gsi_make_registered_items(1);
   ret_attr = gsi_make_attrs_with_items(1);

/* Set the handle of the local object to the handle of
   the item_registration object */
   gsi_set_handle(object[0], obj_handle);

/* For this object, identified by the object handle, perform
object-specific initializations, such as allocating memory and
data structures, managing logfiles, connecting the object to an
external system or data point, and so on. These operations
_deregistrations(). */

/* Initialize the object here */

/* Assume that there is an attribute of the
   item_registration object called  'OBJECT-HANDLE'. Set
   the OBJECT-HANDLE attribute name and value of the
   object */
   gsi_set_attr_name(ret_attr[0], "OBJECT-HANDLE");
```

```
   gsi_set_int(ret_attr[0], obj_handle);
   gsi_return_attrs(object[0], ret_attr, 1,
      gsi_current_context());

/* Release the allocated memory */
   gsi_reclaim_registered_items(object);
   gsi_reclaim_attrs_with_items(ret_attr);

   return;

}   /* End of gsi_receive_registration() *
```

gsi_reset_context

Called by G2 Gateway whenever any G2 process that is connected to this G2 process through a permanent connection resets its current knowledge base (KB).

Synopsis

```
void gsi_reset_context()
```

Description

Use this callback to respond as needed whenever G2 is reset. Clearing a KB first resets it, calling gsi_reset_context(). Restarting a KB first resets and then starts it, calling first gsi_reset_context() and then gsi_start_context().

Note This callback is applicable only to permanent connections.

gsi_resume_context

Called by G2 Gateway when a connected G2 process resumes its current KB.

Synopsis

```
void gsi_resume_context()
```

Description

G2 Gateway calls *gsi_resume_context()* when a connected G2 process resumes running its current knowledge base (KB).

If your bridge has any asynchronous operations that you have suspended within your code for *gsi_pause_context()*, you can include code in *gsi_resume_context()* to resume these operations.

For example, you can use *gsi_resume_context()* to prepare the external system to access data, resume unsolicited data collection, record events in a log file, or inform a G2 operator that the application has resumed.

Note When the connected G2 process is paused, a variable can still be registered and deregistered.

Example

The following *gsi_resume_context()* callback prints a message saying that the context has been resumed, and calls a user-defined procedure, *check_for_queued_messages()*, that checks for messages received by the context while it was paused.

```
/* gsi_resume_context() callback function */

void gsi_resume_context()
{
   printf("gsi_resume_context in context %d\n",
      gsi_current_context());

/* No longer idle. Service events that occured
   while context was paused. */
   check_for_queued_messages();
} /* End of gsi_resume_context() */
```

gsi_run_state_change

Called whenever the flow of control enters or leaves G2 Gateway.

Synopsis

```
void gsi_run_state_change(gsi_int direction, gsi_int type,
   char *name)
```

Argument	Description
direction	The direction in which the flow of control changed. The possible values are: *GSI_RUN_STATE_DIRECTION_ENTERING_GSI* *GSI_RUN_STATE_DIRECTION_LEAVING_GSI* *GSI_RUN_STATE_ENTERING_WATCHDOG* *GSI_RUN_STATE_LEAVING_WATCHDOG* *GSI_RUN_STATE_DIRECTION_ENTERING_GSI_BY_SIGNAL*
type	The kind of state change that occurred. The possible values are: *GSI_RUN_STATE_TYPE_API* *GSI_RUN_STATE_TYPE_CALLBACK* *GSI_RUN_STATE_TYPE_WAIT* *GSI_RUN_STATE_TYPE_SIGNAL*
name	The name of the API function or callback the G2 Gateway entered or left, causing the run state change.

Description

The direction codes *GSI_RUN_STATE_DIRECTION_ENTERING_GSI* and *GSI_RUN_STATE_DIRECTION_LEAVING_GSI* are used with the state change types *GSI_RUN_STATE_TYPE_API*, *GSI_RUN_STATE_TYPE_CALLBACK*, and *GSI_RUN_STATE_TYPE_WAIT*.

The direction codes *GSI_RUN_STATE_ENTERING_WATCHDOG*, *GSI_RUN_STATE_LEAVING_WATCHDOG*, and *GSI_RUN_STATE_DIRECTION_ENTERING_GSI_BY_SIGNAL* are used with the state change type *GSI_RUN_STATE_TYPE_SIGNAL*.

Note Using the *gsi_run_state_change* callback to manage a lock is prone to hangs. Use the *GSI_PROTECT_INNER_CALLS* runtime option to prevent hangs in applications that use the *gsi_run_state_change()* callback

Note Before you can use *gsi_run_state_change()*, you must declare and install it. For instructions, see Using G2 Gateway 5.0 Callbacks with G2 Linked Statically or Dynamically

gsi_set_data

Called when G2 executes one or more set actions on a data served GSI variable.

Synopsis

void gsi_set_data(registered_items, count)

Argument	Description
*gsi_registered_item *registered_items*	Array of one or more registered items. Each item has a handle that indicates a G2 variable with G2 Gateway data service requesting to set a value to the external data point to which it is mapped. The type and value of each registered item can be referenced to set the external data point.
gsi_int count	Number of registered items referenced in the *registered_items* argument.

Description

The G2 set action sends a request to G2 Gateway to set the value of an external data point that is mapped to a GSI variable in the G2 KB.

The set action does not change the value of the GSI variable in G2.

G2 sends handles to G2 Gateway for all requests for set operations, either through a rule or a procedure, that have occurred since the last clock tick. G2 Gateway packages the handles into *gsi_registration* structures and passes them to *gsi_set_data()*.

Implementations of *gsi_set_data()* frequently include one or more calls to G2 Gateway return functions, in order to echo the values being set in the external system back to the G2 process. In this way, you can arrange for the last-recorded-value attribute of a GSI variable to change only after its corresponding value in the external system has changed.

Example

The following *gsi_set_data*() callback:

- Receives an array of registered items from G2.
- Loops through the array, determining the data type of each item.
- Sets global variables to the values of the items whose types correspond to the types of the global variables.

This example assumes:

- That each registered item has one of the following handles, which indicate the data type of the item: *string_handle*, *symbol_handle*, *float64_handle*, *integer_handle*, or *logical_handle*.
- The G2 Gateway user code includes the global variables *string_dat*, *symbol_dat*, *float64_dat*, *integer_dat* and *logical_dat* to receive the values of the registered items.

```
/* gsi_set_data() callback function */

void gsi_set_data(registered_item_array, count)
   gsi_registered_item  *registered_item_array;
   gsi_int count;
{
   gsi_int i;

/* For each object sent by G2, set a global variable in
accordance with its type. */

   for(i=0; i<count; ++i) {

   if(gsi_handle_of(registered_item_array[i]) ==
      string_handle) {
      strcpy(string_dat,
         gsi_str_of(registered_item_array[i]));
      continue; }
   if(gsi_handle_of(registered_item_array[i]) ==
      symbol_handle) {
      strcpy(symbol_dat,
         gsi_sym_of(registered_item_array[i]));
      continue; }
   if(gsi_handle_of(registered_item_array[i]) ==
      float64_handle) {
      float64_dat = gsi_flt_of(registered_item_array[i]);
      continue; }
```

```
   if(gsi_handle_of(registered_item_array[i]) ==
      integer_handle) {
      integer_dat = gsi_int_of(registered_item_array[i]);
      continue; }
   if(gsi_handle_of(registered_item_array[i]) ==
      logical_handle) {
      logical_dat = gsi_log_of(registered_item_array[i]);
      continue; }

   } /* End of for loop */

} /* End of gsi_set_data() */
```

gsi_set_up

Perform G2 Gateway-related operations that need to be performed only once during the lifetime of the bridge process.

Synopsis

```
void gsi_set_up()
```

Description

gsi_set_up() is called by the API function *gsi_start()* before it calls any other G2 Gateway callback function.

The following operations can be performed in *gsi_set_up()*:

- Installing a custom error handler.
- Setting or resetting G2 Gateway run-time options.
- Declaring local functions that G2 can call as remote procedures.
- Allocating arrays of G2 Gateway structures.
- Initializing global variable.
- Executing any operation that requires a one-time startup when the bridge is started.

For an example of how to perform these operations, see Performing Once-Only Operations through gsi_set_up().

Note that remote declarations for G2 procedures (as opposed to local declarations for G2 Gateway functions) are placed in *gsi_initialize_context()*, rather than *gsi_set_up()*, because remote procedures are per context. *gsi_set_up()* is called once during the life of the G2 Gateway application process. *gsi_initialize_context()* is called once for each context created.

Example

The following *gsi_set_up()* callback installs an error handler, declares local functions that can be called by G2, and allocates G2 Gateway structures:

```
/* gsi_set_up() callback function */

void gsi_set_up()
{
   gsi_item  error_object;
   gsi_attr *error_object_attrs;
   gsi_attr *name_attr_ptr;
   gsi_attr  name_attr;

/* Install custom error handler. */

   gsi_install_error_handler(itemtest_error_handler);

/* Declare local functions called by G2. */

   gsi_rpc_declare_local(receive_item_or_value,
      "RECEIVE-AND-DISPLAY-ITEM");
   gsi_rpc_declare_local(receive_and_return_copy,
      "RECEIVE-AND-RETURN-ITEM-COPY");
   gsi_rpc_declare_local(receive_item_transfer,
      "RECEIVE-AND-DISPLAY-TRANSFER");
   gsi_rpc_declare_local(receive_request_for_copy,
      "RECEIVE-REQUEST-ITEM-COPY");

/*
* Allocate and set up context-independent global
* G2 Gateway structures.
*/
   error_object_ptr = gsi_make_items(1);
   error_object = *error_object_ptr;
   name_attr_ptr = gsi_make_attrs_with_items(1);
   name_attr = *name_attr_ptr;
   gsi_set_attr_name(name_attr,"NAME");
   gsi_set_sym(name_attr,"ERROR-OBJECT");
   gsi_set_attrs(error_object,name_attr_ptr,1);
   gsi_set_class_name(error_object,"OBJECT");

} /* End of gsi_set_up */
```

gsi_shutdown_context

Shut down a context and perform operations necessary to shut down the external system and clean up the bridge process.

Synopsis

```
void gsi_shutdown_context()
```

Description

G2 Gateway calls *gsi_shutdown_context()* when:

- The connected G2 process disables, deactivates, or deletes the GSI interface.
- The connected G2 process changes the text of the gsi-connection-configuration attribute of the GSI interface that configures this context.
- When the connected G2 is reset.
- When a network error or failure causes the connection to the G2 process to be lost.

G2 Gateway calls *gsi_shutdown_context()* immediately after it calls *gsi_receive_deregistrations()*.

By the time G2 Gateway calls *gsi_shutdown_context()*, **G2 has already closed the network connection that supports the context being shut down, or the connection has already been lost due to a network error or failure.**

Because the network connection is already closed:

Do not try to send data to G2 or invoke any G2 procedures from *gsi_shutdown_context()* through the context that is being shut down.

Do not try to call *gsi_context_socket()* from *gsi_shutdown_context()*. If you need to perform actions when a socket is opened or closed, do this through the callbacks *gsi_open_fd()* and *gsi_close_fd()*.

Example

The following *gsi_shutdown_context()* callback checks to see whether a file specified by the number of the current context is open, and closes this file if it is open, before shutting down the current context:

```
#define CLOSED 1
#define MAX_CONTEXTS 50
gsi_int file_status[MAX_CONTEXTS];
FILE *file_id[MAX_CONTEXTS] = {NULL};

. . .

/* gsi_shutdown_context() callback function */

void gsi_shutdown_context()
{
   if(file_status[current_context] != CLOSED) {
   fclose(file_id[current_context]);
   file_status[current_context] = CLOSED; }
} /* End of gsi_shutdown_context() */
```

gsi_start_context

Called by G2 Gateway whenever any G2 process that is connected to this G2 process through a permanent connection starts its current knowledge base (KB).

Synopsis

```
void gsi_start_context()
```

Description

Use this callback to respond as needed whenever G2 is started. Restarting a KB first resets and then starts it, calling first gsi_reset_context() and then gsi_start_context().

Note This callback is applicable only to permanent connections.

gsi_write_callback

Called by G2 Gateway when there is a change in the state of G2 Gateway's ability to write data to G2 on an open connection.

Synopsis

void gsi_write_callback(context, state)

Argument	Description
gsi_int context	The number of context through which G2 Gateway is trying to write data to the G2.
gsi_int state	*GSI_IO_BLOCKED*: The network cannot deliver the data. *GSI_IO_UNBLOCKED*: The network can deliver the data. These statuses are defined in *gsi_main.h*.

Description

gsi_write_callback() is called by G2 Gateway when there is a change in the state of G2 Gateway's ability to write data to G2. That is, G2 Gateway invokes this callback when:

- It becomes impossible for G2 Gateway to write data to G2, after a period when it was possible; for example, when the G2 is reading data more slowly than G2 Gateway is writing it and the operating systems buffers have become full. In this case, G2 Gateway calls *gsi_write_callback()*, passes the number of the current context to *context*, and sets *state* to *GSI_IO_BLOCKED*.
- It becomes possible for G2 Gateway to write data, after a period when it was not possible; for example, when G2 is once again able to read the data that G2 Gateway is writing. In this case, G2 Gateway calls *gsi_write_callback()*, passes the number of the current context to *context*, and sets *state* to *GSI_IO_UNBLOCKED*.

You can use *gsi_write_callback()* for purposes such as accumulating information about the flow of data, or to set indicators on a control panel.

Note If you intend to use *gsi_write_callback()*, you must declare it as described in Calling Other Functions from Callbacks, even if you are not using G2 Gateway as a DLL.

gsi_writing_fd

Called whenever a specified file descriptor is in use by G2 Gateway for writing.

Synopsis

void gsi_writing_fd (gsi_int file_descriptor)

Argument	Description
gsi_int file_descriptor	A UNIX file descriptor, usually referring to a network socket (or possibly a pipe), on which input or output can occur asynchronously.

Description

gsi_writing_fd() and the three callbacks *gsi_open_fd()*, *gsi_close_fd()*, and *gsi_not_writing_fd()* enable a bridge to wait until there is network activity to which it must respond. Use of these callbacks, rather than the API function *gsi_pause()*, is recommended if the bridge is handling non-blocking I/O outside of the control of G2 Gateway. In this case, a mask containing a 1 bit for each of the currently open file descriptors should be maintained, using *gsi_open_fd()* and *gsi_close_fd()* as well as the non-G2 Gateway file descriptors through which the bridge is performing the non-blocking I/O. This mask, and possibly also a timeout, should be passed to the *select()* system function, to perform the waiting.

Related Functions

Function	Description
gsi_open_fd()	Called whenever a specified file descriptor is opened for network I/O.
gsi_close_fd()	Called whenever a file descriptor for network I/O is closed.
gsi_not_writing_fd()	Called whenever a specified file descriptor stops being in use by G2 Gateway for writing.
gsi_watch_fd()	Specifies a file descriptor that G2 Gateway watches for network read or error activity.

Function	Description
gsi_unwatch_fd()	Causes *gsi_run_loop()* to not wake up when input or output takes place on a file descriptor.
gsi_watch_fd_for_writing()	Starts G2 Gateway watching for write activity on a file descriptor.
gsi_unwatch_fd_for_writing	Stops G2 Gateway from watching for write activity on a file descriptor.
gsi_pause()	Causes the bridge process to sleep for 1 second by default, or until a network event occurs on a network connection to the bridge process.

RPC Support Callback Functions

To support RPCs between G2 and G2 Gateway, you write local functions and receiver functions in your G2 Gateway user code. G2 can invoke (as remote procedures) G2 Gateway local functions. When G2 Gateway invokes (as a remote procedure) a G2 procedure, the G2 procedure can return values by invoking a G2 Gateway receiver function.

The RPC support callback functions include:

- Local functions invoked as remote procedures by G2.
- Receiver functions, which receive values returned by G2 procedures invoked as remote procedures by G2 Gateway.
- Error receiver functions, which receive error values returned by G2 procedures invoked as remote procedures by G2 Gateway.

See the following sections for a description of these types of callback functions:

- local functions.
- receiver functions.
- error receiver functions.
- watchdog functions.

See these other sections for instruction on using the functions:

- Writing a G2 Gateway Local Function to be Called by G2 for information on using local functions.
- Defining a Function to Receive Values Returned by G2 for information on using receiver functions.

local functions

A user-defined G2 Gateway function that G2 can call or start as a remote procedure.

Synopsis

```
void local_function (procedure_user_data, rpc_arguments,
    count, call_identifier)
```

Argument	Description
`gsi_procedure_user_data_type` *procedure_user_data*	User data associated with the call to the local function made by G2. To use this argument, you must compile your G2 Gateway code with the `GSI_USE_USER_DATA_FOR_CALLBACKS` C preprocessor flag defined or use the corresponding compile time switch. For more info see Call Identifiers for Remote Procedure Calls. This argument is optional.
`gsi_item *rpc_arguments`	An array of `gsi_item`. Items passed from G2 to G2 Gateway are stored as elements of this array.
`gsi_int count`	An integer that indicates the number of `gsi_item` structures in the array.
`gsi_call_identifier_type` *call_identifier*	An integer that G2 generates to identify a particular remote procedure call to a G2 Gateway local function, within the current context. The `gsi_rpc_return_values()` or `gsi_rpc_return_error_values()` function references *call_identifier* to indicate the outstanding remote procedure call within a specified context to return values to in G2.

Description

A G2 procedure can send values to a G2 Gateway bridge by invoking a local function in the bridge. The local function can have any name that you specify; however, it must have the arguments of the types specified in the Synopsis section above.

You must declare each receiver function in your header file. As a convenience, you can use the macro *declare_gsi_rpc_local_fn* to create the appropriate prototype declaration. The syntax is:

specifier declare_gsi_rpc_local_fn(local_function_name);

For example:

```
static declare_gsi_rpc_local_fn(my_local_function);
```

Note If you are using a C compiler that supports ANSI C, these *declare-* macros generate prototype declarations for the callbacks; otherwise, Kernighan and Ritchie style declarations are generated.

Each G2 Gateway local function that you want to call from G2 must have the arguments: *arguments*, *count*, and *call_identifier*.

It can also have a *procedure_user_data* argument, if you compile your G2 Gateway application with the *GSI_USE_USER_DATA_FOR_CALLBACKS* C preprocessor flag defined or use the corresponding compile time switch. For information about setting compile time switches, see Call Identifiers for Remote Procedure Calls. For information about how to use the *procedure_user_data* argument, see Procedure User Data for Remote Procedure Calls.

You must declare each local function to be invocable, as a remote procedure, by a connected G2 process. You do so using the API function *gsi_rpc_declare_local()*. If the declaration of the local function is missing, the callback function *gsi_missing_procedure()* is called.

To return values to G2, a local function can call the API function *gsi_rpc_return_values()*. For information about this function, see API Functions.

To return error values to G2, a local function can call the API function *gsi_return_error_values()*. For information about this function, see *gsi_rpc_return_error_values*.

If the G2 Gateway local function is invoked by a start action in G2, the *call_identifier* argument of the local function is set to *GSI_CALL_HANDLE_OF_START*. In this case, the local function should not call *gsi_rpc_return_values()*, because G2 is not expecting the local function to return any values to it.

However, a local function invoked by a start action in G2 can call *gsi_rpc_return_error_values()* to signal an error to G2.

receiver functions

A user-defined G2 Gateway function to which a G2 procedure, invoked remotely by a G2 Gateway bridge, can return values.

Synopsis

```
void receiver_function (procedure_user_data, arguments,
    count, call_identifier)
```

Argument	Description
`gsi_procedure_user_data_type` *procedure_user_data*	User data that G2 passes to G2 Gateway. The receiver function can include this argument to receive user data only if you compile your G2 Gateway application with the `GSI_USE_USER_DATA_FOR_CALLBACKS` preprocessor macro defined or use the corresponding compile time switch. For more info see Call Identifiers for Remote Procedure Calls. This argument is optional.
`gsi_item` **arguments*	An array of `gsi_item`, which contains the data values that G2 is returning to the bridge process.
`gsi_int` *count*	An integer specifying the number of values in the *arguments* array.
`gsi_call_identifier_type` *call_identifier*	Data that G2 sends to G2 Gateway to identify this particular call to the receiver function. The receiver function can include this argument to receive a call identifier value only if you compile your G2 Gateway application with the `GSI_USE_USER_DATA_FOR_CALLBACKS` preprocessor macro defined. This argument is optional.

Description

A G2 procedure invoked by a G2 Gateway bridge can return values to the bridge by invoking a receiver function in the bridge. The receiver function can have any name that you specify; however, it must have the arguments of the types specified in the Synopsis section above.

You must declare each receiver function in your header file. As a convenience, you can use the macro *declare_gsi_rpc_receiver_fn* to create the appropriate prototype declaration. The syntax is:

specifier declare_gsi_rpc_receiver_fn(receiver_function_name);

For example:

```
static declare_gsi_rpc_receiver_fn(my_receiver_function);
```

Note If you are using a C compiler that supports ANSI C, these *declare-* macros generate prototype declarations for the callbacks; otherwise, Kernighan and Ritchie style declarations are generated.

You must also specify the receiver function to which a G2 procedure can return values in the call to *gsi_rpc_declare_remote()* or *gsi_rpc_declare_remote_with_error_handler_and_user_data()* that you use to declare the G2 procedure as a remote procedure.

error receiver functions

A user-defined G2 Gateway function to which a G2 procedure, invoked remotely by a G2 Gateway bridge, can return error values.

Synopsis

void error_handler (arguments)

Argument	Description
gsi_procedure_user_data_type procedure_user_data	User data that G2 passes to G2 Gateway. The receiver function can include this argument to receive user data only if you compile your G2 Gateway application with the *GSI_USE_USER_DATA_FOR_CALLBACKS* preprocessor macro defined or use the corresponding compile time switch. For more info see Call Identifiers for Remote Procedure Calls.
*gsi_item *arguments*	Can be either: • A *gsi_item* representing an error object in G2. • A *symbolic-expression* and a *text-expression*, similar to the arguments of the signal G2 procedure statement. These arguments are identical in meaning to the *error_arguments* in a call to *gsi_rpc_return_error_values()*. For more information about these arguments, see *gsi_rpc_return_error_values*.
gsi_int count	An integer specifying the number of values in the *arguments* array.
gsi_call_identifier_type call_identifier	Data that G2 sends to G2 Gateway to identify this particular call to the receiver function. The receiver function can include this argument to receive a call identifier value only if you compile your G2 Gateway application with the *GSI_USE_USER_DATA_FOR_CALLBACKS* preprocessor macro defined.

Description

When you declare a G2 procedure as a remote procedure that can be invoked by the G2 Gateway bridge, you can specify an error receiver function in the bridge to which G2 can return error values in the case of an error.

To do this, you must use the API function *gsi_rpc_declare_remote_with_error_handler_and_user_data()* to declare the G2 procedure as a remote procedure. For information about this function, see *gsi_rpc_declare_remote_with_error_handler_and_user_data*.

The error receiver function can perform any operations necessary for the application, including evaluation of the error data returned by G2.

watchdog functions

Called by G2 Gateway when a a time-out interval specified in a call to *gsi_watchdog()* expires.

Synopsis

void user_watchdog_function ()

Description

You can write a watchdog function to perform any tasks that you want performed after a specified interval of time.

You specify both a particular watchdog function, and the interval of time after which the watchdog function is called, in a call to the API function *gsi_watchdog()*. When *gsi_watchdog()* is called, a watchdog timer, internal to GSI, begins counting down to zero from a number of seconds, which must be greater than or equal to zero.

If *gsi_watchdog()* is called again before the time-out period expires, GSI's own internal timer is set again to *timeout_interval*, which puts off the call to the user-written watchdog function.

At any time, by calling *gsi_watchdog()* and passing it a *timeout_interval* of zero (0), your G2 Gateway application can disable G2 Gateway's internal watchdog timer.

Note *gsi_watchdog()* is not supported on Windows.

You must declare each watchdog function in your header file. As a convenience, you can use the macro *declare_gsi_watchdog_function* to create the appropriate prototype declaration. The syntax is:

specifier declare_gsi_watchdog_function(watchdog_function_name);

For example:

```
static declare_gsi_watchdog_function(my_watchdog_function);
```

Using the Select Function in G2 Gateway

The C-library *select()* function lets you wait until activity is present on any one of a specified set of file descriptors (fds). Use the *select()* function when writing a bridge that performs network I/O that is independent of G2 Gateway, and needs to determine when activity occurs on both the bridge's fds, and on the fds of G2 Gateway.

G2 Gateway provides functions for:

- A bridge to inform G2 Gateway of its file descriptors.
- G2 Gateway to inform the bridge of its file descriptors.

While it is not necessary for a bridge to use either kind of function, using at least one kind helps to achieve optimal performance.

Use the functions as follows:

These functions...	Are called by...
gsi_watch_fd *gsi_unwatch_fd* *gsi_watch_fd_for_writing* *gsi_unwatch_fd_for_writing*	A bridge to inform G2 Gateway of its fds. G2 Gateway then uses this information inside *gsi_pause()* so that if there is activity on any bridge fd, *gsi_pause()* returns to the user.
gsi_open_fd *gsi_close_fd* *gsi_writing_fd* *gsi_not_writing_fd*	G2 Gateway to inform the bridge of its fds, so that a *select()* statement in the bridge can wake up whenever activity occurs on a G2 Gateway fd.

If a bridge includes a *select()* statement with only the bridge's fds, then the bridge may experience a delay equal to the timeout value you supply to the *select()* statement. Specifying a timeout of null causes the bridge to hang.

Similarly, if the bridge calls *gsi_pause()* without first telling G2 Gateway of the bridge's fds, a delay equal to the default timeout, or that specified by *gsi_set_pause_timeout()*, may occur when there is activity on the bridge's fds.

Supplying Arguments to the Select Function

This section shows you how to use G2 Gateway API functions to create values for *readfds*, *writefds*, and *exceptfds*. For information about the C-library *select()* function itself, its other arguments, and its return value, see the documentation for that function.

The syntax of the *select()* function is:

```
int select(int maxfds,
   fd_set *readfds,
   fd_set *writefds,
   fd_set *exceptfds,
   struct timeval *tvptr)
```

Argument	Description
*fd_set *readfds*	An fd_set that includes both the bridge's fds and those of G2 Gateway. Every time *gsi_open_fd()* is called, you should add the fd argument of that function to the select readfds argument. Every time *gsi_close_fd()* is called, remove the fd argument of that function from the select readfds argument.
*fd_set *writefds*	An fd_set that includes both the bridge's fds and those of G2 Gateway. Every time *gsi_writing_fd()* is called, you should add the fd argument of that function to the select writefds argument. Every time *gsi_not_writing_fd()* is called, remove the fd argument of that function from the select writefds argument.
*fd_set *exceptfds*	An *fd_set* that should be the same as the *readfds* argument.

The following code example shows how to supply arguments to the *select()* function:

```
#include "gsi_main.h"

fd_set all_read_fds, all_write_fds;
fd_set select_read_fds, select_write_fds, select_except_fds;
int max_fd = 0;
declare_gsi_open_fd(gsi_open_fd);
declare_gsi_close_fd(gsi_close_fd);
declare_gsi_writing_fd(gsi_writing_fd);
declare_gsi_not_writing_fd(gsi_not_writing_fd);
```

```
void initialize_fd_sets() {
   FD_ZERO(&all_read_fds);
   FD_ZERO(&all_write_fds);
   gsi_install_open_fd(gsi_open_fd);
   gsi_install_close_fd(gsi_close_fd);
   gsi_install_writing_fd(gsi_writing_fd);
   gsi_install_not_writing_fd(gsi_not_writing_fd);
   }


/* The following four functions will be called as needed by */
/* G2 Gateway. The bridge should also call these functions, */
/* so that all_read_fds and all_write_fds will contain both /*
/* G2 Gateway and bridge fds. */


void gsi_open_fd(fd)
   gsi_int fd; {
   FD_SET(fd, &all_read_fds);
   if (fd > max_fd) max_fd = fd;
   }


void gsi_close_fd(fd)
   gsi_int fd; {
   FD_CLR(fd, &all_read_fds);
   }


void gsi_writing_fd(fd)
   gsi_int fd; {
   FD_SET(fd, &all_write_fds);
   }


void gsi_not_writing_fd(fd)
   gsi_int fd; {
   FD_CLR(fd, &all_write_fds);
   }


void copy_fd_set(out_fd_set, in_fd_set)
   fd_set *out_fd_set, *in_fd_set; {
   int fd;
   FD_ZERO(&out_fd_set);
   for (fd=0; fd<=max_fd; fd++)
   if (FD_ISSET(fd, &in_fd_set))
   FD_SET(fd, &out_fd_set);
   }
```

```
int wait_for_something_to_do(timeout_in_milliseconds)
   int timeout_in_milliseconds; {
   struct timeval timeout, *select_timeout = NULL;
   int select_result;
   if (timeout >= 0) {
      timeout.tv_sec = timeout_in_milliseconds / 1000;
      timeout.tv_usec =
         (timeout_in_milliseconds -
         (timeout.tv_sec * 1000))
         * 1000;
      select_timeout = &timeout;
      }


   /* negative timeout means wait forever */
   copy_fd_set(&select_read_fds, &all_read_fds);
   copy_fd_set(&select_write_fds, &all_write_fds);
   copy_fd_set(&select_except_fds, &all_read_fds);
   select_result = select(max_fd+1, &select_read_fds,
      &select_write_fds, &select_except_fds, select_timeout);
   return select_result;
   }
```

9

API Functions

Describes the capabilities and syntax of the API functions supported by G2 Gateway.

Introduction

G2 Gateway provides a large set of Application Programmer Interface (API) functions.

API functions are commonly called from within G2 Gateway callback functions, which are invoked by G2 Gateway while your bridge process is executing under the control of the *gsi_run_loop()* API function. For detailed information about callback functions, see Callback Functions.

API functions can also be called from outside *gsi_run_loop()*, if *gsi_start()* has already been called. In order to be able to transfer control from *gsi_run_loop()* to other parts of your user code, you must run your G2 Gateway bridge process in one-cycle mode. For information about how to do this, see The main() Function in Continuous and One-Cycle Modes.

Groups of Functionally Related API Functions

This section lists API functions grouped by the kinds of functions that they perform.

G2 Gateway Entry Points

gsi_start()
gsi_run_loop()
gsi_set_runloop_timeout()

Initialization and Run State

gsi_initialize_for_wind32()
gsi_set_include_file_version()
gsi_initialize_callbacks()
gsi_run_state_change()

Context Management

gsi_establish_listener
gsi_establish_secure_listener
gsi_initiate_connection()
gsi_initiate_connection_with_user_data()
gsi_initiate_secure_connection()
gsi_initiate_secure_connection_with_user_data()
gsi_context_received_data()
gsi_current_context()
gsi_current_context_is_secure()
gsi_context_is_secure
gsi_flush()
gsi_set_context_limit()
gsi_kill_context()
gsi_context_user_data()
gsi_set_context_user_data()

Data Structure Access

Type and Value Access

gsi_element_count_of()
gsi_elements_of()
gsi_flt_array_of()
gsi_flt_list_of()
gsi_flt_of()
gsi_handle_of()
gsi_int_array_of()
gsi_int_list_of()
gsi_int_of()
gsi_is_item()
gsi_log_array_of()
gsi_log_list_of()
gsi_log_of()
gsi_long_of()
gsi_name_of()
gsi_str_array_of()
gsi_str_list_of()
gsi_str_of()
gsi_sym_array_of()
gsi_sym_list_of()
gsi_sym_of()
gsi_type_of()
gsi_usv_of()
gsi_usv_length_of()

gsi_set_elements()
gsi_set_flt_array()
gsi_set_flt_list()
gsi_set_flt()
gsi_set_handle()
gsi_set_int_array()
gsi_set_int_list()
gsi_set_int()
gsi_set_log_array()
gsi_set_log_list()
gsi_set_log()
gsi_set_long()
gsi_set_name()
gsi_set_str_array()
gsi_set_str_list()
gsi_set_str()
gsi_set_sym_array()
gsi_set_sym_list()
gsi_set_sym()
gsi_set_type()
gsi_set_usv()
gsi_usv_length_of()

Attribute Access

gsi_attrs_of()
gsi_attr_by_name()
gsi_attr_count_of()
gsi_attr_is_transient()
gsi_attr_is_array_index()
gsi_attr_array_index_of()
gsi_set_attr_array_index()
gsi_identifying_attr_of()

gsi_set_attrs()
gsi_set_attr_by_name()
gsi_set_attr_is_transient()
gsi_attr_is_list_index()
gsi_attr_list_index_of()
gsi_set_attr_list_index()

Attribute Name Access

gsi_attr_name_of()
gsi_set_attr_name()
gsi_unqualified_attr_name_of()
gsi_set_unqualified_attr_name()
gsi_class_qualifier_of()
gsi_set_class_qualifier()
gsi_attr_name_is_qualified()

Timestamp and History Access

gsi_decode_timestamp()
gsi_encode_timestamp()
gsi_timestamp_of()
gsi_set_timestamp()
gsi_extract_history()
gsi_extract_history_spec()
gsi_history_type_of()
gsi_set_history()
gsi_history_count_of()

Miscellaneous Data Structure Access

gsi_status_of()
gsi_set_status()
gsi_interval_of()
gsi_interval_of()
gsi_item_of_attr()
gsi_set_item_of_attr()
gsi_item_of_registered_item()
gsi_set_user_data()
gsi_user_data_of()
gsi_simple_content_copy()
gsi_set_element_count()
gsi_set_class_name()
gsi_class_name_of()
gsi_registration_of()
gsi_identifying_attr_of()
gsi_version_information()
gsi_clear_item()

Data Service

gsi_return_values()
gsi_return_attrs()
gsi_return_timed_values()
gsi_return_timed_attrs()

Data Structure Allocation and Deallocation

gsi_make_attrs()
gsi_make_attrs_with_items()
gsi_reclaim_attrs_with_items()
gsi_make_items()
gsi_make_registered_items()
gsi_reclaim_registered_items()
gsi_reclaim_attrs()
gsi_reclaim_items()

Error Handling

gsi_error_handler()
gsi_initialize_error_variable()
gsi_last_error()
gsi_last_error_call_handle()
gsi_clear_last_error()
gsi_signal_error()
gsi_install_error_handler()
gsi_last_error_message()
gsi_error_message()
gsi_signal_handler()

File Descriptor Management

gsi_open_fd()
gsi_close_fd()

Interruptible Sleep

gsi_context_socket()
gsi_pause()
gsi_unwatch_fd()
gsi_watch_fd()
gsi_listener_socket()
gsi_set_pause_timeout()
gsi_wakeup()

Message Passing

gsi_return_message()

Missing Callback Declarations

gsi_missing_procedure_handler()

Remote Procedure Support

gsi_rpc_declare_local()
gsi_rpc_declare_remote()
gsi_rpc_call()
gsi_rpc_start()
gsi_rpc_declare_remote_with_error_handler_and_user_data()
gsi_rpc_return_error_value()
gsi_rpc_return_values()
gsi_set_rpc_remote_return_value_kind()
gsi_set_rpc_remote_return_exclude_user_attrs
gsi_set_rpc_remote_return_include_system_attrs
gsi_set_rpc_remote_return_include_all_system_attrs_except

Runtime Options

gsi_option_is_set()
gsi_reset_option()
gsi_set_option()

String Conversion

gsi_convert_string_to_unicode()
gsi_convert_unicode_to_string()
gsi_convert_unicode_to_wide_string()
gsi_convert_wide_string_to_unicode()
gsi_string_conversion_style()
gsi_set_string_conversion_style()

Symbol Access

gsi_make_symbol()
gsi_symbol_name()
gsi_symbol_user_data()
gsi_set_symbol_user_data()
gsi_sym_of()
gsi_sym_array_of()
gsi_sym_list_of()
gsi_set_sym()
gsi_set_sym_array()
gsi_set_sym_list()

User Data

gsi_user_data_of() *gsi_set_user_data()*

Watchdog Function

gsi_watchdog()

Required Header File

To use the G2 Gateway API functions, you must include the header file *gsi_main.h*, using the following statement:

```
#include "gsi_main.h"
```

Specifying Symbolic Values in API Function Calls

G2 uses the symbol type for all G2 identifiers, such as names of G2 items, classes, and attributes. G2 expresses symbol values in uppercase letters by default. To include a lower case letter in a G2 symbol in a G2 editor, you must precede it with an escape character.

When G2 Gateway returns a symbol value to G2, all characters in the symbol value arrive in G2 in uppercase form, except for characters that were preceded by escape characters. For this reason, you must use only uppercase letters to specify any API function argument that represents a G2 identifier, unless you know that the corresponding G2 symbol value uses escape characters to express lower case characters.

You must also use uppercase letters to specify arguments that you pass to the API functions *gsi_set_sym()*, *gsi_set_sym_array()* and *gsi_set_sym_list()*, which set the value of a structure to a symbol value (type *GSI_SYMBOL_TAG)*.

Caution Symbols and text strings cannot be used interchangeably. For example, using *gsi_set_sym* on a text attribute will fail and using *gsi_set_str* on a symbol attribute will fail.

API Function Descriptions

The rest of this chapter presents detailed descriptions of the API functions, presented in alphabetical order.

gsi_attr_by_name

Invokes *gsi_item_of_attr_by_name()*.

For information about this API function, see *gsi_item_of_attr_by_name*.

gsi_attr_count_of

Returns the number of attributes associated with an item.

Synopsis

`gsi_int gsi_attr_count_of(item)`

`gsi_int gsi_attr_count_of(attribute)`

Argument	Description
`gsi_item` *item*	A `gsi_item` for which this function returns a count of attributes.
`gsi_attr` *attribute*	A `gsi_attr` containing an embedded `gsi_item` for which this function returns a count of attributes.

Return Value	Description
`gsi_int`	Represents the number of attributes in *item* or *attribute*. **Note**: The return value is `0` if the argument passed to this function is a `gsi_item` extracted from a `gsi_registered_item` by `gsi_item_of_registered_item()`, and this `gsi_registered_item` was obtained from a callback function, such as `gsi_set_data()`.

Description

`gsi_attr_count_of()` returns the number of attributes in an item or embedded item in an attribute.

gsi_attr_is_transient

Returns a value indicating whether or not a specified attribute is transient. Transient attributes are not passed in remote procedure calls.

Synopsis

gsi_int gsi_attr_is_transient(attribute)

Argument	Description
gsi_attr attribute	The attribute.

Return Value	Description
gsi_int	1 if the attribute is transient 0 if it is not transient.

Description

G2 Gateway ignores transient attributes in remote procedure calls, and does not send them to G2.

You can cause remote procedure calls either to pass or not to pass particular attributes by setting the attributes to be not transient or transient. To do this, use the API function *gsi_set_attr_is_transient()*.

The API functions *gsi_return_attrs()* and *gsi_return_timed_attrs()* return transient attributes only.

gsi_attr_name_is_qualified

Returns *true* if the name of an attribute is class-qualified, and *false* otherwise.

Synopsis

*gsi_int gsi_attr_name_is_qualified(**attribute**)*

Argument	Description
*gsi_attr **attribute***	The attribute whose name is examined by this function.

Return Value	Description
gsi_int	1 if the name is class-qualified, and 0 otherwise.

Description

gsi_attr_name_is_qualified() returns a value that indicates whether the *name* component in the *gsi_attr* structure specified by *attribute* includes a class qualifier.

Class qualifiers are required only for items of a class defined using multiple-inheritance. Multiple inheritance makes it possible for a class definition to include different attributes of the same name, inherited from different direct superior classes. The class-qualifiers added to the names of these attributes distinguish the attributes from each other.

Related Functions

Function	Description
gsi_attr_name_of()	Returns the name of an attribute.
gsi_set_attr_name()	Changes the name of an attribute.
gsi_unqualified_attr_name_of()	Returns the unqualified part of an attribute's name.

Function	Description
gsi_class_qualifier_of()	Returns the part of an attribute *name* that is the class qualifier.
gsi_set_class_qualifier()	Changes the part of an attribute name that specifies the G2 class that defines the attribute.

gsi_attr_name_of

Returns the name of an attribute.

Synopsis

gsi_symbol gsi_attr_name_of(attribute)

Argument	Description
gsi_attr attribute	The attribute whose name is returned by this function.

Return Value	Description
gsi_symbol	A read-only symbol giving the name of the attribute. The symbol persists only as long as the *gsi_attr* data structure with which it is associated. If your user code needs to keep the symbol for longer than the life-span of the *gsi_attr* structure, it must copy the symbol either into user allocated memory or a G2 Gateway data structure.

Description

Use *gsi_attr_name_of()* to access the value of the *name* component of the *gsi_attr* specified by the *attribute* argument.

Related Functions

Function	Description
gsi_set_attr_name()	Changes the name of an attribute.
gsi_unqualified_attr_name_of()	Returns the unqualified part of an attribute's name.
gsi_set_unqualified_attr_name()	Sets the unqualified part of an attribute's name.

Function	Description
gsi_attr_name_is_qualified()	Indicates whether an attribute name is qualified.
gsi_class_qualifier_of()	Returns the part of an attribute *name* that is the class qualifier.
gsi_set_class_qualifier()	Changes the part of an attribute name that specifies the G2 class that defines the attribute.

gsi_attrs_of

Returns an array of *gsi_attr* structures, each of which corresponds to an attribute of an item.

Synopsis

*gsi_attr *gsi_attrs_of(item)*

*gsi_attr *gsi_attrs_of(attribute)*

Argument	Description
gsi_item *item*	The *gsi_item* whose attributes are returned.
gsi_attr *attribute*	The *gsi_attr* containing an embedded *gsi_item* whose attributes are returned.

Return Value	Description
*gsi_attr **	An array of *gsi_attr* structures, which correspond to the attributes of a *gsi_item*.

Description

gsi_attrs_of() extracts an array of *gsi_attr* structures corresponding to the attributes of a *gsi_item* or *gsi_attr*.

This function does not allocate any new memory. Its return value points to the array stored in *item* or *attribute*, and neither the array nor any of its elements are copied.

Related Functions

Function	Description
gsi_attr_count_of()	Determines how many attributes are in a *gsi_item* or *gsi_attr*.
gsi_set_attrs()	Changes the attributes in a *gsi_item* or *gsi_attr*.

Function	Description
gsi_attr_by_name()	Obtains a specific attribute in a *gsi_item* or *gsi_attr*.
gsi_set_attr_by_name()	Changes a specific attribute in a *gsi_item* or *gsi_attr*.

gsi_class_name_of

Returns the name of the G2 class of an item.

Synopsis

gsi_symbol gsi_class_name_of(item)

gsi_symbol gsi_class_name_of(attribute)

Argument	Description
gsi_item item	A *gsi_item* whose class name is returned by this function.
gsi_attr attribute	An attribute containing an embedded *gsi_item* whose class name is returned by this function.

Return Value	Description
gsi_symbol	A symbol containing the name of the G2 class of the specified item. If the specified item has no associated class or is not named, the return value is a null pointer. See **Discussion** below. The symbol persists only as long as the data structure with which it is associated. If your user code needs to keep the symbol for longer than the life-span of the data structure, it must copy the symbol into memory that it has allocated itself.

Description

gsi_class_name_of() returns the name of the G2 class of an item represented by a *gsi_item* structure. As an argument, *gsi_class_name_of()* can take either a *gsi_item* or a *gsi_attr* representing an attribute containing an embedded *gsi_item*. In either case, *gsi_class_name_of()* returns the class name of the G2 item represented by the *gsi_item*.

gsi_class_name_of() can return the names of both system-defined and user-defined G2 classes. There is no restriction on the class of G2 item whose class name is returned by *gsi_class_name_of()*.

Caution If the specified item has no associated class or is not named, the return value is a null pointer. On some systems, passing a null pointer to *printf()* causes a segmentation violation. For this reason, you may want to verify that the return value of *gsi_class_name_of()* is not a null pointer before you pass it to *printf()*. For example:

```
gsi_symbol char_temp;
char_temp = gsi_class_name_of(item);
if (char_temp)
   printf("\ class: %s", char_temp);
else
   printf("\n class:NULL_PTR");
```

gsi_class_qualifier_of

Returns the part of the *name* component of an attribute that is the class qualifier.

Synopsis

gsi_symbol gsi_class_qualifier_of(attribute)

Argument	Description
gsi_attr attribute	The attribute whose name is class-qualified.

Return Value	Description
gsi_symbol	A read-only symbol that contains the class-qualifier part of the *name* component of *attribute*. The symbol persists only as long as the data structure with which it is associated. If your user code needs to keep the symbol for longer than the life-span of the data structure, it must copy the symbol into memory that it has allocated itself.

Description

Use *gsi_class_qualifier_of()* to access the class-qualifier portion of the *name* component of a *gsi_attr*.

This function does not allocate any new memory. Its return value points to a C string stored in *attribute*, and not to a copy of that string.

Related Functions

Function	Description
gsi_attr_name_of()	Returns the name of an attribute.
gsi_set_attr_name()	Changes the name of an attribute.
gsi_unqualified_attr_name_of()	Returns the unqualified part of an attribute's name.

Function	Description
gsi_set_unqualified_attr_name()	Sets the unqualified part of an attribute's name.
gsi_attr_name_is_qualified()	Indicates whether an attribute name is qualified.
gsi_set_class_qualifier()	Changes the part of an attribute name that specifies the G2 class that defines the attribute.

gsi_class_type_of

Returns the type of the history data values associated with an item, a registered item, or an item that is embedded in an attribute.

Synopsis

gsi_int gsi_class_type_of(item)

gsi_int gsi_class_type_of(registered_item)

gsi_int gsi_class_type_of(attribute)

Argument	Description
gsi_item item	An item from which this function returns the type of the associated history data values.
gsi_registered_item registered_item	A registered item from which this function returns the type of the associated history data values.
gsi_attr attribute	An attribute containing an embedded item from which this function returns the type of the associated history data values.

Return Value	Description
gsi_int	One of the following G2 Gateway types: *GSI_INTEGER_TAG* *GSI_SYMBOL_TAG* *GSI_STRING_TAG* *GSI_LOGICAL_TAG* *GSI_FLOAT64_TAG* *GSI_VALUE_TAG* *GSI_QUANTITY_TAG* *GSI_NULL_TAG*

Description

gsi_class_type_of() returns the type of the history data values associated with an item, a registered item, or with an item that is embedded in an attribute. If there is no history associated with the item, registered item, or embedded item in

the attribute, *gsi_class_type_of()* returns the type that most closely corresponds to the variable's type in G2.

If a *gsi_item* that G2 sends to G2 Gateway through a remote procedure call corresponds to a variable-or-parameter, this field derives from the data type in G2, and reflects the allowable values for the variable-or-parameter.

If the item is neither a variable nor a parameter, the function returns *GSI_NULL_TAG*.

The types of the history values are shown in the following table:

Return Value	G2 Gateway Element Type	C Element Type
GSI_INTEGER_TAG	homogeneous integer values	*gsi_int*
GSI_SYMBOL_TAG	homogeneous symbol values	*char **
GSI_STRING_TAG	homogeneous string values	*char **
GSI_LOGICAL_TAG	homogeneous truth-values	*gsi_int*
GSI_FLOAT64_TAG	homogeneous floating-point numbers	*double*
GSI_VALUE_TAG	heterogeneous values	*gsi_item*
GSI_QUANTITY_TAG	heterogeneous numbers (*gsi_int* or *double*)	*gsi_item*

Related Functions

Function	Description
gsi_history_count_of()	Returns the number of history data values associated with an item.
gsi_extract_history()	Returns history data values associated with an item.
gsi_extract_history_spec()	Returns the history-keeping specification for an item.

gsi_clear_item

Clears an item for reuse.

Synopsis

void gsi_clear_item(item)

void gsi_clear_item(regitem)

void gsi_clear_item(attribute)

Argument	Description
gsi_item item	The *gsi_item* structure that this function clears for reuse.
gsi_registered_item regitem	The *gsi_registered_item* structure that points to the *gsi_item* cleared for reuse.
gsi_attr attribute	The *gsi_attr* structure containing an embedded *gsi_item* cleared for reuse.

Description

The *gsi_clear_item()* function clears the specified *gsi_registered_item*, *gsi_item*, or *gsi_attr* structure by:

- Reclaiming memory associated with the *value* and *history* components of the structure. This memory is not in the form of G2 Gateway data structures, but is stored in arrays of simple value types.
- Setting counts to zero.
- Setting the type to *GSI_NULL_TAG*.
- Setting the class name to *NULL_PTR*.
- Eliminating references to other structures.

  gsi_clear_item() does not reclaim memory explicitly allocated by your user code. If any of these data structures were explicitly allocated by your G2 Gateway user code, the user code is responsible for reclaiming the memory used by those data structures.

gsi_clear_last_error

Resets the value of G2 Gateway's last error number.

Synopsis

```
void gsi_clear_last_error()
```

Description

`gsi_clear_last_error()` sets the value of G2 Gateway's error number to zero (0).

For information about the role of G2 Gateway's last error number, see Error Handling.

gsi_close_listeners

Closes all network listeners.

Synopsis

```
void gsi_close_listeners()
```

Description

The function `gsi_close_listeners()` closes all network listeners. Use this function if you want to close listeners before the G2 Gateway process has finished.

gsi_context_is_secure

Returns the security status of the specified context.

Synopsis

```
gsi_int gsi_context_is_secure(context)
```

Return Value	Description
gsi_int	The security status of the current context: 0 (insecure) or 1 (secure).

Description

Returns the security status of the specified context, which is a *gsi_int*.

gsi_context_received_data

Indicates whether there was network activity during the most recent invocation of *gsi_run_loop()*.

Synopsis

gsi_int gsi_context_received_data(context_number)

Argument	Description
gsi_int context_number	A context number. You can obtain the number of the current context using the API function *gsi_current_context()*.

Return Value	Description
gsi_int	1 if there was network activity during the most recent invocation; otherwise 0.

Description

gsi_context_received_data() takes a context number as an input argument and returns 1 if there was network activity of any kind on the specified context during the most recent pass through *gsi_run_loop()*. If there was no network activity on the specified context during the most recent pass through *gsi_run_loop()*, this function returns 0.

The function detects any kind of network activity, such as remote procedure calls from G2 or calls to any of the callback functions associated with data service.

Note This function is designed to be used in one-cycle mode. It is not useful in continuous mode.

gsi_context_remote_host

Returns a string naming the host to which G2 Gateway is connected.

Synopsis

```
gsi_char *gsi_context_remote_host (context)
```

Argument	Description
`gsi_int context`	The current G2 Gateway context.

Return value	Description
`gsi_char *`	A string of the host name.

Description

`gsi_context_remote_host()` returns the name of the host to which G2 Gateway is connected. The returned value is the same string that G2 prints in its title block.

gsi_context_remote_listener_port

Returns the TCP/IP port on which the remote G2 is listening.

Synopsis

gsi_int gsi_context_remote_listener_port(*context*)

Argument	Description
gsi_int context	The current G2 Gateway context.

Return value	Description
gsi_int	The TCP/IP port number.

Description

gsi_context_remote_listener_port() returns the G2 TCP/IP port.

gsi_context_remote_process_start_time

Returns a floating-point value representing the G2 process launch time.

Synopsis

```
double gsi_context_remote_process_start_time (context)
```

Argument	Description
gsi_int context	The current G2 Gateway context.

Return value	Description
double	A floating-point value timestamp.

Description

gsi_context_remote_process_start_time() returns a floating-point value, which you can pass to *gsi_decode_timestamp()* to obtain a timestamp.

Related Functions

Function	Description
gsi_decode_timestamp()	Converts a floating-point timestamp value into its component parts.

gsi_context_socket

Returns the file descriptor associated with a specified connection (context).

Synopsis

gsi_int gsi_context_socket(context_number)

Argument	Description
gsi_int context_number	A context number. You can obtain this number using the API function *gsi_current_context()*.

Return Value	Description
gsi_int	The file descriptor associated with the context specified by *context_number*.

gsi_context_user_data

Returns the user data associated with a context that was initiated by a call to *gsi_initiate_connection_with_user_data()* and set by a call to *gsi_set_context_user_data()*.

Synopsis

gsi_context_user_data_type gsi_context_user_data(context)

Argument	Description
gsi_int context	The context from which user data is returned.

Return Value	Description
gsi_context_user_data_type	The user data associated with *context*.

gsi_convert_string_to_unicode

Converts a string in a specified style to Unicode.

Synopsis

*short *gsi_convert_string_to_unicode(string, style)*

Argument	Description
*char *string*	The string to be converted to Unicode.
gsi_int style	The string conversion style from which *string* is converted. For a list of the supported string conversion styles, see the description of *gsi_set_string_converson_style*.

Return Value	Description
*short **	An array each element of which contains a character code.

Description

Use *gsi_convert_string_to_unicode()* to convert a particular string to Unicode. Because this function returns an array of *short* each time, reclaiming the array it returned last time, you need to copy the array of *short* before the next call to *gsi_convert_string_to_unicode()*.

For information about how to specify automatic string conversions for all strings, see *gsi_set_string_converson_style*.

Note The result of *gsi_convert_string_to_unicode()* remains valid until the next call of any of the unicode convert functions:

gsi_convert_string_to_unicode()
gsi_convert_unicode_to_string()
gsi_convert_unicode_to_wide_string()
gsi_convert_wide_string_to_unicode()

gsi_convert_unicode_to_string

Converts a string of characters in Unicode to a specified string conversion style.

Synopsis

*char *gsi_convert_unicode_to_string(string, style)*

Argument	Description
*short *string*	The string to be converted to Unicode.
gsi_int style	The string conversion style to which *string* is converted. For a list of the supported string conversion styles, see the description of *gsi_set_string_converson_style*.

Return Value	Description
*char **	The converted string of characters.

Description

Use *gsi_convert_unicode_to_string()* to convert a particular string in Unicode characters to a specified string conversion style.

Since this function returns an array of *short* each time, reclaiming the array it returned last time, you need to copy the array of *short* before the next call to *gsi_convert_unicode_to_string()*.

For information about how to specify automatic string conversions for all strings, see *gsi_set_string_converson_style*.

Note The result of *gsi_convert_unicode_to_string()* remains valid until the next call of any of the unicode convert functions:

gsi_convert_string_to_unicode()
gsi_convert_unicode_to_string()
gsi_convert_unicode_to_wide_string()
gsi_convert_wide_string_to_unicode()

gsi_convert_unicode_to_wide_string

Converts a string of characters in Unicode to a wide string conversion style.

Synopsis

*short *gsi_convert_unicode_to_wide_string(string, style)*

Argument	Description
*short *string*	The Unicode string of characters to convert to *style*.
gsi_int style	The string conversion style to which *string* is converted.

Return Value	Description
*short **	An array each element of which contains a character code.

Description

Since this function returns an array of *short* each time, reclaiming the array it returned last time, you need to copy the array of *short* before the next call to *gsi_convert_unicode_to_wide_string()*.

Note The result of *gsi_convert_unicode_to_wide_string()* remains valid until the next call of any of the unicode convert functions:

gsi_convert_string_to_unicode()
gsi_convert_unicode_to_string()
gsi_convert_unicode_to_wide_string()
gsi_convert_wide_string_to_unicode()

gsi_convert_wide_string_to_unicode

Converts a string of *short* characters to Unicode.

Synopsis

*short *gsi_convert_wide_string_to_unicode(string, style)*

Argument	Description
*short *string*	The string to be converted to Unicode characters.
gsi_int style	The string conversion style from which *string* is converted. For a list of the supported string conversion styles, see *gsi_set_string_converson_style*.

Return Value	Description
*short **	An array each element of which contains a character code.

Description

Since this function returns an array of *short* each time, reclaiming the array it returned last time, you need to copy the array of *short* before the next call to *gsi_convert_wide_string_to_unicode()*.

Note The result of *gsi_convert_wide_string_to_unicode()* remains valid until the next call of any of the unicode convert functions:

gsi_convert_string_to_unicode()
gsi_convert_unicode_to_string()
gsi_convert_unicode_to_wide_string()
gsi_convert_wide_string_to_unicode()

gsi_current_context

Returns the number of the current context.

Synopsis

```
gsi_int gsi_current_context()
```

Return Value	Description
gsi_int	An integer from 0 to 49 that represents the current context number, or -1, indicating that the context is undefined.

Description

gsi_current_context() returns the number of the current context. A context number is an integer value that identifies a connection between the G2 Gateway bridge process and a GSI interface object in some G2 process. If more than one G2 process is connected to the same bridge process, each connection has a unique context number.

The possible value of an active context is between 0 and 49.

gsi_current_context() returns -1 if the current context is undefined. The current context is undefined when:

- The bridge process is executing outside the extent of *gsi_run_loop()*.
- When the current context has been shut down and the bridge process is executing within a customized error handler.

gsi_current_context_is_secure

Returns the security status of the current context.

Synopsis

```
gsi_int gsi_current_context_is_secure()
```

Return Value	Description
gsi_int	The security status of the current context: 0 (insecure) or 1 (secure).

Description

Returns the security status of the current context.

gsi_decode_timestamp

Converts a floating-point timestamp value into its component parts.

Synopsis

```
void gsi_decode_timestamp(timestamp, year_address, month_address,
   day_address, hour_address, minute_address, second_address)
```

Argument	Description
`double` *timestamp*	The timestamp converted by this function.
`gsi_int` **year_address*	A pointer to the 4-digit year component of the timestamp.
`gsi_int` **month_address*	A pointer to the month component of the timestamp.
`gsi_int` **day_address*	A pointer to the day component of the timestamp.
`gsi_int` **hour_address*	A pointer to the hour component of the timestamp.
`gsi_int` **minute_address*	A pointer to the minute component of the timestamp.
`gsi_int` **second_address*	A pointer to the second component of the timestamp.

Description

The function `gsi_decode_timestamp()` converts a floating-point timestamp value into its component parts by modifying its *xxx_address* arguments. UNIX format places 0.0 at 12 AM January 1, 1970 GMT.

Caution The internal G2 clock has a limit of +/- 17 years from the time that G2 is started. Using a timestamp that extends beyond this limitation may cause unexpected results. You are encouraged to do validity checking of timestamps if you suspect that this may be an issue.

gsi_element_count_of

Returns the number of elements in the *value* component of an item.

Synopsis

gsi_int gsi_element_count_of(item)

gsi_int gsi_element_count_of(reg-item)

gsi_int gsi_element_count_of(attribute)

Argument	Description
gsi_item item	The *gsi_item* for which this function returns the count of elements in the *value* component.
gsi_registered_item reg-item	The *gsi_registered_item* pointing to a *gsi_item* for which the count is returned.
gsi_attr attribute	The *gsi_attr* containing an embedded *gsi_item* for which a count is returned.

Return Value	Description
gsi_int	The number of elements in the *value* components of an item, registered item, or embedded item in an attribute.

Description

Use *gsi_element_count_of()* to determine the number of elements in the *value* component of an item, item referenced by a registered item, or embedded item in an attribute that is a list or array.

The element count returned by this function is non-zero if the type of the argument passed to the function is any of the following:

- A list type containing one or more elements.
- An array type containing one or more elements.
- A sequence containing one or more elements.
- A string the element count is the length of the string.
- An unsigned short vector the element count is the length of the vector.

gsi_elements_of

Returns an array of *gsi_item*, representing the *value(s)* component of an item when the value is of type *GSI_ITEM_ARRAY_TAG*, *GSI_ITEM_LIST_TAG*, *GSI_VALUE_ARRAY_TAG*, or *GSI_VALUE_LIST_TAG*.

Synopsis

*gsi_item *gsi_elements_of(item)*

*gsi_item *gsi_elements_of(attribute)*

Argument	Description
gsi_item item	The item from which this function returns the *value* component as an array of *gsi_item*.
gsi_attr attribute	The attribute containing an embedded *gsi_item* whose *value* component is returned by this function returns as an array of *gsi_item*.

Return Value	Description
*gsi_item **	An array of *gsi_item* instances.

Description

The function gsi_elements_of() extracts an array of *gsi_item* instances from *item* or *attribute*. This array is a one-dimensional C array, suitable for manipulation by address arithmetic.

If the *value* component of *item* or *attribute* is neither an item array, item list, value array, nor value list, G2 Gateway signals an error.

To see a complete description of type tags for item arrays and lists refer to G2 Data Types and G2 Gateway Type Tags.

For value arrays and value lists, the type of each element will be one of:

GSI_FLOAT64_TAG
GSI_STRING_TAG
GSI_LOGICAL_TAG
GSI_SEQUENCE_TAG
GSI_SYMBOL_TAG
GSI_STRUCTURE_TAG
GSI_INTEGER_TAG

For quantity arrays and lists, the type of each element will be one of:

GSI_FLOAT64_TAG
GSI_INTEGER_TAG

This function does not allocate any new memory. It returns the array or list stored in *item* or *attribute*, and neither it nor any of its elements are copied.

Related Functions

Function	Description
gsi_type_of()	You can use this function to determine the type of the element returned by *gsi_elements_of()*.
gsi_set_elements()	Modifies a *gsi_item* or *gsi_attr* so that its *value* component stores an item array, item list, value array, or value list.

gsi_encode_timestamp

Converts a year, month, day, hour, minute, and second into a floating-point timestamp value.

Synopsis

double gsi_encode_timestamp (year,month,day,hour,minute,second)

Argument	Description
gsi_int year	A value that is converted into the 4-digit year component of the timestamp.
gsi_int month	A value that is converted into the month component of the timestamp.
gsi_int day	A value that is converted into the day component of the timestamp.
gsi_int hour	A value that is converted into the hour component of the timestamp.
gsi_int minute	A value that is converted into the minute component of the timestamp.
gsi_int second	A value that is converted into the second component of the timestamp.

Return Value	Description
double	Represents a floating-point timestamp value.

Description

gsi_encode_timestamp() converts a year, month, day, hour, minute, and second into a floating-point timestamp value.

You can pass the *double* value returned from this function to the API function *gsi_set_timestamp()*, to set a timestamp on a *gsi_item* structure.

Caution The internal G2 clock has a limit of +/- 17 years from the time that G2 is started. Using a timestamp that extends beyond this limitation may cause unexpected results. You are encouraged to do validity checking of timestamps if you suspect that this may be an issue.

gsi_error_message

Returns the G2 Gateway error message text associated with an error number.

Synopsis

*gsi_char *gsi_error_message(error_code)*

Argument	Description
gsi_int error_code	The error code for which this function returns the associated text message.

Return Value	Description
*gsi_char **	Points to the text of the G2 Gateway error message for *error_code*.

Description

gsi_error_message() gets the text of the G2 Gateway error message for a particular error number.

The error message text returned by *gsi_error_message()* may contain place-holding character sequences such as *~S*, which are similar to the sequences like *%s* used by *printf()*. To obtain the full text of an error message, with current values rather than place-holding codes, use *gsi_last_error_message*.

gsi_establish_listener

Establishes a network listener that G2 can connect to when ready.

Synopsis

gsi_int gsi_establish_listener(network, port, exact)

Argument	Description
*gsi_char *network*	Specify "*TCP-IP*". You can specify only the first letter of the protocol name, in upper or lower case: "*T*" or "*t*".
*gsi_char *port*	Specify the number of the port on which the G2 process is listening for a connection to this bridge process.
gsi_int exact	A value of 1 directs *gsi_establish_listener()* to try to open a network connection at the specified port only. A value of 0 directs the new *gsi_establish_listener()* to try successive ports until it is able to establish a network connection.

Return Value	Description
gsi_int	Returns a value of 1 if the listener is successfully launched and 0 if the launch fails.

Description

Establishes a network listener that G2 can connect to when ready. G2 Gateway does the equivalent of calling this function during the call to *gsi_start*, after the user-written callback *gsi_set_up* returns, unless the command line switch *-nolistener* was specified.

You might like to use *gsi_establish_listener*() if you want to establish a network listener sometime after the G2 Gateway process is launched. In that case,

the command line that establishes G2 Gateway should use the -*nolistener* switch so that no listeners are established at start up.

gsi_establish_secure_listener

Attempts to establish a listener, using the SSL protocol.

Synopsis

```
gsi_int gsi_establish_secure_listener
    (network, port, exact, certificate)
```

Argument	Description
`gsi_char *network`	Specify "`TCP-IP`". You can specify only the first letter of the protocol name, in upper or lower case: "`T`" or "`t`".
`gsi_char *port`	Specify the number of the port on which the G2 process is listening for a connection to this bridge process.
`gsi_int exact`	A value of 1 directs `gsi_establish_listener()` to try to open a network connection at the specified port only. A value of 0 directs the new `gsi_establish_listener()` to try successive ports until it is able to establish a network connection.
`gsi_char* certificate`	The name of the certificate, which is a name in the certificate store on Windows or a filename on UNIX. The certificate can be NULL.

Return Value	Description
`gsi_int`	Returns a value of 1 if the listener is successfully launched and 0 if the launch fails.

Description

Behaves exactly like `gsi_establish_listerer` except establishes a secure connection.

Note that if a request is received from a clear text (insecure) connection, it is accepted as clear text.

gsi_extract_history

Extracts the history data associated with an item or registered item embedded item in an attribute.

Synopsis

void gsi_extract_history(item, values_address, timestamps_address, type_address)

void gsi_extract_history(attribute, values_address, timestamps_address, type_address)

Argument	Description
gsi_item item	The item of which this function extracts the history data.
gsi_attr attribute	The attribute containing an embedded item from which this function extracts the history data.
*void **values_address*	Address of an array of undetermined type, whose values represent the history.
*double **timestamps_address*	Address of an array of *double* whose values are floating-point timestamp values.
*gsi_int *type_address*	This function sets the value of this *gsi_int* to the type of the history.

Description

The function gsi_extract_history() extracts the history data associated with an item or registered item embedded item in an attribute.

The G2 Gateway type of *item* can be: *GSI_INTEGER_TAG*, *GSI_SYMBOL_TAG*, *GSI_FLOAT64_TAG*, *GSI_LOGICAL_TAG*, *GSI_STRING_TAG*, *GSI_QUANTITY_TAG*, or *GSI_VALUE_TAG*. The value array can be of type *gsi_int*, *gsi_char**, *double*, or *gsi_item*, in accordance with the type of the history values. You can use *gsi_history_type_of()* to return the data type of the history data associated with an item.

On UNIX and Windows, a timestamp value representing the number of seconds since midnight, January 1, 1970, GMT. Use the API function *gsi_decode_timestamp()* to decode a floating-point timestamp value into its component parts (month, day, year, and so on).

Note History data can be passed between a G2 Gateway bridge and a G2 only by means of remote procedure calls, and not through data service on GSI variables. For information about remote procedure calls, see Remote Procedure Calls.

Related Functions

Information from the related functions helps you create the arrays, values_ address and timestamps into which *gsi_extract_history()* stores its data.

Function	Description
gsi_history_count_of()	Returns the size of the two arrays.
gsi_extract_history_spec()	Returns the history-keeping specification for this variable or parameter within G2.

gsi_extract_history_spec

Extracts the history-keeping specification from an item or registered item embedded item in an attribute.

Synopsis

gsi_int gsi_extract_history_spec(item, maximum_count_address, maximum_age_address, minimum_interval_address)

Argument	Description
gsi_item item	The item from which this function extracts the history-keeping specification.
*gsi_int *maximum_count_address*	Address of a *gsi_int* whose value is set to the maximum count specified within the history-keeping-specification of the variable.
*gsi_int *maximum_age_address*	Address of a *gsi_int* whose value is set to the maximum age, in seconds, specified in the history-keeping-specification of the variable.
*gsi_int *minimum_interval_address*	Address of a *gsi_int* whose value is set to the minimum interval, in milliseconds, specified within the history-keeping-specification of the variable.

Return Value	Description
gsi_int	Not meaningful. Do not use.

Description

gsi_extract_history_spec() extracts history-keeping specification that G2 associates with an item or registered item embedded item in an attribute.

Related Functions

Function	Description
gsi_history_count_of()	Returns the number of history data values associated with an item.
gsi_extract_history()	Returns the history data values associated with an item.

gsi_flt_array_of

Returns the array of floating-point numbers stored in an item or registered item embedded item in an attribute.

Synopsis

*double *gsi_flt_array_of(item)*

*double *gsi_flt_array_of(reg-item)*

*double *gsi_flt_array_of(attribute)*

Argument	Description
gsi_item item	The *gsi_item* from which this function returns an array of floating-point numbers.
gsi_registered_item reg-item	The *gsi_registered_item* from which this function returns an array of floating-point numbers.
gsi_attr attribute	The *gsi_attr* containing an embedded *gsi_item* from which this function returns an array of floating-point numbers.

Return Value	Description
*double **	A one-dimensional C array of floating-point values.

Description

gsi_flt_array_of() returns the array of floating-point values stored in a *gsi_item* or *gsi_attr*. This function does not allocate any new memory.

If the argument to this function is neither a *gsi_item*, a *gsi_attr*, nor a *gsi_registered_item*, G2 Gateway signals an error.

To determine whether a *gsi_item* or *gsi_attr* represents an array of floating-point values, verify that the value returned by the API function *gsi_type_of()* is *GSI_FLOAT64_ARRAY_TAG*.

gsi_flt_list_of

Returns a one-dimensional C array that represents the list of floating-point values stored in an item or registered item embedded item in an attribute.

Synopsis

*double *gsi_flt_list_of(item)*

*double *gsi_flt_list_of(regitem)*

*double *gsi_flt_list_of(attribute)*

Argument	Description
gsi_item item	The *gsi_item* from which this function returns a C-array of floating-point numbers.
gsi_registered_item reg-item	The *gsi_registered_item* from which this function returns a C-array of floating-point numbers.
gsi_attr attribute	The *gsi_attr* containing an embedded *gsi_item* from which this function returns a C-array of floating-point numbers.

Return Value	Description
*double **	A one-dimensional C array of floating-point values.

Description

gsi_flt_list_of() returns a pointer to the list of floating-point values stored in a *gsi_item* or *gsi_attr*.

If the argument to this function is neither a *gsi_item* nor points to a *gsi_item*, G2 Gateway signals an error.

To determine whether a *gsi_item* or *gsi_attr* represents a floating-point list, verify that the value returned by the API function *gsi_type_of()* is *GSI_FLOAT64_LIST_TAG*.

To modify a *gsi_item* or *gsi_attr* so that it stores a floating-point list (represented as a C array), use the API function *gsi_set_flt_list()*.

This function does not allocate any new memory.

gsi_flt_of

Returns a C *double* that represents the floating-point value of an item, registered item, or embedded item in an attribute.

Synopsis

double gsi_flt_of(item)

double gsi_flt_of(reg-item)

double gsi_flt_of(attribute)

Argument	Description
gsi_item item	The *gsi_item* whose value is returned by this function.
gsi_registered_item reg-item	The *gsi_registered_item* of the item whose value is returned.
gsi_attr attribute	The *gsi_attr* containing the item whose value is returned.

Return Value	Description
double	Represents the value of the item, registered item, or the embedded item in an attribute.

Description

The function gsi_flt_of() returns the floating-point value of a *gsi_item*, *gsi_registered_item*, or *gsi_attr*.

The G2 Gateway type of the argument must be *GSI_FLOAT64_TAG*. Otherwise, G2 Gateway signals an error.

gsi_flush

Immediately flushes the G2 Gateway write buffer for the specified context.

Synopsis

void gsi_flush(context_number)

Argument	Description
gsi_int context_number	Context that identifies a connection to a GSI interface object in a connected G2 process.

Description

Both G2 Gateway and the host operating system buffer network input and output. This is done to achieve better performance, by minimizing the overhead spent calling the host operating system and by reducing the overhead associated with transmitting information on the network. However, this technique can result in a delay in the delivery of data to G2.

Calling *gsi_flush()* enables your G2 Gateway application to ensure that all of GSI's output buffers have been written out over the network.

For example, the application can call *gsi_flush()* after calling the API function *gsi_rpc_start()*, to ensure that the specified remote procedure is started in the connected G2 process as soon as possible.

gsi_handle_of

Given a *gsi_item* structure, returns a value of type *GSI_HANDLE_TAG* that represents a registered item. Given a G2 Gateway structure of type *gsi_registered_item* or *gsi_registration*, returns the handle for that structure.

Synopsis

gsi_int gsi_handle_of(item)

gsi_int gsi_handle_of(registered_item)

gsi_int gsi_handle_of(registration)

Argument	Description
gsi_item item	A G2 Gateway item, from which this function returns a value of the type *GSI_HANDLE_TAG*.
gsi_registered_item registered_item	A registered item of which this function returns the handle.
gsi_registration registration	A registration of which this function returns the handle.

Return Value	Description
gsi_int	An integer that stands for a particular G2 item registered in a given context.

Description

gsi_handle_of() returns the handle for a given G2 Gateway item, either a *gsi_item*, *gsi_registered_item*, or *gsi_registration*. The behavior of *gsi_handle_of()* depends upon the type of the argument.

If passed a *gsi_item*, *gsi_handle_of()* returns the value of the item, which must be of G2 Gateway type *GSI_HANDLE_TAG*.

If passed a *gsi_registration* or *gsi_registered_item*, *gsi_handle_of()* returns the handle of the registration or the handle of the registered item.

gsi_history_count_of

Returns the number of history data values that are associated with an item or embedded item in an attribute.

Synopsis

gsi_int gsi_history_count_of(item)

gsi_int gsi_history_count_of(attribute)

Argument	Description
gsi_item item	An item from which this function returns the number of history data values.
gsi_attr attribute	An attribute containing an embedded item from which this function returns the number of history data values.

Return Value	Description
gsi_int	Represents the number of history data values associated with *item* or *attribute*.

Description

gsi_history_count_of() returns the number of history data values associated with an item.

If G2 does not pass the history values associated with an item to G2 Gateway, *gsi_history_count_of()* returns a history count of 0 for that item, even though the item in G2 may have 1 or more history values. G2 can pass to G2 Gateway the history values associated with an item only through remote procedure calls, and not through data service operations such as those performed by *gsi_receive_registration()* and *gsi_get_data()*.

G2, by default, does not pass an object's system-defined attributes through remote procedure calls. You can override this default and specify in the remote procedure declaration in G2 that system-defined attributes such as the history values be passed. For information about how to pass the system-defined attributes, see *G2 Reference Manual*.

Note History data can be passed between a G2 Gateway bridge and a G2 only by means of remote procedure calls, and not through data service on GSI variables. For information about remote procedure calls, see Remote Procedure Calls.

gsi_history_type_of

An alias for *gsi_class_type_of()*.

For information about this function, see *gsi_class_type_of*.

gsi_identifying_attr_of

An alias for *gsi_item_of_identifying_attr_of()*.

For information about this function, see *gsi_item_of_identifying_attr_of*.

gsi_initialize_callbacks

Sends the addresses of the user's callback functions to G2 Gateway.

Synopsis

```
void gsi_initialize_callbacks(name1, funct1, ... (char *)0)
```

Argument	Description
`char *name1`	The name of the callback as it appears in `gsi_main.h`.
function_pointer _type1 funct1	A pointer to the callback. Each callback initialized must be specified by both a **name1* and an *funct1* argument.
`(char *)0`	Required to end the argument list.

Description

G2 Gateway provides a way to automatically initialize GSI 4.1 and G2 Gateway 5.0 callback functions when you compile your application. See Defining C Preprocessor Flags for instructions on doing so.

G2 Gateway does not provide a way to automatically initialize user-written, local, receiver, error receiver, and watchdog functions. You must use the `gsi_initialize_callbacks()` function to do so. See the following parts of the documentation for instructions:

- User-written functions specified in a call to the API function `gsi_watchdog()`. For information about this function, see `gsi_watchdog`.
- G2 Gateway local functions. For information about these functions, see Writing a G2 Gateway Local Function to be Called by G2.
- G2 Gateway receiver functions. For information about these functions, see Defining a Function to Receive Values Returned by G2.
- G2 Gateway error receiver functions. For information about these functions, see Defining a Function to Receive Error Values Returned by G2.

`gsi_initialize_callbacks()` takes a variable number of arguments. Each pair of arguments names a G2 Gateway callback and provides the address of the user-written version of that callback. If you omit a callback, G2 Gateway still links dynamically. However, G2 Gateway returns warning messages rather than invoke the omitted callbacks.

gsi_initialize_error_variable

Sets an error variable to the error code if an error occurs.

Synopsis

void gsi_initialize_error_variable(variable_address)

Argument	Description
*gsi_int * variable_address*	The address of the error variable whose value is set to the error code.

Description

G2 Gateway recognizes a set of minor errors that do not invoke the G2 Gateway error handling mechanism.

You can call *gsi_initialize_error_variable()* to set an error variable to the error code values of these minor errors, as they occur. When this function is used, the default and user-defined error handlers are not invoked. It is the responsibility of your user code to check the value of the error variable after every call to an API function.

It is recommended that your user code reset the error variable to 0 whenever it detects that the error variable has been set to a non-zero value.

gsi_initialize_for_win32

On a Windows platform, must be called prior to any other GSI API functions.

Synopsis

```
void gsi_initialize_for_win32(hInstance, lpCmdLine);
```

Argument	Description
HANDLE hInstance	Must correspond to the first parameter of *WinMain()*.
*char *lpCmdLine*	Must correspond to the third argument of *WinMain()*.

Description

If you use the supplied *gsimmain.c* or *gsi_main.c*, you do not need to add a call to this function. *gsi_initialize_for_win32* performs Windows-specific initialization. In a Windows application, it will also look for a *-log* command-line option. If the *-log* option is present, it redirects console output to a file whose name is specified by the argument following *-log*. For logging to work, in the call to *gsi_initialize_for_win32*, the first argument must correspond to the first parameter of *WinMain()*, and the second argument must correspond to the third parameter of *WinMain()*. Windows bridges must call this function prior to calling any other GSI API functions.

For more information about how to use this function, see Compiling and Linking G2 Gateway on Windows.

gsi_initiate_connection

Initiates a connection to a G2 process and causes G2 to create a GSI interface for that connection.

Synopsis

gsi_int gsi_initiate_connection(interface_name, class_name, keep_connection, network, host, port, rpis)

Argument	Description
*gsi_char *interface_name*	The name of the GSI interface created. G2 uses this GSI interface to communicate with the bridge process that calls *gsi_initiate_connection()*. If you specify *NULL* for *interface_name*, G2 creates an unnamed GSI interface. G2 can use the unnamed GSI interface for remote procedure calls, but not for data service for GSI variables.
*gsi_char *class_name*	The name of an existing class definition in the G2 KB. The GSI interface *interface_name* is created as an instance of this class. If you specify *NULL* for *class_name*, the GSI interface is created as an instance of the G2 class gsi-interface.
gsi_int keep_connection	*TRUE* causes the connection not to close when G2 is reset. The GSI interface is not deleted. *FALSE* causes the connection to close when G2 is reset. The GSI interface generated for this connection is deleted from the KB.
*gsi_char *network*	Specify "*TCP-IP*". You can specify only the first letter of the protocol name, in upper or lower case: "*T*" or "*t*".
*gsi_char *host*	Specify the host on which the G2 process is running.

Argument	Description
*gsi_char *port*	Specify the number of the port on which G2 process is listening for a connection to this bridge process.
*gsi_char *rpis*	The remote process initialization string. At startup time, G2 Gateway passes this string to the callback function *gsi_initialize_context()*.

Return Value	Description
0	The attempt to initiate a connection began successfully. If the attempt fails after this function returns 0, G2 Gateway signals an error. Check for errors to verify that the connection was successfully initiated. For information about error handling, see Error Handling.
1	The attempt to initiate a connection failed immediately.

Description

gsi_initiate_connection() initiates a connection to a G2 process running on a specified host. It causes G2 to create a GSI interface, which the G2 uses to communicate with the G2 Gateway process that calls *gsi_initiate_connection()*.

You can call *gsi_initiate_connection()* from within the callback *gsi_set_up()*, or from any subsequent function in your G2 Gateway user code. A bridge process can call *gsi_initiate_connection()* any number of times to initiate connections to G2 processes.

When *gsi_initiate_connection()* establishes a connection to a G2 process, G2 Gateway calls the callback function *gsi_initialize_context()*. If the connection cannot be established, G2 Gateway signals an error.

When you call *gsi_initiate_connection()* with *keep_connection* set to *FALSE*, G2 makes the GSI interface object transient and automatically deletes it when the connection is closed or G2 is reset. For this reason, G2 does not need the information provided by the *GSI-connection-configuration* attribute and therefore sets it to *none*.

You should be aware that if *keep_connection* is set to *FALSE*, but you made the GSI interface permanent [for example, by a rule that fires when the gsi-interface-status becomes 2 (the connection is active)] G2 does not automatically delete the GSI interface when the connection is closed or G2 is reset. You must first make the permanent GSI interface transient and then delete it.

You can configure a G2 KB to prohibit G2 Gateway processes from initiating connections to the KB. To do this, add a configuration statement to your KB Configuration system table that prohibits connect access to G2 Gateway. For information about how to specify network security for a G2 process, see *G2 Reference Manual*.

Related Functions

Function	Description
gsi_initialize_context()	Initialize a connection between a GSI interface in G2 and G2 Gateway, or reject the connection.
gsi_set_up()	Perform G2 Gateway-related operations that need to be performed only once during the lifetime of the bridge process.

gsi_initiate_connection_with_user_data

Initiates a connection to a G2 process, causes G2 to create a GSI interface for the connection, and associates user data with the connection.

Synopsis

gsi_int gsi_initiate_connection_with_user_data (interface_name, class_name, keep_connection, network, host, port, rpis, context_user_data)

Argument	Description
*gsi_char *interface_name*	The name of the GSI interface created. G2 uses this GSI interface to communicate with the bridge process that calls *gsi_initiate_connection_with_user_data()*.
	If you specify *NULL* for *interface_name*, G2 creates an unnamed GSI interface. G2 can use the unnamed GSI interface for remote procedure calls, but not for data service for GSI variables.
*gsi_char *class_name*	The name of an existing class definition in the G2 KB. The GSI interface *interface_name* is created as an instance of this class.
	If you specify *NULL* for *class_name*, the GSI interface is created as an instance of the G2 class gsi-interface.
gsi_int keep_connection	Specify *FALSE*. This argument is not supported in this release.
*gsi_char *network*	Specify "*TCP-IP*".
	You can specify only the first letter of the protocol name, in upper or lower case: "*T*" or "*t*".
*gsi_char *host*	Specify the host on which the G2 process is running.
*gsi_char *port*	Specify the number of the port on which G2 process is listening for a connection to this bridge process.

Argument	Description
gsi_char ***rpis*	The remote process initialization string. At startup time, G2 Gateway passes this string to the callback function *gsi_initialize_context()*.
gsi_context_user_data_type context_user_data	The user data associated with the context initiated by this call to *gsi_initiate_connection_with_user_data*().

Return Value	Description
0	The attempt to initiate a connection began successfully. If the attempt fails after this function returns 0, G2 Gateway signals an error. Check for errors to verify that the connection was successfully initiated. For information about error handling, see Error Handling.
1	The attempt to initiate a connection failed immediately.

Description

The function *gsi_initiate_connection_with_user_data*() enables a G2 Gateway bridge to initiate a connection and to associate context user data with the connection to identify its origin, purpose, or other characteristics.

You can call *gsi_initiate_connection_with_user_data()* from within the callback *gsi_set_up()*, or from any subsequent function in your G2 Gateway user code. A bridge process can call *gsi_initiate_connection_with_user_data()* any number of times to initiate connections to G2 processes.

When *gsi_initiate_connection_with_user_data()* establishes a connection to a G2 process, G2 Gateway calls the callback function *gsi_initialize_context()*. If the connection cannot be established, G2 Gateway signals an error.

You can configure a G2 KB to prohibit G2 Gateway processes from initiating connections to the KB. To do this, add a configuration statement to your KB Configuration system table that prohibits connect access to G2 Gateway. For information about how to specify network security for a G2 process, see *G2 Reference Manual*.

Use the API function *gsi_context_user_data()* to return the context user data associated with a specified context. You can associate user data with an existing context using the API function *gsi_set_context_user_data()*.

gsi_initiate_secure_connection

Initiates a secure connection to a G2 process and causes G2 to create a GSI interface for that connection.

Synopsis

gsi_int gsi_initiate_connection(interface_name, class_name, keep_connection, network, host, port, rpis)

Argument	Description
*gsi_char *interface_name*	The name of the GSI interface created. G2 uses this GSI interface to communicate with the bridge process that calls *gsi_initiate_connection()*. If you specify *NULL* for *interface_name*, G2 creates an unnamed GSI interface. G2 can use the unnamed GSI interface for remote procedure calls, but not for data service for GSI variables.
*gsi_char *class_name*	The name of an existing class definition in the G2 KB. The GSI interface *interface_name* is created as an instance of this class. If you specify *NULL* for *class_name*, the GSI interface is created as an instance of the G2 class gsi-interface.
gsi_int keep_connection	*TRUE* causes the connection not to close when G2 is reset. The GSI interface is not deleted. *FALSE* causes the connection to close when G2 is reset. The GSI interface generated for this connection is deleted from the KB.
*gsi_char *network*	Specify "*TCP-IP*". You can specify only the first letter of the protocol name, in upper or lower case: "*T*" or "*t*".
*gsi_char *host*	Specify the host on which the G2 process is running.

Argument	Description
*gsi_char *port*	Specify the number of the port on which G2 process is listening for a connection to this bridge process.
*gsi_char *rpis*	The remote process initialization string. At startup time, G2 Gateway passes this string to the callback function *gsi_initialize_context()*.

Return Value	Description
0	The attempt to initiate a connection began successfully. If the attempt fails after this function returns 0, G2 Gateway signals an error. Check for errors to verify that the connection was successfully initiated. For information about error handling, see Error Handling.
1	The attempt to initiate a connection failed immediately.

Description

Behaves exactly like *gsi_initiate_connection*, but tries to make a secure connection to G2 with SSL.

gsi_initiate_secure_connection_with_user_data

Initiates a secure connection to a G2 process, causes G2 to create a GSI interface for the connection, and associates user data with the connection.

Synopsis

gsi_int gsi_initiate_connection_with_user_data (interface_name, class_name, keep_connection, network, host, port, rpis, context_user_data)

Argument	Description
*gsi_char *interface_name*	The name of the GSI interface created. G2 uses this GSI interface to communicate with the bridge process that calls *gsi_initiate_connection_with_user_data()*.
	If you specify *NULL* for *interface_name*, G2 creates an unnamed GSI interface. G2 can use the unnamed GSI interface for remote procedure calls, but not for data service for GSI variables.
*gsi_char *class_name*	The name of an existing class definition in the G2 KB. The GSI interface *interface_name* is created as an instance of this class.
	If you specify *NULL* for *class_name*, the GSI interface is created as an instance of the G2 class gsi-interface.
gsi_int keep_connection	Specify *FALSE*. This argument is not supported in this release.
*gsi_char *network*	Specify "*TCP-IP*".
	You can specify only the first letter of the protocol name, in upper or lower case: "*T*" or "*t*".
*gsi_char *host*	Specify the host on which the G2 process is running.

Argument	Description
gsi_char ***port*	Specify the number of the port on which G2 process is listening for a connection to this bridge process.
gsi_char ***rpis*	The remote process initialization string. At startup time, G2 Gateway passes this string to the callback function *gsi_initialize_context()*.
gsi_context_user_data_type context_user_data	The user data associated with the context initiated by this call to *gsi_initiate_connection_with_user_data*().

Return Value	Description
0	The attempt to initiate a connection began successfully. If the attempt fails after this function returns 0, G2 Gateway signals an error. Check for errors to verify that the connection was successfully initiated. For information about error handling, see Error Handling.
1	The attempt to initiate a connection failed immediately.

Description

Behaves exactly like *gsi_initiate_connection_with_user_data*, but tries to make a secure connection to G2 with SSL.

gsi_install_error_handler

Invokes *gsi_initialize_callbacks()* to initialize the *gsi_error_handler()* callback function.

Synopsis

void gsi_install_error_handler(gsi_error_handler)

Argument	Description
gsi_error_handler	The callback function to install.

Description

gsi_install_error_handler() calls *gsi_initialize_callbacks()* to install the callback *gsi_error_handler()*.

You can complete the code of *gsi_error_handler*() to perform any customized error handling required by your application. For information about this callback, see *gsi_error_handler*.

gsi_int_array_of

Returns a pointer to a C array of *gsi_int* values, which corresponds to the values in an item, registered item, or embedded item in an attribute, whose value must be of G2 Gateway type *GSI_INTEGER_ARRAY_TAG*.

Synopsis

*gsi_int *gsi_int_array_of(item)*

*gsi_int *gsi_int_array_of(registered_item)*

*gsi_int *gsi_int_array_of(attribute)*

Argument	Description
gsi_item item	An item to whose values this function returns a pointer.
gsi_registered_item registered_item	A registered item to whose values this function returns a pointer.
gsi_attr attribute	An attribute containing an embedded item to whose values this function returns a pointer.

Return Value	Description
*gsi_int **	Points to a C array of integer values that correspond to the elements of the G2 Gateway type *GSI_INTEGER_ARRAY_TAG*.

Description

gsi_int_array_of() returns a C array of *gsi_int* values, representing the elements of the *GSI_INTEGER_ARRAY_TAG* value specified by the argument to this function.

The values in this array correspond to the elements of the *GSI_INTEGER_ARRAY_TAG* value of a *gsi_item*, *gsi_registered_item*, or *gsi_attr*.

The argument *item*, *registered_item*, or *attribute* must be of G2 Gateway type *GSI_INTEGER_ARRAY_TAG*; otherwise, G2 Gateway signals an error.

This function does not allocate any memory.

gsi_int_list_of

Returns a C array of *gsi_int* values, which corresponds to the values in an item, registered item, or embedded item in an attribute, whose value must be of G2 Gateway type *GSI_INTEGER_LIST_TAG*.

Synopsis

*gsi_int *gsi_int_list_of(item)*

*gsi_int *gsi_int_list_of(registered_item)*

*gsi_int *gsi_int_list_of(attribute)*

Argument	Description
gsi_item item	An item to whose values this function returns a pointer.
gsi_registered_item registered_item	A registered item to whose values this function returns a pointer.
gsi_attr attribute	An attribute containing an embedded item to whose values this function returns a pointer.

Return Value	Description
*gsi_int **	A C array of integer values that correspond to the elements of the G2 Gateway type *GSI_INTEGER_LIST_TAG*.

Description

gsi_int_list_of() returns an array of *gsi_int* values. The values in this array correspond, in order, to elements of the *GSI_INTEGER_LIST_TAG* value of a *gsi_item*, *gsi_registered_item*, or *gsi_attr*.

The argument *item*, *registered_item* or *attribute* must be of G2 Gateway type *GSI_INTEGER_LIST_TAG*; otherwise, G2 Gateway signals an error.

This function does not allocate any memory.

gsi_int_of

Returns the integer value of an item, registered item, or embedded item in an attribute.

Synopsis

gsi_int gsi_int_of(item)

gsi_int gsi_int_of(registered_item)

gsi_int gsi_int_of(attribute)

Argument	Description
gsi_item item	An item whose integer value is returned by this function.
gsi_registered_item registered_item	A registered item whose integer value is returned by this function.
gsi_attr attribute	An attribute containing an embedded item whose integer value is returned by this function.

Return Value	Description
gsi_int	Represents the value of G2 Gateway type *GSI_INTEGER_TAG* stored in *item*, *registered_item*, or *attribute*.

Description

gsi_int_of() returns the value of an item, registered item, or embedded item in an attribute that is of G2 Gateway type *GSI_INTEGER_TAG*. The value is returned as a *gsi_int*.

If the argument is not of G2 Gateway type *GSI_INTEGER_TAG*, G2 Gateway signals an error.

gsi_interval_of

Returns the current default update interval associated with a registered item or an item registration.

Synopsis

`double gsi_interval_of(registered_item)`

`double gsi_interval_of(registration)`

Argument	Description
`gsi_registered_item`*registered_item*	The registered item whose default update interval is returned by this function.
`gsi_registration` *registration*	The registration whose default update interval is returned by this function.

Return Value	Description
`double`	Represents the default update interval in *registered_item*, given in seconds.

Description

`gsi_interval_of()` returns the default update interval of a registered item.

To change the default update interval in a registered item, use the API function `gsi_set_interval()`.

gsi_is_item

Determines whether a specified *gsi_item* represents an item or a value in G2.

Synopsis

gsi_int gsi_is_item(item)

Argument	Description
gsi_item item	The item examined by this function.

Return Value	Description
gsi_int	A non-zero value if *item* is an object that inherits from the G2 class item, or 0 if it represents a value. The *gsi_item* represents an object that inherits from item if the *class name* component of the *gsi_item* contains the name of a G2 class. The *gsi_item* represents a value in G2 if the *class name* component is set to *NULL*.

Description

gsi_is_item() is useful for determining whether an argument received from G2 by a G2 Gateway local function represents an object that inherits from the G2 class item, or a value of one of the G2 value types.

If *gsi_is_item()* indicates that an argument represents an object, your user code can traverse the object to access its attributes individually.

For more information about this use of *gsi_is_item()*, see How a Local Function Can Process Argument Arrays Received from G2.

gsi_item_of_attr

Returns the *gsi_item* structure embedded in a *gsi_attr* structure.

Synopsis

gsi_item gsi_item_of_attr(attribute)

Argument	Description
gsi_attr attribute	An attribute containing an embedded item.

Return Value	Description
gsi_item	Represents the embedded item in *attribute*.

Description

gsi_item_of_attr() returns the *gsi_item* contained in the specified *gsi_attr* structure.

gsi_item_of_attr_by_name

Returns the *gsi_item* structure contained in a specified *gsi_attr* structure.

Synopsis

gsi_item gsi_item_of_attr_by_name(item, search_name)

gsi_item gsi_item_of_attr_by_name(attr, search_name)

Argument	Description
gsi_item item	If you specify a *gsi_item*, this argument represents an item one of whose attributes, specified by *search_name*, contains the item that this function returns.
gsi_attr attr	If you specify a *gsi_attr*, this argument represents an attribute that contains an embedded item. In this case, it is the embedded item that has the attribute, specified by *search_name*, that contains the item returned by this function.
gsi_symbol search_name	The name of the attribute that contains the embedded *gsi_item* returned by this function. Specify this name in uppercase letters, to correspond to the uppercase letters ordinarily used in G2 identifiers, which are of the G2 type symbol.

Return Value	Description
gsi_item	The *gsi_item* structure embedded in the *gsi_attr* structure whose name is *search_name*.

Description

The function gsi_item_of_attr_by_name() returns the *gsi_item* embedded in the *gsi_attr* structure whose name is specified by *search_name*. If no such attribute exists, G2 Gateway signals an error.

Related Functions

Function	Description
gsi_attr_count_of()	Returns the number of attributes in a *gsi_item* or *gsi_attr*.
gsi_attrs_of()	Obtains the attributes in a *gsi_item* or *gsi_attr*.
gsi_set_attrs()	Changes the attributes in a *gsi_item* or *gsi_attr*
gsi_set_attr_by_name()	Changes a specific attribute in a *gsi_item* or *gsi_attr*.

gsi_item_of_identifying_attr_of

Returns a *gsi_item* from an identifying attribute stored in a *gsi_registration* structure.

Synopsis

gsi_item gsi_item_of_identifying_attr_of(registration, attribute_index)

Argument	Description
gsi_registration registration	The item registration.
gsi_int attribute_ index	An integer, between 1 and 6 inclusive, that specifies one of the six identifying attributes. If you specify a value greater than 6, or greater than the number of identifying attributes, G2 Gateway returns an error.

Return Value	Description
gsi_item	The identifying attribute specified by *attribute_ index*.

Description

gsi_item_of_identifying_attr_of() returns one of the identifying attributes stored in the *identifying attributes* component of a *gsi_registration* structure.

When you configure a GSI interface, you designate the attributes to be used as the identifying attributes for instances of each class of GSI variable that uses the GSI interface. For information about how to specify the identifying attribute of a class of GSI variable, see Identifying-Attributes Attribute.

The identifying attributes stored in a *gsi_registration* structure do not include the names of the attributes.

gsi_item_of_registered_item

Returns the *gsi_item* structure that is pointed to by a specified *gsi_registered_item* structure.

Synopsis

gsi_item gsi_item_of_registered_item(registered_item)

Argument	Description
gsi_registered_item registered_item	The item structure.

Return Value	Description
gsi_item	The item structure contained in *registered_item*.

Description

gsi_item_of_registered_item() returns the *gsi_item* that is contained in *registered_item*.

gsi_kill_context

Shuts down a G2 Gateway context immediately.

Synopsis

void gsi_kill_context(context_number)

Argument	Description
gsi_int context_number	The context that this function shuts down.

Description

gsi_kill_context() shuts down a specified context and performs operations necessary to shut down the external system and clean up the bridge process. When the context is shut down, G2 Gateway calls the callback function *gsi_shutdown_context()* to perform any customized operations associated with the shutdown that your application requires.

G2 Gateway calls *gsi_shutdown_context()* when *any* event closes a context — such as a G2 shutting down or being reset, or a connection being broken. For information about this callback function, see *gsi_shutdown_context*.

gsi_last_error

Returns the error number of the last error condition to which the global error flag was set by an API function in your G2 Gateway application.

Synopsis

```
gsi_int gsi_last_error()
```

Return Value	Description
gsi_int	Represents the number of the error condition most recently encountered in the G2 Gateway application.

Description

gsi_last_error() returns the error number of the error condition most recently encountered in the G2 Gateway application.

For information about how to use the global error flag, see Checking the Global Error Flag.

gsi_last_error_call_handle

Returns the call identifier that G2 generated and passed to the G2 Gateway local function that invokes an error handler function.

Synopsis

gsi_call_identifier_type gsi_last_error_call_handle()

Return Value	Description
gsi_call_identifier_type	The call-identifier received from G2 by the local function that invoked the error handler function.

Description

Use *gsi_last_error_call_handle()* inside a user-defined error-handler function that is invoked during an API call made by a G2 Gateway local function.

This function returns the call-identifier of the local function that invoked the error handler. You can pass the call-identifier to the API function *gsi_rpc_return_error_values()*, which enables you to control the error message that is sent back to G2.

gsi_last_error_message

Returns the message text of the last reported error.

Synopsis

```
gsi_char *gsi_last_error_message()
```

Return Value	Description
*gsi_char**	The text of the last error message.

Description

Use *gsi_last_error_message()* rather than *gsi_error_message()*, which can return formatting templates rather than the simple text of the error message.

gsi_listener_socket

Returns the UNIX file descriptor associated with the G2 Gateway bridge's TCP listener.

Synopsis

```
gsi_int gsi_listener_socket()
```

Return Value	Description
gsi_int	The file descriptor associated with the bridge's TCP listener, or -1, if not such listener exists.

Description

gsi_listener_socket() returns the UNIX file descriptor associated with the G2 Gateway bridge's TCP listener. If the bridge has no listener, for example, if the bridge has been started without a TCP listener, this function returns -1.

gsi_log_array_of

Returns the array of truth-values stored in an item or registered item embedded item in an attribute.

Synopsis

*gsi_int *gsi_log_array_of(item)*

*gsi_int *gsi_log_array_of(registered_item)*

*gsi_int *gsi_log_array_of(attribute)*

Argument	Description
gsi_item item	An item from which this function returns the array of truth-values.
gsi_registered_item registered_item	A registered item from which this function returns the array of truth-values.
gsi_attr attribute	An attribute containing an item from which this function returns the array of truth-values.

Return Value	Description
*gsi_int **	A C array of type *gsi_int*. Each element represents a truth-value, ranging from *GSI_FALSE* (*-1000*) for completely false to *GSI_TRUE* (*+1000*) for completely true.

Description

gsi_log_array_of() returns a pointer to the array of truth-values stored in a *gsi_item* or *gsi_attr*.

If the argument to this function is neither a *gsi_item* nor a data structure that points to a *gsi_item*, G2 Gateway signals an error.

To determine whether a *gsi_item*, *gsi_registered_item*, or *gsi_attr* represents a truth-value array, verify that the value returned by the API function *gsi_type_of()* is *GSI_LOGICAL_ARRAY_TAG*.

To modify a *gsi_item* or *gsi_attr* so that it stores a truth-value array, use the API function *gsi_set_log_array()*.

gsi_log_list_of

Returns a one-dimensional C array that represents the list of truth-values stored in an item or registered item embedded item in an attribute.

Synopsis

*gsi_int *gsi_log_list_of(item)*

*gsi_int *gsi_log_list_of(registered_item)*

*gsi_int *gsi_log_list_of(attribute)*

Argument	Description
gsi_item item	An item from which this function returns a C array representing a list of truth values.
gsi_registered_item registered_item	A registered item from which this function returns a C array representing a list of truth values.
gsi_attr attribute	An attribute containing an embedded item.

Return Value	Description
*gsi_int **	A one-dimensional C array of type *gsi_int*. Each element represents a truth-value, ranging from *GSI_FALSE* (*-1000*) for completely false to *GSI_TRUE* (*+1000*) for completely true.

Description

gsi_log_list_of() returns the list of truth-values stored in a *gsi_item* or *gsi_attr*.

If the argument to this function is neither a *gsi_item* nor a data structure that points to a *gsi_item*, G2 Gateway signals an error.

This function does not allocate any new memory — that is, it returns the actual array stored in a *gsi_item*, *gsi_registered_item*, or *gsi_attr*, not to a copy.

To determine whether a *gsi_item*, *gsi_registered_item*, or *gsi_attr* represents a truth-value list, verify that the value returned by the API function *gsi_type_of()* is *GSI_LOGICAL_LIST_TAG*.

To modify a *gsi_item*, *gsi_registered_item*, or *gsi_attr* so that it stores a truth-value list, use the API function *gsi_set_log_list()*.

gsi_log_of

Returns a *gsi_int* that represents the value of an item, registered item, or embedded item in an attribute whose value is of G2 Gateway type *GSI_LOGICAL_TAG*.

Synopsis

gsi_int gsi_log_of(item)

gsi_int gsi_log_of(registered_item)

gsi_int gsi_log_of(attribute)

Argument	Description
gsi_item item	An item whose value is returned by this function.
gsi_registered_item registered_item	A registered item whose value is returned by this function.
gsi_attr attribute	An attribute containing an embedded item whose value is returned by this function.

Return Value	Description
gsi_int	A truth-value, ranging from *GSI_FALSE* (*-1000*) for completely false to *GSI_TRUE* (*+1000*) for completely true.

Description

gsi_log_of() returns the value of a *gsi_item*, *gsi_registered_item*, or *gsi_attr*.

The G2 Gateway type of the argument must be *GSI_LOGICAL_TAG*; otherwise, G2 Gateway signals an error.

gsi_long_of

Returns the long value of an item, registered item, or embedded item in an attribute.

Synopsis

gsi_long gsi_long_of(item)

gsi_long gsi_long_of(registered_item)

gsi_long gsi_long_of(attribute)

Argument	Description
gsi_item item	An item whose integer value is returned by this function.
gsi_registered_item registered_item	A registered item whose integer value is returned by this function.
gsi_attr attribute	An attribute containing an embedded item whose integer value is returned by this function.

Return Value	Description
gsi_long	Represents the value of G2 Gateway type *GSI_LONG_TAG* stored in *item*, *registered_item*, or *attribute*.

Description

gsi_long_of() returns the value of an item, registered item, or embedded item in an attribute that is of G2 Gateway type *GSI_LONG_TAG*. The value is returned as a *gsi_long*.

If the argument is not of G2 Gateway type *GSI_LONG_TAG*, G2 Gateway signals an error.

gsi_make_array

Allocates an array containing space for a specified number of *gsi_items*.

Synopsis

*gsi_item *gsi_make_array(count)*

Argument	Description
gsi_int count	The number of elements in the array allocated by this function call.

Return Value	Description
*gsi_item **	The array of *gsi_item* allocated by this function call.

Description

Note, *gsi_make_array()* doesn't create the *gsi_items*, only the array itself. You populate the array using *gsi_make_item()* or a related API function.

You are responsible for deallocating any arrays that you allocate using *gsi_make_array()*. G2 Gateway does not deallocate it automatically. You can deallocate an array using *gsi_reclaim_array()*.

G2 Gateway automatically deallocates any arrays that it allocates itself. Do not attempt to deallocate arrays automatically allocated by G2 Gateway.

gsi_make_attrs

Allocates one or more instances of the *gsi_attr* structure.

Synopsis

*gsi_attr *gsi_make_attrs(count)*

Argument	Description
gsi_int count	Number of instances of *gsi_attr* to allocate.

Return Value	Description
*gsi_attr **	The allocated C array of *gsi_attr* structures.

Description

gsi_make_attrs() allocates *gsi_attr* instances for the use of your G2 Gateway bridge process. Use array-index pointer arithmetic to access each element of the allocated array.

You can set the *item* component of any *gsi_attr* structure to reference any *gsi_item* created by G2 Gateway itself or by the API function *gsi_make_items()*, using the API function *gsi_set_item_of_attr()*.

It is good practice to deallocate the *gsi_attr* structures that you allocate using *gsi_make_attrs()* as soon as these structures are no longer needed by your G2 Gateway bridge process.

Use the API function *gsi_reclaim_attrs()* to deallocate *gsi_attr* structures. You need to deallocate only those *gsi_attr* structures that you have allocated by making calls to *gsi_make_attrs()*or *gsi_make_attrs_with_items()*. You do not need to deallocate the *gsi_attr* structures that G2 Gateway has allocated automatically, and should not attempt to do this.

gsi_make_attrs_with_items

Allocates one or more instances of the *gsi_attr* structure, each with its own *gsi_item* structure.

Synopsis

*gsi_attr *gsi_make_attrs_with_items(count)*

Argument	Description
gsi_int count	Number of instances of *gsi_attr* to allocate.

Return Value	Description
*gsi_attr **	A newly allocated C array of *gsi_attr* structures.

Description

gsi_make_attrs_with_items() allocates one or more instances of *gsi_attr*. Each *gsi_attr* contains its own *gsi_item* structure.

It is good practice to deallocate the *gsi_attr* structures that you allocate using *gsi_make_attrs_with_items()* as soon as these structures are no longer needed by your G2 Gateway bridge process.

Use the API function *gsi_reclaim_attrs_with_items()* to deallocate this array (or any array of *gsi_attr* instances that your G2 Gateway application has populated with *gsi_item* instances). You need to deallocate only those *gsi_attr* structures that you have allocated by making calls to *gsi_make_attrs()* or *gsi_make_attrs_with_items()*. You do not need to deallocate the *gsi_attr* structures that G2 Gateway has allocated automatically, and should not attempt to do this.

gsi_make_item

Allocates a single *gsi_item* structure.

Synopsis

gsi_item gsi_make_item()

Return Value	Description
gsi_item	The *gsi_item* allocated by this function call.

Description

You are responsible for deallocating any items that you allocate using *gsi_make_item()*. G2 Gateway does not deallocate it automatically. You can deallocate an array using *gsi_reclaim_item()*.

G2 Gateway automatically deallocates any items that it allocates itself. Do not attempt to deallocate items automatically allocated by G2 Gateway.

gsi_make_items

Allocates an array of one or more instances of *gsi_item*.

Synopsis

*gsi_item *gsi_make_items(count)*

Argument	Description
gsi_int count	Number of instances of *gsi_item* to allocate.

Return Value	Description
*gsi_item **	A newly allocated C array of *gsi_item* structures.

Description

gsi_make_items() allocates *gsi_item* instances for use in the user code of your G2 Gateway application. Use array-index pointer arithmetic to access each element of the allocated array.

It is good practice to deallocate the *gsi_item* structures that you allocate using *gsi_make_items()* as soon as these structures are no longer needed by your G2 Gateway bridge process.

Use the API function *gsi_reclaim_items()* to deallocate *gsi_item* structures. You need to deallocate only those *gsi_item* structures that you have allocated by making calls to *gsi_make_items()*. You do not need to deallocate the *gsi_item* structures that G2 Gateway has allocated automatically, and should not attempt to do this.

gsi_make_registered_items

Allocates one or more instances of the *gsi_registered_item* structure.

Synopsis

*gsi_registered_item *gsi_make_registered_items(count)*

Argument	Description
gsi_int count	Number of instances of *gsi_registered_item* to allocate.

Return Value	Description
*gsi_registered_item **	A newly allocated C array of *gsi_registered_item* structures.

Description

gsi_make_registered_items() allocates an array of one or more *gsi_registered_item* structures. You can pass arrays allocated by this function to the API functions *gsi_return_values()*, *gsi_return_timed_values()*, *gsi_return_attrs()*, and *gsi_return_timed_attrs()*.

You can deallocate the *gsi_registered_item* structures that you allocate using *gsi_make_registered_items()* as soon as these structures are no longer needed by your G2 Gateway bridge process. However, do not deallocate the *gsi_registered_item* structures until any API function to which the structures have been passed completes and returns. For example, if you pass an array of *gsi_registered_item* to *gsi_return_values()*, do not deallocate this array until *gsi_return_values()* returns.

Use the API function *gsi_reclaim_registered_items()* to deallocate *gsi_registered_item* structures. You need to deallocate only those *gsi_registered_item* structures that you have allocated by making calls to *gsi_make_registered_items()*. You do not need to deallocate the *gsi_registered_item* structures that G2 Gateway has allocated automatically, and should not attempt to do this.

gsi_make_symbol

Returns a symbol, given the name of that symbol.

Synopsis

gsi_symbol gsi_make_symbol(symbol-name)

Argument	Description
*gsi_char *symbol-name*	The name of the symbol.

Return Value	Description
gsi_symbol	The symbol that corresponds to the specified symbol name.

Description

When the *GSI_NEW_SYMBOL_API* runtime option is set, *gsi_make_symbol()*, results in a (*void **) pointer whose contents does not change throughout the lifetime of the GSI process. Calling *gsi_make_symbol()* at a later time with a string having the same contents produces exactly the same result.

When the *GSI_NEW_SYMBOL_API* runtime option is *not* set, *gsi_make_symbol()* returns a string that is a copy of its argument.

gsi_name_of

Returns a symbol representing the name of a specified GSI item.

Synopsis

gsi_symbol gsi_name_of(item)

gsi_symbol gsi_name_of(attr)

gsi_symbol gsi_name_of(reg)

Argument	Description
gsi_item item	If you specify *gsi_item*, the name of that item is returned.
gsi_attr attr	If you specify a *gsi_attr*, the name of the embedded *gsi_item* in that attribute is returned.
gsi_registration reg	If you specify a *gsi_registration*, the name of the *gsi_item* for which this is the registration is returned.

Return Value	Description
gsi_symbol	A symbol giving the name of the specified data structure. The symbol persists only as long as the data structure with which it is associated. If your user code needs to keep the symbol for longer than the life-span of the data structure, it must copy the symbol into memory it allocates either using *malloc()* or via one of the *make_xxx* G2 Gateway API functions. **Note**: The return value is *NULL* if the argument passed to this function is a *gsi_item* extracted from a *gsi_registered_item* by *gsi_item_of_registered_item()*, and this *gsi_registered_item* was obtained from a callback function, such as *gsi_set_data()*.

Description

gsi_name_of() returns the value of the *name* attribute of a *gsi_item*. A call to this API function can reference the *gsi_item* structure directly, or through a *gsi_registration* or *gsi_attr* structure.

Caution If the specified item has no associated class or is not named, the return value is a null pointer. On some systems, passing a null pointer to *printf()* causes a segmentation violation. For this reason, you may want to verify that the return value of *gsi_name_of()* is not a null pointer before you pass it to *printf()*. For example:

```
gsi_symbol char_temp;
char_temp = gsi_name_of(item);
if (char_temp)
printf("\n name: %s", char_temp);
else
printf("\n name: NULL_PTR");
```

gsi_option_is_set

Returns whether a particular G2 Gateway global run-time option is set.

Synopsis

gsi_int gsi_option_is_set(option)

Argument	Description
gsi_int option	Symbolic constant that represents a G2 Gateway global run-time option.

Return Value	Description
gsi_int	Either 1 (*TRUE*) or 0 (*FALSE*), depending on whether the specified option is currently set or reset, respectively.

Description

gsi_option_is_set() returns a value that indicates whether a particular G2 Gateway global run-time option is set.

For *option*, specify the symbolic constant that represents a G2 Gateway runtime option, as shown in the table below.

Global Run-time Option	Purpose
GSI_NEW_SYMBOL_API	Enables API functions to access symbols efficiently. Use of this option is recommended if you user code includes calls to functions that access symbols.
GSI_NO_SIGNAL_HANDLERS	Prevents G2 Gateway from registering its own signal handlers with the operating system.
GSI_NON_C	When set, directs G2 Gateway to use pass-by-reference when calling the G2 Gateway callback functions. When reset (the default), directs G2 Gateway to use pass-by-value.

Global Run-time Option	Purpose
GSI_ONE_CYCLE	When set, directs G2 Gateway to run in One-Cycle mode. When reset, directs G2 Gateway to run in Continuous mode
GSI_STRING_CHECK	Determines whether G2 Gateway automatically filters all strings passed between G2 Gateway and G2 (that is, automatically converts all carriage-returns, at-signs, tildes, and backslashes, as described in the *G2 Reference Manual*.)
GSI_SUPPRESS_OUTPUT	When set, directs GSI not to send information and error messages to standard output. When reset, (the default) GSI sends information and error messages to standard output.
GSI_TRACE_RUN_STATE	Prints a message whenever the flow of control enters or leaves G2 Gateway. If the *gsi_run_state_change()* callback is initialized, it prints the message before this callback is called.
GSI_USER_DATA_FOR_CALLBACKS	Enables the user of user_data arguments in remote procedure calls made from the bridge to G2. For information about this option, see Procedure User Data for Remote Procedure Calls.
GSI_WIDE_STRING_API	Enables the use of the wide string character type. For information about this type, see Wide String Type.

gsi_owner_of

Indicates whether user code or G2 Gateway allocated a specified item.

Synopsis

gsi_int gsi_owner_of(item)

gsi_int gsi_owner_of(regitem)

gsi_int gsi_owner_of(attr)

gsi_int gsi_owner_of(reg)

Argument	Description
gsi_item item	A *gsi_item*.
gsi_registered_item regitem	A *gsi_registered_item*.
gsi_attr attr	A *gsi_attr*.
gsi_registration reg	A *gsi_registration*.

Return Value	Description
gsi_int	Indicates whether the specified item was allocated by user code or by G2 Gateway. Can be any of the following values: 0 (*GSI_OWNER_IS_USER*):: The item was allocated by user code, and the user code is responsible for deallocating it. 1 (*GSI_OWNER_IS_GSI*):: The item was allocated automatically by G2 Gateway, which will deallocate it automatically when it is no longer needed. 2 (*GSI_OWNER_IS_CONTEXT*):: The item was allocated automatically on behalf of the context by G2 Gateway, which will deallocate the item automatically when it is no longer needed.

Description

The function *gsi_owner_of*() enables your user code to determine whether it has memory management responsibility for a specified item. If *gsi_owner_of()* indicates that the user code allocated the item, the user code is responsible for deallocating that item when it no longer needs the item. If *gsi_owner_of()* indicates that the item was generated automatically by G2 Gateway, the item will be deallocated automatically by G2 Gateway when it is no longer needed, and the user code has no memory management responsibility for that item. For more information about the memory management responsibilities of G2 Gateway user code, see Memory Management Responsibilities of G2 Gateway User Code.

gsi_pause

Causes the G2 Gateway bridge process to sleep for 1 second, or until a network event occurs on a network connection to the G2 Gateway bridge process.

Synopsis

```
void gsi_pause()
```

Description

By default, *gsi_pause()* causes the G2 Gateway bridge process to enter an interruptible sleep for 1 second. You can override this default one-second interval and set the maximum amount of time that the bridge process sleeps using the API function *gsi_set_pause_timeout()*.

Three events can awaken a bridge that has been paused:

- A timeout occurs on the specified pause interval.
- Network activity occurs on particular connections to external systems that you instruct the bridge process to monitor using the *gsi_watch_fd()* or *gsi_watch_fd_for_writing()* API functions.
- Network activity occurs over active connections to G2 processes. G2 Gateway automatically watches these connections; you do not have to instruct it to watch for them specifically.

In continuous mode, *gsi_run_loop()* calls *gsi_pause()* automatically at the end of each loop.

In one-cycle mode, the G2 Gateway bridge process enters an interruptible sleep only when your user code calls the *gsi_pause()* function. The bridge process does not enter an interruptible sleep automatically.

If you call *gsi_pause()* more than once, without calling *gsi_run_loop()* between the calls to *gsi_pause()*, only the first call to *gsi_pause()* causes the bridge to enter an interruptible sleep. Thus, if you call *gsi_pause()* ten times in succession, without intervening calls to *gsi_run_loop()*, the bridge sleeps for a maximum of only 1 second.

Note You can use the API function *gsi_wakeup()* in a multi-threaded application to cause a *gsi_pause()* in another thread to exit, allowing that thread to wake up.

Related Functions

Function	Description
gsi_set_pause_timeout()	Specifies the maximum amount of time that *gsi_pause()* can pause the bridge.
gsi_wakeup()	In a multi-threaded application, causes a *gsi_pause()* running in another thread to exit.
gsi_watch_fd()	Specifies a file descriptor that G2 Gateway watches for network read or error activity.
gsi_watch_fd_for_writing()	Specifies a file descriptor that G2 Gateway watches for network write activity.

gsi_print_backtrace

Prints a backtrace to the console on Sun4 and Solaris platforms.

Synopsis

```
void gsi_print_backtrace()
```

Description

`gsi_print_backtrace()` prints a backtrace to the console. If the G2 Gateway executable is stripped, G2 Gateway prints a numeric bracktrace; if it is not, G2 Gateway prints a symbolic backtrace. This funtion is useful for debugging.

gsi_reclaim_array

Deallocates an array of *gsi_item* structures.

Synopsis

*void gsi_reclaim_array(**array**)*

Argument	Description
*gsi_item **array*	The array of *gsi_item* deallocated by this function call.

Description

gsi_reclaim_array() deallocates an array of *gsi_item* structures.

You can allocate an array of *gsi_item* using the API function *gsi_make_array()*.

G2 Gateway automatically deallocates arrays that it allocates automatically. Do not attempt to deallocate automatically allocated arrays.

gsi_reclaim_attrs

Reclaims one or more instances of the *gsi_attr* structure.

Synopsis

void gsi_reclaim_attrs(attributes)

Argument	Description
*gsi_attr *attributes*	An array of *gsi_attr* structures that was previously allocated using the API function *gsi_make_attrs()*.

Description

gsi_reclaim_attrs() frees storage that was previously allocated by the API function *gsi_make_attrs()* for one or more instances of the *gsi_attr* structure.

gsi_reclaim_attrs_with_items

Reclaims one or more instances of the *gsi_attr* structure.

Synopsis

*void gsi_reclaim_attrs_with_items(**attributes**)*

Argument	Description
*gsi_attr **attributes*	An array of *gsi_attr* structures that was previously allocated using the API function *gsi_make_attrs_with_items()*.

Description

gsi_reclaim_attrs_with_items() frees storage that was previously allocated by the API function *gsi_make_attrs_with_items()* for one or more instances of the *gsi_attr* structure.

gsi_reclaim_item

Deallocates a specified *gsi_item* structure.

Synopsis

void gsi_reclaim_item(item)

Argument	Description
gsi_item item	The *gsi_item* deallocated by this function call.

Description

gsi_reclaim_item() deallocates a *gsi_item* structure. You are responsible for deallocating any items that you allocate using the API function *gsi_make_item()*.

G2 Gateway automatically deallocates any *gsi_item* structures that it allocates automatically. Do not attempt to deallocate automatically allocated *gsi_item* structures.

You should not attempt to reclaim the same items more than once.

gsi_reclaim_items

Reclaims an array of instances of *gsi_item* that were allocated using *gsi_make_items()*.

Synopsis

void gsi_reclaim_items(items)

Argument	Description
*gsi_item *items*	An array of instances of the *gsi_item* structure that was previously allocated using the API function *gsi_make_items()*.

Description

gsi_reclaim_items() frees storage that was previously allocated by the API function *gsi_make_items()* for one or more instances of the *gsi_item* structure.

You cannot partially reclaim arrays that were allocated using the API function *gsi_make_items()*.

You should not attempt to reclaim the same items more than once.

gsi_reclaim_registered_items

Reclaims an array of *gsi_registered_item* structures.

Synopsis

void gsi_reclaim_registered_items(registered_items)

Argument	Description
*gsi_registered_item *registered_items*	An array of *gsi_registered_item* structures that was allocated using the API function *gsi_make_registered_items()*.

Description

gsi_reclaim_registered_items() frees storage that was previously allocated using the API function *gsi_make_registered_items()*, for one or more instances of the *gsi_registered_item* structure.

gsi_registration_of_handle

Returns the *gsi_registration* for the given *item_handle* and *context*.

Synopsis

gsi_registration gsi_registration_of_handle
(item_handle, context_number)

Argument	Description
gsi_int item_handle	The item handle.
gsi_int context_number	The context.

Return Value	Description
gsi_registration	The *gsi_registration* structure that corresponds to *item_handle* and *context*, or a *NULL* pointer if *item_handle* is not a valid handle.

Description

gsi_registration_of_handle() returns a *gsi_registration* corresponding to a registered item. As input arguments, this function takes only the handle and context of the registered item.

Because *gsi_registration_of_handle*() returns a *NULL* pointer if the item handle specified by the *item_handle* argument is not valid, you can use this function to test the validity of handles.

gsi_registration_of_item

Returns the *gsi_registration* associated with a *gsi_registered_item*, *gsi_item*, or *gsi_registration*.

Synopsis

gsi_registration gsi_registration_of_item(item)

gsi_registration gsi_registration_of_item(regitem)

gsi_registration gsi_registration_of_item(registration)

Argument	Description
gsi_item item	The *gsi_item* for which this function returns the *gsi_registration*.
gsi_registered_item regitem	The *gsi_registered_item* for which this function returns the *gsi_registration*.
gsi_registration registration	If you specify a *gsi_registration* for this argument, the function returns that same *gsi_registration*.

Return Value	Description
gsi_registration	The registration returned by this function.

Description

gsi_registration_of_item() returns the registration of a registered item or *gsi_item*. If you specify a *gsi_registration* for *regitem_or_registration*, this function returns that *gsi_registration*.

gsi_reset_option

Turns off a G2 Gateway global run-time option.

Synopsis

```
void gsi_reset_option(option)
```

Argument	Description
`gsi_int` *option*	Symbolic constant that represents a G2 Gateway global run-time option.

Description

`gsi_reset_option()` turns off the G2 Gateway global run-time option specified by *option*.

G2 Gateway run-time options are global settings that control G2 Gateway operations and communications. For most purposes, it is best to set and reset these options in the callback function `gsi_set_up()`. However, you can set and reset the options from any place in your user code after `gsi_start()` has been called.

For *option*, specify a symbolic constant that represents a G2 Gateway runtime option, as listed in the following table.

G2 Gateway Runtime Options

Global Run-time Option	Purpose
`GSI_NO_SIGNAL_HANDLERS`	When set, directs G2 Gateway not to register its own signal handlers with the operating system. This can in some cases make debugging easier. When reset, directs G2 Gateway to register its own signal handlers. This is the default.
`GSI_ONE_CYCLE`	When set, allows control to be returned to your main function once per cycle. Refer to Processing Events through gsi_run_loop() for more information.
`GSI_PROTECT_INNER_CALLS`	When set, after encountering an error, G2 Gateway returns control to the caller rather than returning control to `gsi_run_loop()`.

G2 Gateway Runtime Options

Global Run-time Option	Purpose
GSI_STRING_CHECK	When set, filters out all non-ASCII characters sent to (but not *from*) G2.
GSI_SUPPRESS_OUTPUT	When set, prevents all output generated by G2 Gateway or the communications link from appearing as standard output to your screen.
GSI_TRACE_RUN_LOOP	When set, prints a message whenever *gsi_start()* or *gsi_run_loop()* are entered or exited.
GSI_TRACE_RUN_STATE	When set, prints a message whenever the flow of control enters or leaves G2 Gateway. If the *gsi_run_state_change()* callback is initialized, it prints the message before this callback is called.

gsi_return_attrs

Returns a value to a registered GSI variable and sets one or more of its attributes.

Synopsis

`void gsi_return_attrs(registered_item, attributes, count, context_number)`

Argument	Description
`gsi_registered_item`*registered_item*	The registered item that contains attribute values.
`gsi_attr` **attributes*	Array of attributes whose names, types, and values are set.
`gsi_int` *count*	Number of items represented in *attributes*.
`gsi_int` *context_number*	Context number specifying the GSI interface object through which *registered_item* was registered.

Description

`gsi_return_attrs()` returns a value (optionally null-typed, in which case no value is sent) to a GSI variable and returns values to one or more of the variable's attributes.

The values returned by this function (that is, both the value of the variable and the values of the variable's attributes) can be timestamped.

To update a variable's attributes without modifying its value, set the type of *registered_item* to `NULL_TAG`.

gsi_return_message

Returns a text string to the message-board item in the current KB of the connected G2 process.

Synopsis

void gsi_return_message(message,context_number)

Argument	Description
*gsi_char *message*	Text string passed to the connected G2 process. *gsi_return_message()* does not retain the *message* argument. If your G2 Gateway user code allocated the memory for the *message* string, it can deallocate this memory after *gsi_return_message()* completes, if it has no further use for the string.
gsi_int context_number	Context number that identifies a connection to a GSI interface.

Description

gsi_return_message() passes the text of a message to the message-board item in the current KB of a connected G2 process.

For example, the following *gsi_initialize_context()* callback function invokes *gsi_return_message()* to send the remote process initialization string of the GSI interface to the message-board:

```
gsi_int gsi_initialize_context(remote_process_init_string,
      length)
   gsi_char *remote_process_init_string;
   gsi_int length;
{
   char ret_msg[100];

   sprintf(ret_msg, "Initialization string %s",
            remote_process_init_string);
   gsi_return_message(ret_msg, gsi_current_context());
   return(GSI_ACCEPT);
}
```

gsi_return_timed_attrs

Returns a timestamped value and one or more optionally timestamped attribute values to a registered G2 variable.

Synopsis

void gsi_return_timed_attrs(registered_item, attributes, count, context_number)

Argument	Description
gsi_registered_item registered_item	The registered item that contains the timestamped value.
*gsi_attr *attributes*	Array of attributes whose names, types, and values have been set. Each attribute can be timestamped.
gsi_int count	Number of items represented in *attributes*.
gsi_int context_number	Context number specifying the GSI interface object through which *registered_item* was registered.

Description

gsi_return_timed_attrs() returns a timestamped value (optionally, null-typed, in which case, no value is sent) to a G2 variable and returns timestamped values to one or more of the variable's attributes.

The values returned by this function (that is, both the value of the variable and of the variable's attributes) can be timestamped. No special structure is required to send timestamped values.

To update a variable's attributes without modifying its value, set the type of *registered_item* to *NULL_TAG*.

gsi_return_timed_values

Returns one or more timestamped values to the last-recorded-value attribute of one or more GSI variables.

Synopsis

void gsi_return_timed_values(registered_items, count, context_number)

Argument	Description
*gsi_registered_item *registered_items*	An array of registered items that are set with new values for the last-recorded-value attribute. This value must be timestamped.
gsi_int count	Number of items represented in *registered_items*.
gsi_int context_number	Context number specifying the GSI interface object through which *registered_items* was registered.

Description

gsi_return_timed_values() passes new values to GSI variables in the connected G2 process.

Caution The internal G2 clock has a limit of +/- 17 years from the time that G2 is started. Using a timestamp that extends beyond this limitation may cause unexpected results. You are encouraged to do validity checking of timestamps if you suspect that this may be an issue.

gsi_return_values

Returns one or more values to the last-recorded-value attribute of one or more GSI variables, or to one or more G2 items.

Synopsis

void gsi_return_values(registered-items, count, context_number)

Argument	Description
*gsi_registered_item *registered-items*	An array of registered items. The values of these registered items are used to set the last-recorded-value attributes of GSI variables in G2. Each *gsi_registered_item* has a handle that refers to a *gsi_registration* or *gsi_item* that contains a value. You can make any desired changed to the attributes of the *gsi_item* structures. When *gsi_return_values()* is called, these changes are reflected in the original GSI variables for which G2 requested data service from the G2 Gateway bridge. For information about data service for GSI variables, see Implementing Data Service in G2 Gateway.
gsi_int count	Number of items represented in *registered_items*.
gsi_int context_number	Context number specifying the GSI interface object through which *registered_items* was registered.

Description

gsi_return_values() passes new values to GSI variables.

If your G2 application sets data points in the external system, the G2 Gateway bridge should call *gsi_return_values()* to echo these values back to G2.

gsi_rpc_call

Calls a G2 procedure that can return values to the G2 Gateway bridge.

Synopsis

void gsi_rpc_call(function_handle, arguments, call_identifier, context)

Argument	Description
gsi_function_handle_type function_handle	The G2 procedure. You must specify the same *function_handle* that is specified in the call to *gsi_rpc_declare_remote()* or *gsi_rpc_declare_remote_with_error_handler_and_user_data()* that declares the G2 procedure as a remote procedure.
*gsi_item *arguments*	The arguments passed to the G2 procedure through this remote procedure call.
gsi_call_identifier_type call_identifier	User data associated with this call to the G2 procedure and with any corresponding return call that G2 makes to the G2 Gateway receiver function. To use this argument, you must compile your G2 Gateway code with the GSI_USE_USER_DATA_FOR_CALLBACKS preprocessor macro defined or use the corresponding compile time switch. For more info see Call Identifiers for Remote Procedure Calls.
gsi_int context	The context through which this remote procedure call is made.

Description

Use *gsi_rpc_call()* to call a G2 procedure that can return values to the G2 Gateway bridge. The remote G2 procedure is executed as soon as G2 receives the remote procedure call from G2 Gateway.

Before your G2 Gateway user code calls a G2 procedure, it must declare the G2 procedure using the API function *gsi_rpc_declare_remote()* or *gsi_rpc_declare_remote_with_error_handler_and_user_data()*.

Your G2 Gateway user code must declare the *function_handle* argument as a *gsi_function_handle_type*, and assign a value to it by calling the API function *gsi_rpc_declare_remote()* or *gsi_rpc_declare_remote_with_error_handler_and_user_data*().

When the G2 procedure called by this function returns, the receiver function previously specified by *gsi_rpc_declare_remote()* or *gsi_rpc_declare_remote_with_error_handler_and_user_data*() is called with the return arguments.

Note If the G2 procedure does not return values to the G2 Gateway bridge, invoke the procedure using *gsi_rpc_start()* rather than *gsi_rpc_call()*.

---

gsi_rpc_call_with_count

Calls a G2 procedure that can return values to the G2 Gateway bridge, specifying the number of return arguments that G2 can pass back to G2 Gateway.

Synopsis

void gsi_rpc_call_with_count(function_handle, arguments, count, call_identifier, context)

Argument	Description
gsi_function_handle _type function_handle	The G2 procedure. You must specify the same *function_handle* that is specified in the call to *gsi_rpc_declare_remote()* that declares the G2 procedure as a remote procedure.
*gsi_item *arguments*	Arguments to be passed to the G2 procedure.
gsi_int count	Specifies the number of arguments passed to the G2 procedure in the remote procedure call. You need to include the *count* argument in this call only if the call to *gsi_rpc_ declare_remote()* specified -1 (negative one) for *argument_count*, which means an indeterminate number of arguments. The number of arguments that are in fact used in a call to the G2 procedure is specified in this *count* argument of *gsi_rpc_call_with_ count()*.

Argument	Description
gsi_call_identifier_type call_identifier	The user data associated with this call to the G2 procedure and with any corresponding return call that G2 makes to the G2 Gateway receiver function. To use this argument, you must compile your G2 Gateway code with the *GSI_USE_USER_DATA_FOR_CALLBACKS* preprocessor macro defined or use the corresponding compile time switch. For more info see Call Identifiers for Remote Procedure Calls.
gsi_int context	The number of the context in which this remote procedure call is made.

Description

gsi_rpc_call_with_count() enables you to call a G2 procedure that was declared as a remote procedure with an indeterminate number of arguments. To declare a remote procedure with an indeterminate number of arguments, you specify a value of -1 for *argument_count* in the call to *gsi_rpc_declare_remote()*.

If you declare a G2 procedure with an indeterminate number of arguments, you can call that procedure only with *gsi_rpc_call_with_count()*, and not with *gsi_rpc_call()*. You must specify the number of arguments that you pass to the G2 procedure in the *count* argument of *gsi_rpc_call_with_count()*.

You can pass user data to the G2 procedure through a call to *gsi_rpc_call_with_count()* if you compile your G2 Gateway application with the C preprocessor macro *GSI_USE_USER_DATA_FOR_CALLBACKS* defined.

gsi_rpc_declare_local

Declares a C function in your G2 Gateway application to be invocable, as a remote procedure, by a connected G2 process.

Synopsis

void gsi_rpc_declare_local(function, procedure_user_data, g2_function_name)

Argument	Description
*gsi_rpc_local_fn_type *function*	A pointer to the local function in your G2 Gateway user code. For information about the syntax and required arguments of local functions, see Writing a G2 Gateway Local Function to be Called by G2.
gsi_procedure_user_data_type procedure_user_data	To use this argument, you must compile your G2 Gateway code with the *GSI_USE_USER_DATA_FOR_CALLBACKS* preprocessor macro defined or use the corresponding compile time switch. For more info see Call Identifiers for Remote Procedure Calls
*gsi_char *g2_function_name*	The text string specified in the name-in-remote-system attribute of the remote procedure declaration, in G2, for the local function *function*.

Description

A G2 process can invoke any user-defined functions in your G2 Gateway user code that have been declared using *gsi_rpc_declare_local()*. Functions so declared as known as local functions. In your G2 Gateway user code, invoke *gsi_rpc_declare_local()* from within the callback function *gsi_set_up()*.

G2 can invoke local functions that return values to G2 using the call action, and invoke local functions that do not return values using the start action. To return values to G2, a local function can call the API functions *gsi_rpc_return_values()*, *gsi_return_timed_values()*, *gsi_return_attrs()*, or *gsi_return_timed_attrs()*. A local function that is invoked by either call or start can signal an error to G2 by calling *gsi_rpc_return_error_values()*.

gsi_rpc_declare_remote

Declares a G2 procedure in a G2 process to be invocable, as a remote process, by a connected G2 Gateway bridge process.

Synopsis

void gsi_rpc_declare_remote(function_handle, g2_function_name, receiver_function, procedure_user_data, argument_count, return_count, context_number)

Argument	Description
*gsi_function_handle_type *function_handle*	A pointer to the global variable used to identify the remote function. See Creating a Handle for the Remote Procedure for more information.
*gsi_char *g2_function_name*	Specifies the name of the remote function as it is known in G2. This remote function is a G2 procedure. This string must match the name of the G2 procedure as it appears in G2. Specify *g2_function_name* in uppercase letters, the usual format for G2 procedure names, which are of the type symbol. However, if the KB developer included lower case letters in the G2 procedure name through the use of escape sequences, specify the corresponding characters in *g2_function_name* in lower case. For information about how to specify symbols in G2, see the *G2 Reference Manual*.
*gsi_rpc_receiver_fn_type *receiver_function*	A pointer to the receiver function in the G2 Gateway bridge that will receive the values returned by the G2 procedure, or *NULL_PTR* if no values are returned. For information about the required argument syntax of receiver functions, see Defining a Function to Receive Values Returned by G2.

Argument	Description
gsi_procedure_user_data_type procedure_user_data	The user data associated with this call to the G2 procedure. G2 associates this procedure user data with any return call that it makes to *receiver_function*. It does not read or process the procedure user data itself. To use this argument, you must compile your G2 Gateway code with the *GSI_USE_USER_DATA_FOR_CALLBACKS* preprocessor macro defined or use the corresponding compile time switch. For more info see Call Identifiers for Remote Procedure Calls.
gsi_int argument_count	Specifies the number of arguments passed to the remote function (G2 procedure). A value of -1 (negative one) for *argument_count* means an indeterminate number of arguments. If you declare a G2 procedure with an indeterminate number of arguments, you can call it only with *gsi_rpc_call_with_count()*; in the call to this procedure, you specify the number of arguments that you are passing to the G2 procedure.
gsi_int return_count	Specifies the number of values returned by the remote function (G2 procedure) to the bridge. A value of -1 (negative one) means that the G2 procedure can return an indeterminate number of values.
gsi_int context_number	The context used by this function. The context identifies one particular G2.

Description

Calls to *gsi_rpc_declare_remote()* should be made from your G2 Gateway application's callback function *gsi_initialize_context()*, because remote procedure declarations are specific to a *context*. G2 Gateway must know the context for which a remote procedure call is declared in order to identify the G2 that contains the remote procedure.

The *function_handle* argument points to a function of the type *gsi_function_handle_type*. It is used in calls to the API functions *gsi_rpc_start()*, *gsi_rpc_call()*, and *gsi_rpc_call_with_count()*.

gsi_rpc_declare_remote_with_error_handler_and_user_data

Declares a G2 procedure in a G2 process to be invocable, as a remote process, by a connected G2 Gateway bridge process. Specifies the G2 Gateway error receiver function that receives error values returned by the G2 procedure, and associates user data with the call to the G2 procedure.

Synopsis

```
void gsi_rpc_declare_remote_with_error_handler_and_user_data
    (function_handle, g2_function_name, receiver_function,
     error_handler_function, procedure_user_data, argument_count,
     return_count, context_number)
```

Argument	Description
*gsi_function_handle_type *function_handle*	A pointer to the global variable used to identify the remote function. See Creating a Handle for the Remote Procedure for more information.
*gsi_char *g2_function_name*	Specifies the name of the remote function as it is known in G2. This remote function is a G2 procedure. This string must match the name of the G2 procedure as it appears in G2. Specify *g2_function_name* in uppercase letters, the usual format for G2 procedure names, which are of the type symbol. However, if the KB developer included lower case letters in the G2 procedure name through the use of escape sequences, specify the corresponding characters in *g2_function_name* in lower case. For information about how to specify symbols in G2, see the *G2 Reference Manual*.

Argument	Description
*gsi_rpc_receiver_fn_type *receiver_function*	A pointer to the receiver function in the G2 Gateway bridge that will receive the values returned by the G2 procedure, or *NULL_PTR* if no values are returned. For information about the required argument syntax of receiver functions, see Defining a Function to Receive Values Returned by G2.
*gsi_rpc_receiver_fn_type *error_handler_function*	The G2 Gateway receiver function that receives error return values from the G2 procedure if an error occurs during this remote procedure call. The format of the arguments received by *error_handler_function* is the same as the format of the error arguments for *gsi_rpc_return_error_values()*. For information about how to write an error handler function, see Defining a Function to Receive Error Values Returned by G2.
gsi_procedure_user_data_type procedure_user_data	The user data associated with this call to the G2 procedure. G2 associates this procedure user data with any return call that it makes to *receiver_function*. It does not read or process the procedure user data itself. To use this argument, you must compile your G2 Gateway code with the *GSI_USE_USER_DATA_FOR_CALLBACKS* preprocessor macro defined or use the corresponding compile time switch. For more info see Call Identifiers for Remote Procedure Calls.
gsi_int argument_count	Specifies the number of arguments passed to the remote function (G2 procedure). A value of -1 (negative one) for *argument_count* means an indeterminate number of arguments. If you declare a G2 procedure with an indeterminate number of arguments, you can call it only with *gsi_rpc_call_with_count()*; in the call to this procedure, you specify the number of arguments that you are passing to the G2 procedure.

Argument	Description
gsi_int return_count	Specifies the number of values returned by the remote function (G2 procedure) to the bridge. A value of -1 (negative one) means that the G2 procedure can return an indeterminate number of values.
gsi_int context_number	The context used by this function. The context identifies one particular G2.

Description

Calls to *gsi_rpc_declare_remote_with_error_handler_and_user_data()* should be made from your G2 Gateway application's callback function *gsi_initialize_context()*, because remote procedure declarations are specific to a *context*. G2 Gateway must know the context for which a remote procedure call is declared in order to identify the G2 that contains the remote procedure.

The *function_handle* argument is the address of a *gsi_function_handle_type* declared by the user. It is used in calls to the API functions *gsi_rpc_start()*, *gsi_rpc_call()*, and *gsi_rpc_call_with_count()*.

gsi_rpc_return_error_values

Signals an error to G2 from within a G2 Gateway local function.

Synopsis

```
void gsi_rpc_return_error_values(error_arguments, count,
   call_identifier, context)
```

Argument	Description
*gsi_item *error_arguments*	The error arguments returned to G2. Specify either: • An array of a single *gsi_item* representing an error object in G2. • An array of two gsi items, a *symbolic-expression* and a *text-expression*, similar to the arguments of the signal G2 procedure statement. See Description below for more information about how to specify the *error_arguments* argument.
gsi_int count	The count of arguments specified for *error_arguments* in this call to *gsi_rpc_return_error_values*(). Must be 1 or 2.
gsi_call_identifier_type call_identifier	Specify the *call_identifier* value that G2 used in its call to the local function that is invoking *gsi_rpc_return_error_values()*. G2 generates a unique call identifier for each call that it makes to a G2 Gateway local function.
gsi_int context	The context in which G2 made the remote procedure call and the error occurred.

Description

If an error occurs when G2 makes a remote procedure call to a G2 Gateway local function, you may want to return error values to G2, rather than data. You can use *gsi_rpc_return_error_values()* to do this.

`gsi_rpc_return_error_values()` signals an error to G2 from within G2 Gateway. How it does this depends on whether you specify one or two values for *error_arguments* in your call to this function.

- A single value for *error_arguments* must be a `gsi_item` representing an error object in G2. This item must:
  - Have the `null` type tag.
  - Should represent a valid error class in G2, such as default-error.
  - Have an attribute named error-description, containing a string representing the text of the error message.
  - Have an attribute named foundation-class, containing the symbol ERROR. The attribute enables G2 to create an error object for the returned error if the specified error class does not exist in G2.
- Two values for *error_arguments* must represent:
  - a `gsi_item` whose value is a *symbolic-expression*, which must name an error class or be the symbol rpc-error.
  - a `gsi_item` whose value is a *text-expression*, representing the error message.

The error argument or arguments are returned to the error handler of the G2 procedure that invoked the local function, to a procedure that invoked this procedure, or to the default error handler. For more information about G2 error handling, see the *G2 Reference Manual*.

gsi_rpc_return_values

Returns values to a G2 procedure, for a particular call over a particular context.

Synopsis

void gsi_rpc_return_values(arguments, count, call_identifier, context_number)

Argument	Description
*gsi_item *arguments*	An array of *gsi_item* that this procedure returns to G2.
gsi_int count	The number of elements in *arguments*.
gsi_call_identifier_type call_identifier	Specify the *call_identifier* value that G2 used in its call to the local function that is invoking *gsi_rpc_return_values()*. G2 generates a unique call identifier for each call that it makes to a G2 Gateway local function.
gsi_int context_number	Specify the integer ID of the context, or the macro *current_context*. This macro invokes the function *gsi_current_context()*.

Description

gsi_rpc_return_values() returns zero or more values to a G2 procedure that uses the G2 call action to invoke a C function as a remote procedure. Do not attempt to use this function in a local function that G2 invokes using the start action.

When G2 invokes a G2 Gateway local function that the G2 Gateway bridge has declared using the API function *gsi_rpc_declare_local()*, G2 generates a call identifier value and passes this value to the *call_identifier* argument of the G2 Gateway local function. When *gsi_rpc_return_values()* returns a value to G2, it must specify the *call_identifier* value received by the local function, to ensure that it returns the value to the outstanding G2 remote procedure call that invoked the local function, in the context in which the call was made.

Note An error occurs if you attempt to call *gsi_rpc_return_values()* more than once with the same *call_identifier* within the same *context*.

Multiple outstanding calls are supported, and can be returned in an order different from which they were made, *and* from outside the context in which they were called.

The arguments that this function returns must match those expected by the remote G2 procedure. If the remote G2 procedure expects zero return arguments, you must still call `gsi_rpc_return_values()` (with *count* of 0) to return from the call.

If a G2 Gateway local function is invoked by a start action in G2, the *call_identifier* argument of the local function is set to `GSI_CALL_HANDLE_OF_START`. In this case, the local function should not call `gsi_rpc_return_values()`, because G2 is not expecting the local function to return any values to it.

Caution G2 Gateway aborts when a wrong *call_identifier* is passed to `gsi_rpc_return_values`. To avoid this error, ensure that you pass the correct call index.

gsi_rpc_start

Starts a G2 procedure that does not return values to the G2 Gateway bridge.

Synopsis

void gsi_rpc_start(function_handle, arguments, context_number)

Argument	Description
gsi_function_handle_type function_handle	Specify the same *function_handle* value that is used in the call to *gsi_rpc_declare_remote()* that declares the G2 procedure as a remote procedure.
*gsi_item *arguments*	A pointer to the arguments passed to the G2 procedure.
gsi_int context_number	Specify the integer ID of the context, or *current_context*.

Description

gsi_rpc_start() starts a G2 procedure in a connected G2 process. The G2 procedure that is started cannot return values to the G2 Gateway bridge.

Before your G2 Gateway user code can invoke a G2 procedure, it must declare the G2 procedure using the API function *gsi_rpc_declare_remote()*.

Your G2 Gateway user code must declare the *function_handle* argument as a *gsi_function_handle_type*, and assign a value to it by calling the API function *gsi_rpc_declare_remote()*.

Note To call a remote G2 procedure that returns a value to the bridge, use *gsi_rpc_call()* rather than *gsi_rpc_start()*.

gsi_rpc_start_with_count

Starts a G2 procedure that does not return values to the G2 Gateway bridge.

Synopsis

void gsi_rpc_start_with_count(function_handle, arguments, count, context)

Argument	Description
gsi_function_handle_type function_handle	Specify the same *function_handle* value that is used in the call to *gsi_rpc_declare_remote()* that declares the G2 procedure as a remote procedure.
*gsi_item *arguments*	The arguments passed to the G2 procedure.
gsi_int count	Specifies the number of arguments passed to the G2 procedure in the remote procedure call. You need to include the *count* argument in this call only if the call to *gsi_rpc_declare_remote()* specified -1 (negative one) for *argument_count*, which means an indeterminate number of arguments. The number of arguments that *gsi_rpc_start_with_count()* passes to the G2 procedure by is specified in this *count* argument.
gsi_int context_number	Specify the integer ID of the context, or *current_context*.

Description

gsi_rpc_start_with_count() is identical to *gsi_rpc_start()*, except that it specifies the number of arguments that it is passing to the G2 procedure specified in *function_handle*.

gsi_run_loop

Provides the main event-handling loop of a G2 Gateway process.

Synopsis

```
void gsi_run_loop()
```

Description

Each time *gsi_run_loop()* is executed, it does the following:

1. It makes any new connections requested by G2.
2. In each currently active context on which there is network activity, it responds to all outstanding messages received from G2.

   gsi_run_loop() processes all messages that are outstanding at the time when it is called. Because G2 Gateway is single-threaded, *gsi_run_loop()* processes only those messages that are already completed at the time when *gsi_run_loop()* is called. It does not process messages that arrive or are completed during the current call to *gsi_run_loop()*; these messages are processed by the next call to *gsi_run_loop()*.

   For each message, the bridge executes a corresponding callback function or remote procedure call. The order in which *gsi_run_loop()* visits contexts cannot be predicted or controlled by the user.

The callback functions that *gsi_run_loop()* calls can in turn call other, nested statements and functions. The functions that *gsi_run_loop()* calls, both directly and through nested calls, are known as the gsi_run_loop() call tree.

gsi_run_loop() behaves differently in continuous and one-cycle modes of bridge operation:

- In continuous mode, *gsi_run_loop()* loops repeatedly after being called by *gsi_start()* as long as no fatal error occurs. If your bridge is running in continuous mode, your user code does not call *gsi_run_loop()* explicitly.

  At the end of each loop, *gsi_run_loop()* calls *gsi_pause()*, which causes the bridge process to enter an interruptible sleep. The bridge automatically wakes up when it detects network activity on any socket connected to any G2 process. If no network activity occurs within 1 second, the bridge wakes up automatically; if it finds no network activity to respond to, it goes back to sleep.

  Continuous mode is the better mode for polling an external system for data. The bridge can poll the external system using the callback function *gsi_g2_poll()*, which is invoked by G2 Gateway approximately once per second.

- In one-cycle mode, *gsi_start()* exits after the first completed execution of *gsi_run_loop()*. Control then passes from *gsi_start()* to *main()*. *gsi_run_loop()* does not loop automatically. To reenter *gsi_run_loop()*, your user code must call *gsi_run_loop()* explicitly.

  Running the bridge in one-cycle mode enables you to pass control from *gsi_run_loop()* to other functions within your G2 Gateway bridge process, as required by your application.

  One-cycle mode is the better mode for bridges designed to respond to network activity on connections to external systems, rather than to poll the external systems actively.

Caution An error results if you attempt to invoke *gsi_run_loop()* from within the *gsi_run_loop()* call tree. Do not attempt to make nested calls to *gsi_run_loop()*.

For more information about continuous and one-cycle modes, see Processing Events through gsi_run_loop().

gsi_set_attr_by_name

Invokes *gsi_set_item_of_attr_by_name*().

For information about this function, see *gsi_set_item_of_attr_by_name*.

gsi_set_attr_count

Sets the count of attributes of a G2 Gateway item that are available to your G2 Gateway user code.

Synopsis

void gsi_set_attr_count(item, count)

void gsi_set_attr_count(registered_item, count)

void gsi_set_attr_count(attribute, count)

void gsi_set_attr_count(registration, count)

Argument	Description
gsi_item item	A *gsi_item* structure whose count of accessible attributes is set by this function.
gsi_registered_item registered_item	A *gsi_registered_item* structure that points to the *gsi_item* structure whose count of accessible attributes is set.
gsi_attr attribute	A *gsi_attr* structure containing an embedded *gsi_item* structure whose count of accessible attributes is set.
gsi_registration registration	A *gsi_registration* pointing to the *gsi_item* whose count of accessible attributes is set by this function.
gsi_int count	The count of attributes.

Description

gsi_set_attr_count() specifies the number of attributes in a *gsi_item* structure that can be accessed by your G2 Gateway user code.

gsi_set_attr_count() is useful for specifying a subset of the attributes of a *gsi_item* structure that you want your G2 Gateway user code to be able to access or return to G2. For example, if you want only the first five attributes of a *gsi_item* to be available to your user code, you can use this function to set the attribute count to five.

To increase the number of attributes of a G2 Gateway data structure, use the API function *gsi_set_attrs()*. For information about this function, see *gsi_set_attrs*.

gsi_set_attr_is_transient

Sets a specified attribute to be transient or not transient.

Synopsis

void gsi_set_attr_is_transient(attribute, new_value)

Argument	Description
gsi_attr attribute	The attribute
gsi_int new_value	Specify 1 to make *attribute* transient, or 0 to make it not transient.

Description

G2 Gateway ignores transient attributes during remote procedure calls and does not send them to G2.

To determine whether an attribute is or is not transient, use the function *gsi_attr_is_transient()*.

gsi_set_attr_name

Changes the name of an attribute.

Synopsis

void gsi_set_attr_name(attribute, attribute_name)

Argument	Description
gsi_attr attribute	The attribute whose name is changed by this function.
gsi_symbol attribute_name	The new name of *attribute*. Specify *attribute_name* in uppercase letters only. If you are setting a class-qualified attribute name, use the following syntax: *CLASS_NAME::ATTRIBUTE_NAME* For example: *EQUIPMENT::APPLICATION* If you are setting an attribute name that is not class-qualified, use the following syntax: *ATTRIBUTE_NAME* For example: *APPLICATION*

Description

gsi_set_attr_name() changes the name stored in the *name* component of a *gsi_attr* structure.

This function changes both the class-qualified and the unqualified part of the attribute name.

gsi_set_attr_name() does not retain the *attribute_name* string. If your user code allocated memory for the *attribute_name* string, it can deallocate this memory after *gsi_set_attr_name()* completes, if it has no further use for the name.

Related Functions

Function	Description
gsi_attr_name_of()	Returns the name of an attribute.
gsi_unqualified_attr_name_of()	Returns the unqualified part of an attribute's name.
gsi_set_unqualified_attr_name()	Sets the unqualified part of an attribute's name.
gsi_attr_name_is_qualified()	Indicates whether an attribute name is qualified.
gsi_class_qualifier_of()	Returns the part of an attribute *name* that is the class qualifier.
gsi_set_class_qualifier()	Changes the part of an attribute name that specifies the G2 class that defines the attribute.

gsi_set_attrs

Initializes or replaces the set of *gsi_attr* instances associated with an item or an embedded item in an attribute.

Synopsis

void gsi_set_attrs(item, new_attributes, count)

void gsi_set_attrs(attribute, new_attributes, count)

Argument	Description
gsi_item item	An item whose attributes are set by this function.
gsi_attr attribute	An attribute containing an embedded item whose attributes are set by this function.
*gsi_attr *new_attributes*	An array of *gsi_attr* structures that this function uses to replace or initialize the attributes of the item or embedded item.
gsi_int count	The number of attributes pointed to by **new_attributes*.

Description

gsi_set_attrs() initializes or replaces the array of *gsi_attr* instances stored in a *gsi_item* or in a *gsi_item* embedded in a *gsi_attr*. In the *attribute(s)* component of the *gsi_item* structure, *gsi_set_attrs()* sets a pointer to the first element of the *new_attributes* array. *gsi_set_attrs()* does not copy the elements of the *new_attributes* array into the *gsi_item*.

You can allocate memory for the *new_attributes* array using the API functions *gsi_make_attrs()* or *gsi_make_attrs_with_items()*.

Do not deallocate the memory for the *gsi_attr* elements of *new_attributes* before or while you are sending the *gsi_item* to G2. Because the *gsi_item* contains only a pointer to the *new_attributes* array, the *gsi_item* loses its attributes if the array elements are deallocated before the *gsi_item* is received by G2.

The *new_attributes* argument must be a pointer to the *first* element of an array. An error results if the *new_attributes* argument points to any element of the array other than the first.

For example, given the following declaration:

```
gsi_attr *my_attrs = gsi_make_attrs_with_items(2);
```

the following call correctly sets a pointer to the first *gsi_attr* element of the *my_attrs* array:

```
gsi_set_attrs(my_item, my_attrs[0], attr_count);
```

However, the following call results in an error, because it attempts to pass the *second* element of the *my_attrs* array to *gsi_set_attrs()*:

```
gsi_set_attrs(my_item, my_attrs[1], attr_count);
```

If the array of attributes associated with the item or embedded item was allocated by your G2 Gateway user code — using the API functions *gsi_make_attrs()* or *gsi_make_attrs_with_items()* — this function does not deallocate the array of attributes. In this case, your user code is responsible for deallocating the attributes by calling the API functions *gsi_reclaim_attrs()* or *gsi_reclaim_attrs_with_items()*.

Related Functions

Function	Description
gsi_attr_count_of()	Determines how many attributes are in a *gsi_item* or *gsi_attr*.
gsi_attrs_of()	Obtains the attributes in a *gsi_item* or *gsi_attr*.
gsi_attr_by_name()	Obtains a specific attribute in a *gsi_item* or *gsi_attr*.
gsi_set_attr_by_name()	Changes a specific attribute in a *gsi_item* or *gsi_attr*.

gsi_set_class_name

Sets the value of the *class name* component of a *gsi_item* or an embedded *gsi_item* in an attribute.

Synopsis

void gsi_set_class_name(item, class_name)

void gsi_set_class_name(attribute, class_name)

Argument	Description
gsi_item item	An item whose G2 class is specified by this function.
gsi_attr attribute	An attribute containing an embedded item whose G2 class is specified by this function.
gsi_symbol class_name	Points to the name to which this function sets the G2 class name of *item* or *attribute*. Specify the class name in uppercase letters only. *gsi_set_class_name()* does not retain the *class_name* symbol. If your user code allocated memory for the *class_name* symbol, it can deallocate this memory after *gsi_set_class_name()* completes, if it has no further use for the name.

Description

Use *gsi_set_class_name()* to set the class name of any *gsi_item* that your G2 Gateway bridge will pass to G2 through a remote procedure call. When G2 receives the *gsi_item*, in creates an instance of the specified class based on the *gsi_item*.

Setting the class name is optional for a null or simple G2 type. If you specify a class name for a simple type, it must be to a variable or parameter type. Specifying a class name is also optional for list or array types. If you specify a class name for a list or array type, it must refer to the correct G2 list or array class, or to a subclass of one of these classes. You cannot set the class name to a sequence or structure type.

For information about the relationship between G2 data types and G2 Gateway type tags, see G2 Data Types and G2 Gateway Type Tags.

gsi_set_class_qualifier

Changes the part of the name of an attribute that specifies the G2 class that defines the attribute.

Synopsis

void gsi_set_class_qualifier(attribute, attribute_name)

Argument	Description
gsi_attr attribute	The name of the attribute.
gsi_symbol attribute_name	The new value to which the class-qualified part of the name is changed. Specify the class-qualified part of the attribute name in uppercase letters only.

Description

gsi_set_class_qualifier() changes the class-qualified part of the name of a *gsi_attr*.

In cases where an object inherits attributes with the same name from more than one superior class, G2 Gateway assigns a class-qualifier to the name of each of these attributes. The class qualifier specifies the name of the class from which the object inherits each attribute, and thus distinguishes the attributes from each other.

When *gsi_set_class_qualifier()* is executed, G2 Gateway copies the string specified by the *attribute_name* argument into its own memory. If your G2 Gateway user code allocated the memory for this string, it should deallocate this memory as soon as it has no further use for the string. The *gsi_set_class_qualifier()* function does not itself deallocate the *attribute_name* argument.

Related Functions

Function	Description
gsi_attr_name_of()	Returns the name of an attribute.
gsi_set_attr_name()	Changes the name of an attribute.
gsi_unqualified_attr_name_of()	Returns the unqualified part of an attribute's name.

Function	Description
gsi_set_unqualified_attr_name()	Sets the unqualified part of an attribute's name.
gsi_attr_name_is_qualified()	Indicates whether an attribute name is qualified.
gsi_class_qualifier_of()	Returns the part of an attribute *name* that is the class qualifier.
gsi_set_class_qualifier()	Changes the part of an attribute name that specifies the G2 class that defines the attribute.

gsi_set_class_type

Sets the type of the history data values associated with an item, a registered item, or an item that is embedded in an attribute.

Synopsis

```
void gsi_set_class_type (item, gsi_type)
void gsi_set_class_type (registered_item, gsi_type)
void gsi_set_class_type (attribute, gsi_type)
```

Argument	Description
`gsi_item` *item*	An item for which this function sets the type of the associated history data values.
`gsi_registered_item` *registered_item*	A registered item for which this function sets the type of the associated history data values.
`gsi_attr` *attribute*	An attribute containing an embedded item for which this function sets the type of the associated history data values.

`gsi_int` *gsi_type*	One of the following G2 Gateway types:
	`GSI_INTEGER_TAG`
	`GSI_SYMBOL_TAG`
	`GSI_STRING_TAG`
	`GSI_LOGICAL_TAG`
	`GSI_FLOAT64_TAG`
	`GSI_VALUE_TAG`
	`GSI_QUANTITY_TAG`

Description

`gsi_set_class_type()` sets the type of the history data values associated with an item, a registered item, or an item that is embedded in an attribute.

If a `gsi_item` that G2 Gateway sends to G2 through a remote procedure call corresponds to a variable-or-parameter, this field reflects the data type in G2, and indicates the allowable types for values for the variable-or-parameter.

The types of the history values are as follows:

Type	G2 Gateway Element Type	C Element Type
GSI_INTEGER_TAG	homogeneous integer values	*gsi_int*
GSI_SYMBOL_TAG	homogeneous symbol values	*char **
GSI_STRING_TAG	homogeneous string values	*char **
GSI_LOGICAL_TAG	homogeneous truth-values	*gsi_int*
GSI_FLOAT64_TAG	homogeneous floating-point numbers	*double*
GSI_VALUE_TAG	heterogeneous values	*gsi_item*
GSI_QUANTITY_TAG	heterogeneous numbers (*gsi_int* or *double*)	*gsi_item*

Related Functions

Function	Description
gsi_history_count_of()	Returns the number of history data values associated with an item.
gsi_extract_history()	Returns history data values associated with an item.
gsi_extract_history_spec()	Returns the history-keeping specification for an item.

gsi_set_context_limit

Overrides the G2 Gateway default limit on the number of active contexts.

Synopsis

void gsi_set_context_limit(limit)

Argument	Description
gsi_int limit	Any positive integer, or -1 (no limit)

Description

By default, G2 Gateway limits the number of active contexts to a maximum of 50, even if the platform on which G2 Gateway is running supports more than this.

You can call *gsi_set_context_limit()* to override this default.

Note This function cannot override any limits imposed by your operating system.

gsi_set_context_user_data

Associates user data with a connection that was initiated by a call to *gsi_initiate_connection_with_user_data()*.

Synopsis

void gsi_set_context_user_data(context, context_user_data)

Argument	Description
gsi_int context	The connection with which user data is associated.
gsi_context_user_data_type context_user_data	The user data associated with *context*.

Description

gsi_set_context_user_data() associates user data that you specify for *context_user_data* with the context specified by *context*.

The API function *gsi_context_user_data()* returns the user data associated with a specified context.

gsi_set_element_count

Sets the count of elements in an array or list contained in a *gsi_item* structure.

Synopsis

void gsi_set_element_count(item, count)

void gsi_set_element_count(attr, count)

Argument	Description
gsi_item item	A *gsi_item* containing the array or list.
gsi_attr attr	A *gsi_attr* whose embedded *gsi_item* contains the array or list.
gsi_int count	The number of elements in the *value(s)* component of *item* that is available to your G2 Gateway user code.

Description

gsi_set_element_count() specifies the number of elements in the *value(s)* component of a *gsi_item* or *gsi_attr* structure that can be accessed by your G2 Gateway user code.

gsi_set_element_count() is useful for specifying a subset of the elements in the *value(s)* component that you want your G2 Gateway user code to be able to access or return to G2. For example, if you want only the first five elements to be available to your user code, you can use this function to set the element count to five.

gsi_set_elements

Changes the contents of an item array, item list, value array, or value list to new contents.

Synopsis

`void gsi_set_elements(`*item, elements_array, count, type_tag*`)`

`void gsi_set_elements(`*attribute, elements_array, count, type_tag*`)`

Argument	Description
`gsi_item` *item*	An item to which this function writes a new array.
`gsi_attr` *attribute*	An attribute containing an embedded `gsi_item` to which this function writes a new array.
`gsi_item *`*elements_array*	The new contents of the array stored in the `value(s)` component of the item or embedded item. You can obtain an array to use as *elements_array* by allocating the array, using the API function `gsi_make_items()`. You can also obtain an item array by calling the API function `gsi_elements_of()`. This function does not make copies of the *elements_array* array or its elements.

Argument	Description
gsi_int count	The number of items in *elements_array*.
gsi_int type_tag	The type to which this function sets the *gsi_item* or *gsi_attr* specified as the first argument to this function. You can specify any of the following type tags: *GSI_VALUE_ARRAY_TAG* *GSI_VALUE_LIST_TAG* *GSI_ITEM_ARRAY_TAG* *GSI_ITEM_LIST_TAG* *GSI_QUANTITY_ARRAY_TAG* *GSI_QUANTITY_LIST_TAG* *GSI_ITEM_OR_VALUE_ARRAY_TAG* *GSI_ITEM_OR_VALUE_LIST_TAG* *GSI_SEQUENCE_TAG*

Description

gsi_set_elements() changes the contents of the *value(s)* component of a *gsi_item* to a new item array, item list, value array, or value list. It also sets the element count and type tag of the item array, item list, value array, or value list.

A value list or value array contains *gsi_item* structures that represent values. The *class name* component of a *gsi_item* structure that represent a value is null.

An item list or item array contains *gsi_item* structures that represent G2 items. The *class name* component of a *gsi_item* structure that represent an item contains the name of a G2 class.

Every element of a value list, value array, item list, or item array has a G2 Gateway type tag that specifies its data type. The elements of a *GSI_QUANTITY_ARRAY_TAG* or *GSI_QUANTITY_LIST_TAG* item must be integer or float.

The elements of a *GSI_VALUE_ARRAY_TAG* or *GSI_VALUE_LIST_TAG* item can be any value data type (float, integer, truth-value, symbol, text, sequence, or structure). For more information about G2 data types and the corresponding G2 Gateway data type tags, see G2 Data Types and G2 Gateway Type Tags.

If the item elements of *item* or *attribute* were automatically allocated by G2 Gateway (for example, as a result of the registration of an object passed through a remote procedure call or a GSI variable on which data service is performed) the *gsi_set_elements()* function deallocates the item elements as soon as it replaces them with the new ones in *elements_array*. However, *gsi_set_elements()* does not deallocate any existing elements of *item* or *attribute* if these elements were allocated by your G2 Gateway user code, using the API function *gsi_make_*

items(). In this case, your use code can call *gsi_reclaim_items()* to deallocate the items as soon as it has no further use for them.

Note G2 Gateway signals an error if the first argument to this function is neither a *gsi_item* nor a *gsi_attr*, or if the third argument is not of type *gsi_int* or is out-of-bounds relative to the given array.

To access the elements of a *gsi_item* or *gsi_attr*, use the API function *gsi_elements_of()*.

gsi_set_flt

Sets the value of an item, registered item, or embedded item in an attribute to a floating-point value

Synopsis

`void gsi_set_flt(`*item*, *float_value*`)`

`void gsi_set_flt(`*registered_item*, *float_value*`)`

`void gsi_set_flt(`*attribute*, *float_value*`)`

Argument	Description
`gsi_item` *item*	An item whose value is set by this function.
`gsi_registered_item` *registered_item*	A registered item whose value is set by this function.
`gsi_attr` *attribute*	An attribute containing an embedded item whose value is set by this function.
`double` *float_value*	The value to which this function sets *item*, *registered_item*, or *attribute*.

Description

`gsi_set_flt()` sets the value of the first argument, which can be an item, registered item, or embedded item in an attribute, to the specified C `double` floating-point value.

This function sets the G2 Gateway type of the first argument to `GSI_FLOAT64_TAG`.

gsi_set_flt_array

Initializes or replaces the array of floating-point values stored in an item with a new array.

Synopsis

void gsi_set_flt_array(item, doubles_array, count)

void gsi_set_flt_array(attribute, doubles_array, count)

Argument	Description
gsi_item item	An item to which this function writes the elements of a floating-point array.
gsi_attr attribute	An attribute containing an embedded item (*gsi_item*) to which this function writes the elements of a floating-point array.
*double *doubles_array*	An array whose contents are copied into the *value(s)* component of the item or embedded item. If *doubles_array* is located in a block of memory previously allocated with *malloc()*, you can deallocate this memory after *gsi_set_flt_array()* completes if your user code no longer needs it.
gsi_int count	The number of elements in *doubles_array*.

Description

gsi_set_flt_array() copies the contents of an array of floating-point values into the *value(s)* component of an item or embedded item. The original and copied arrays are represented by C arrays of type *double*.

If the first argument to this function is neither a *gsi_item* nor a *gsi_attr*, or if the *count* argument is out-of-bounds relative to the array in *item* or *attribute*, G2 Gateway signals an error.

The G2 Gateway type of the *gsi_item* or *gsi_attr* must be *GSI_FLOAT64_ARRAY_TAG*; otherwise, G2 Gateway signals an error. To determine whether a *gsi_item* or *gsi_attr* represents an array of floating-point values, verify that the value returned by the API function *gsi_type_of()* is *GSI_FLOAT64_ARRAY_TAG*.

To access the contents of a *gsi_item* or *gsi_attr* that stores an array of floating-point values, use the API function *gsi_flt_array_of()*.

Note If a *gsi_item* structure was allocated by the G2 Gateway user code through a call to the API function *gsi_make_items()*, the structure does not have a class name until you assign one to it through a call to *gsi_set_class_name()*.

You must assign a class name to the *gsi_item* structure if you intend to pass the *gsi_item* back to G2 as an object. In this case, the *gsi_item* structure must have a *class name* so that G2 can create a G2 object of that class to represent the *gsi_item*. The *class name* of the *gsi_item* must be the name of an existing G2 class.

If you are allocating a *gsi_item* that your bridge will send back to G2 as a value, you do not have to assign a class name to the *gsi_item*.

---

gsi_set_flt_list

Initializes or replaces the list of floating-point values stored in an item or registered item embedded item in an attribute with a new list

Synopsis

void gsi_set_flt_list(item, doubles_array, count)

void gsi_set_flt_list(attribute, doubles_array, count)

Argument	Description
gsi_item item	An item whose contents are initialized or replaced by this function.
gsi_attr attribute	An attribute containing an embedded item whose contents are initialized or replaced by this function.
*double *doubles_array*	An array of C type *double* that represents the list values in a G2 float-list item. If *doubles_array* is located in a block of memory previously allocated with *malloc()*, you can deallocate this memory after *gsi_set_flt_list()* completes if your user code no longer needs it.
gsi_int count	Number of elements in *doubles_array*.

Description

gsi_set_flt_list() replaces the list of floating-point values stored in the *value(s)* component of a *gsi_item* or *gsi_attr* with a new list of values. The *gsi_item* or *gsi_attr* uses a C array of type *double* to represent the old and new lists of floating-point values.

G2 Gateway signals an error if the first argument to this function is neither a *gsi_item* nor a *gsi_attr*, or if the third argument is out-of-bounds relative to the list in *item* or *attribute*.

The G2 Gateway type of the *gsi_item* or *gsi_attr* must be of G2 Gateway type *GSI_FLOAT64_LIST_TAG*; otherwise, G2 Gateway signals an error. To determine whether a *gsi_item* or *gsi_attr* represents a floating-point list, verify that the value returned by the API function *gsi_type_of()* is *GSI_FLOAT64_LIST_TAG*.

To access the contents of a *gsi_item* or *gsi_attr* that stores a floating-point list, use the API function *gsi_flt_list_of()*.

Note If a *gsi_item* structure was allocated by the G2 Gateway user code through a call to the API function *gsi_make_items()*, the structure does not have a class name until you assign one to it through a call to *gsi_set_class_name()*.

You must assign a class name to the *gsi_item* structure if you intend to pass the *gsi_item* back to G2 as an object. In this case, the *gsi_item* structure must have a *class name* so that G2 can create a G2 object of that class to represent the *gsi_item*. The *class name* of the *gsi_item* must be the name of an existing G2 class.

If you are allocating a *gsi_item* that your bridge will send back to G2 as a value, you do not have to assign a class name to the *gsi_item*.

---

gsi_set_handle

Sets the *handle* component in a *gsi_item* or *gsi_registered_item*

Synopsis

void gsi_set_handle(item, handle_value)

void gsi_set_handle(registered_item, handle_value)

Argument	Description
gsi_item item	An item whose handle component is set by this function.
gsi_registered_item registered_item	A registered item whose handle component is set by this function.
gsi_int handle_value	The value to which this function sets the handle component of *item* or *registered_item*.

Description

gsi_set_handle() sets the handle appropriately for a *gsi_item* or for a *gsi_registered_item*. Note that the behaviors described next for a *gsi_item* and *gsi_registered_item* never conflict, because G2 has no handle data type. Instead, G2 can pass a registered item's handle as an argument in a remote procedure call.

For *gsi_item*:

A *gsi_item* can have type tag *GSI_HANDLE_TAG*. This indicates that the *gsi_item* contains a reference to an object in G2. This function sets the type of the *gsi_item* to *GSI_HANDLE_TAG* and the value to *handle_value*.

Note A *gsi_item* of type *GSI_HANDLE_TAG* can be used only as an argument to a remote procedure call. It cannot be passed back as an attribute of a data-served GSI variable.

For *gsi_registered_item*:

A *gsi_registered_item* contains a handle, which refers to some object in G2 that has been registered for G2 Gateway data service through the API function *gsi_receive_registrations()* function.

Setting the handle of such a structure means changing the object in G2 to which the *gsi_registered_item* corresponds. Therefore, the only handle values that can sensibly be set into the handle slot of a *gsi_registered_item* are those that represent G2 items which have been registered for G2 Gateway data service. This is the same set of handle values obtained during calls to the G2 Gateway callback function *gsi_receive_registrations()*.

gsi_set_history

Sets the history data and history-keeping specification of an item or of an item embedded in an attribute.

Synopsis

`void gsi_set_history(`*item, values, timestamps, count, type, maximum_count, maximum_age, min_interval*`)`

`void gsi_set_history(`*attribute, values, timestamps, count, type, maximum_count, maximum_age, min_interval*`)`

Argument	Description
`gsi_item` *item*	An item whose history data and history-keeping specification are set by this function.
`gsi_attr` *attribute*	An attribute containing an item whose history data and history-keeping specification are set by this function.
`void` **values*	An array of values to which the history data of the item is set.
`double` **timestamps*	An array of timestamp values. Each timestamp is associated with one of the values in the *values* array.
`gsi_int` *count*	Number of elements in *values* and *timestamps*.
`gsi_int` *type*	Type of the elements in *values*.
`gsi_int` *maximum_count*	Maximum count desired in the item's history-keeping specification in G2.
`gsi_int` *maximum_age*	An interval in seconds, expressed in G2 as a positive integer. Used in the item's history-keeping specification in G2.
`gsi_int` *min_interval*	An interval in seconds, expressed in G2 as a float. Used in the item's history-keeping specification in G2.

Description

gsi_set_history() specifies the history data values and history-keeping specification for a *gsi_item* or *gsi_attr*. The maximum number of data points in the history is specified by *maximum_count*, and the maximum age of the data points, in seconds, is specified by *maximum_age*. The minimum interval between data points, in seconds, is specified by *min_interval*.

Note History data can be passed between a G2 Gateway bridge and a G2 only by means of remote procedure calls, and not through data service on GSI variables. For information about remote procedure calls, see Remote Procedure Calls.

---

gsi_set_include_file_version

Specifies the major and minor version of the G2 Gateway software release and its revision number so that G2 Gateway can make sure that the version of *gsi_main.h* corresponds with the version of the G2 Gateway software release.

Synopsis

void gsi_set_include_file_version(major,minor, rev)

Argument	Description
gsi_int major	The major version of the G2 Gateway software release.
gsi_int minor	The minor version of the G2 Gateway software release.
gsi_int rev	The revision number of the G2 Gateway software release.

Description

Previous versions of G2 Gateway defined the variables *gsi_include_file_major_version* and *gsi_include_file_minor_version*, which G2 Gateway used to verify that user code was compiled with the version of *gsi_main.h* that corresponds with the version of the G2 Gateway library being used.

You cannot use these variables with a G2 Gateway that is delivered as a DLL. Thus, if you are using G2 Gateway on WIN32 platforms (in addition to being delivered as three libraries, as before), use the *gsi_set_include_file_version()* function in place of the *gsi_include_file_major_version* and *gsi_include_file_minor_version* variables.

If your G2 Gateway is not delivered as a DLL, you can specify the major and minor versions and the revision of the G2 Gateway software release using the variables *gsi_include_file_major_version*, *gsi_include_file_minor_version*, and *gsi_include_file_revision_version* or the function *gsi_set_include_file_version()*.

Note The version of *gsi_main.c* provided with G2 Gateway includes a call to *gsi_set_include_file_version()*.

gsi_set_int

Sets the value of an item, registered item, or embedded item in an attribute to an integer value.

Synopsis

void gsi_set_int(item, integer_value)

void gsi_set_int(registered_item, integer_value)

void gsi_set_int(attribute, integer_value)

Argument	Description
gsi_item item	An item whose value is set by this function.
gsi_registered_item registered_item	A registered item whose value is set by this function.
gsi_attr attribute	An attribute containing an embedded item whose value is set by this function.
gsi_int integer_value	The integer value to which this function sets the value of *item*, *registered_item*, or the item embedded in *attribute*.

Description

gsi_set_int() sets the value of an item, registered item, or embedded item in an attribute to the specified *gsi_int* integer value.

This function also sets the G2 Gateway type of the *gsi_item*, *gsi_registered_item*, or item embedded in the *gsi_attr* to *GSI_INTEGER_TAG*.

gsi_set_int_array

Sets the array of integers stored in an item or embedded item to a new array of integers.

Synopsis

void gsi_set_int_array(item, integer_array, count)

void gsi_set_int_array(attribute, integer_array, count)

Argument	Description
gsi_item item	An item to which this function writes the contents of an integer array.
gsi_attr attribute	An attribute containing an embedded item (*gsi_item*) to which this function writes the contents of an integer array.
*gsi_int *integer_array*	An array of integers, whose contents are copied into the *value(s)* component of the item or embedded item. If *integer_array* is located in a block of memory previously allocated with *malloc()*, you can deallocate this memory after *gsi_set_int_array()* completes if your user code no longer needs it.
gsi_int count	The number of elements in *integer_array*.

Description

gsi_set_int_array() copies the contents of an integer array into the *value(s)* component of a *gsi_item* or embedded *gsi_item*. The original and copied arrays are represented by C arrays of type *gsi_int*. This function also sets the G2 Gateway type of the *gsi_item* or *gsi_item* embedded in the *gsi_attr* to *GSI_INTEGER_ARRAY_TAG*.

Note If a *gsi_item* structure was allocated by the G2 Gateway user code through a call to the API function *gsi_make_items()*, the structure does not have a class name until you assign one to it through a call to *gsi_set_class_name()*.

You must assign a class name to the *gsi_item* structure if you intend to pass the *gsi_item* back to G2 as an object. In this case, the *gsi_item* structure must have a *class name* so that G2 can create a G2 object of that class to represent the *gsi_item*. The *class name* of the *gsi_item* must be the name of an existing G2 class.

If you are allocating a *gsi_item* that your bridge will send back to G2 as a value, you do not have to assign a class name to the *gsi_item*.

gsi_set_int_list

Sets the value of an item or embedded item in an attribute to a value of G2 Gateway type *GSI_INTEGER_LIST_TAG*

Synopsis

void gsi_set_int_list(item, integer_array, count)

void gsi_set_int_list(attribute, integer_array, count)

Argument	Description
gsi_item item	An item that this function sets to a *GSI_INTEGER_LIST_TAG* value.
gsi_attr attribute	An attribute containing an embedded item that this function sets to a *GSI_INTEGER_LIST_TAG* value.
*gsi_int *integer_array*	A list of integers, to which the value of *item* or *attribute* is set.
gsi_int count	The number of elements in *integer_array*.

Description

Use *gsi_set_int_list()* to set the value of a *gsi_item* or *gsi_attr* so that it contains a C array of type *gsi_int*, as represented by *integer_array*.

This function also sets the G2 Gateway type of the *gsi_int* or *gsi_attr* to *GSI_INTEGER_LIST_TAG*.

Note If a *gsi_item* structure was allocated by the G2 Gateway user code through a call to the API function *gsi_make_items()*, the structure does not have a class name until you assign one to it through a call to *gsi_set_class_name()*.

You must assign a class name to the *gsi_item* structure if you intend to pass the *gsi_item* back to G2 as an object. In this case, the *gsi_item* structure must have a *class name* so that G2 can create a G2 object of that class to represent the *gsi_item*. The *class name* of the *gsi_item* must be the name of an existing G2 class.

If you are allocating a *gsi_item* that your bridge will send back to G2 as a value, you do not have to assign a class name to the *gsi_item*.

gsi_set_interval

Changes the default update interval associated with a registered item

Synopsis

void gsi_set_interval(registered_item, interval)

Argument	Description
gsi_registered_item registered_item	A registered item whose default update interval is changed by this function.
gsi_int interval	The number of seconds in the new default update interval. Must be 0 or a positive integer.

Description

gsi_set_interval() modifies the default update interval of a registered item. The *interval* argument represents the default update interval as a number of seconds.

To access the default update interval of a registered item, use the API function *gsi_interval_of()*.

Note Although *gsi_set_interval()* is supported, it is not useful because the field to which it writes is not read by G2 Gateway. For this reason, use of this function is not recommended.

gsi_set_item_append_flag

Causes contents of a *gsi_item* with a list or array type to be appended to values in an existing G2 array or list when returned to G2 through a call to *gsi_return_values()*.

Synopsis

void gsi_set_item_append_flag(item, flag)

Argument	Description
gsi_item item	The item that you intend to return to G2 through a call to *gsi_return_values()*.
gsi_int flag	Specify 1 to turn the flag on, causing the values in *gsi_item* to be appended to the array or list in G2. Specify 0 to turn the flag off, causing the values not to be appended.

gsi_set_item_of_attr

Replaces an existing embedded item in a *gsi_attr* with a different *gsi_item*.

Synopsis

void gsi_set_item_of_attr(attribute, source_item)

Argument	Description
gsi_attr attribute	An attribute into which this function sets a new item.
gsi_item source_item	The item that this function sets into *attribute*.

Description

gsi_set_item_of_attr() sets the *gsi_item* contained in a *gsi_attr* structure.

gsi_set_item_of_attr_by_name

Embeds a specified *gsi_item* in a specified attribute.

Synopsis

void gsi_set_item_of_attr_by_name(destination_item, search_name, source-item)

void gsi_set_item_of_attr_by_name(destination_attribute, search_name, source-item)

Argument	Description
gsi_item destination_item	An item that has the attribute specified by *search-name*.
gsi_attr destination_attribute	An attribute containing an embedded *gsi_item* that has the attribute specified by *search-name*.
gsi_symbol search-name	The name of the attribute whose embedded *gsi_item* is set by this function. Specify this name in uppercase letters, to correspond to the uppercase letters ordinarily used in G2 identifiers, which are of the G2 type symbol. *gsi_set_item_of_attr_by_name()* does not retain the *search-name* string. If your user code allocated memory for the *search-name* string, it can deallocate this memory after *gsi_set_item_of_attr_by_name()* completes, if it no longer needs the string.
gsi_item source-item	An item that this function embeds in the attribute specified by *search-name*.

Description

gsi_set_item_of_attr_by_name() sets the contents of the *gsi_item* embedded in a specified attribute to the item specified by the *source-item* argument. The name of the attribute is specified by *search-name*.

The new contents, as specified by the *source-item* argument, must be a *gsi_item*; otherwise, G2 Gateway signals an error. The destination must be a *gsi_item* or

gsi_attr; otherwise, G2 Gateway signals an error. If no matching attribute is found, G2 Gateway signals a warning.

The *gsi_set_item_of_attr_by_name()* function deallocates memory for any existing *gsi_item* embedded in the specified *gsi_attr* structure, if this *gsi_item* was allocated automatically by G2 Gateway itself.

However, the *gsi_set_item_of_attr_by_name()* function does not deallocate the old *gsi_item* structure if your user code itself allocated the *gsi_item* structure by calling *gsi_make_attr()* or *gsi_make_attr_with_item()*. In this case, your user code should deallocate the old *gsi_item* structure after *gsi_set_item_of_attr_by_name*() is executed, if it has no further use for the *gsi_item*.

Related Functions

Function	Description
gsi_attr_count_of()	Determines how many attributes are in a *gsi_item* or *gsi_attr*.
gsi_attrs_of()	Obtains the attributes in a *gsi_item* or *gsi_attr*.
gsi_set_attrs()	Changes the attributes in a *gsi_item* or *gsi_attr* to a new group of attributes.
gsi_attr_by_name()	Obtains a specific attribute in a *gsi_item* or *gsi_attr*.

gsi_set_log

Sets the value of an item, registered item, or embedded item in an attribute to a logical value (truth-value).

Synopsis

void gsi_set_log(item, truth_value)

void gsi_set_log(registered_item, truth_value)

void gsi_set_log(attribute, truth_value)

Argument	Description
gsi_item item	An item whose value is set by this function.
gsi_registered_item registered_item	A registered item whose value is set by this function.
gsi_attr attribute	An attribute containing an embedded item whose value is set by this function.
gsi_int truth_value	A truth-value, ranging from *GSI_FALSE* (*-1000*) for completely false to *GSI_TRUE* (*+1000*) for completely true.

Description

gsi_set_log() sets the value of a *gsi_item*, *gsi_registered_item*, or *gsi_attr* to a truth-value.

This function sets the G2 Gateway type of the first argument to

GSI_LOGICAL_TAG..

gsi_set_log_array

Sets the array of truth-values stored in an item or registered item embedded item in an attribute to a new array.

Synopsis

void gsi_set_log_array(item, truth_values_array, count)

void gsi_set_log_array(attribute, truth_values_array, count)

Argument	Description
gsi_item item	An item to which this function writes the elements of a truth-value array.
gsi_attr attribute	An attribute containing an embedded item (*gsi_item*) to which this function writes the elements of a truth-value array.
*gsi_int *truth_values_array*	An array of truth-values, each of whose values is a truth-value, ranging from *GSI_FALSE* (*-1000*) for completely false to *GSI_TRUE* (*+1000*) for completely true. The contents of this array are copied into the *value(s)* component of the item or embedded item. If *truth_values_array* is located in a block of memory previously allocated with *malloc()*, you can deallocate this memory after *gsi_set_log_array()* completes if your user code no longer needs it.
gsi_int count	Number of elements in *truth_values_array*.

Description

gsi_set_log_array() copies the contents of an array of truth-values into the *value(s)* component of an item or embedded item.

If the first argument to this function is neither a *gsi_item* nor a *gsi_attr*, or if the third argument is out-of-bounds relative to *truth_values_array*, G2 Gateway signals an error.

To determine whether a *gsi_item* or *gsi_attr* represents a truth-value array, verify that the value returned by the API function *gsi_type_of()* is *GSI_LOGICAL_ARRAY_TAG*.

To access the contents of a *gsi_item* or *gsi_attr* that stores a truth-value array, use the API function *gsi_log_array_of()*.

Note If a *gsi_item* structure was allocated by the G2 Gateway user code through a call to the API function *gsi_make_items()*, the structure does not have a class name until you assign one to it through a call to *gsi_set_class_name()*.

You must assign a class name to the *gsi_item* structure if you intend to pass the *gsi_item* back to G2 as an object. In this case, the *gsi_item* structure must have a *class name* so that G2 can create a G2 object of that class to represent the *gsi_item*. The *class name* of the *gsi_item* must be the name of an existing G2 class.

If you are allocating a *gsi_item* that your bridge will send back to G2 as a value, you do not have to assign a class name to the *gsi_item*.

---

gsi_set_log_list

Sets the list of truth-values stored in an item or registered item embedded item in an attribute to a new list.

Synopsis

void gsi_set_log_list(item, truth_values_array, count)

void gsi_set_log_list(attribute, truth_values_array, count)

Argument	Description
gsi_item item	An item whose list of truth-values is set by this function.
gsi_attr attribute	An attribute containing an embedded item whose list of truth-values is set by this function.
*gsi_int *truth_values_array*	An array of truth-values, each of which can range from *GSI_FALSE* (*-1000*) for completely false to *GSI_TRUE* (*+1000*) for completely true.
gsi_int count	The number of truth values in the *truth_values_array* array.

Description

gsi_set_log_list() changes the array of truth-values stored in a *gsi_item* or *gsi_attr* to a new list.

If the first argument to this function is neither a *gsi_item* nor a *gsi_attr*, or if the third argument is out-of-bounds relative to *truth_values_array*, G2 Gateway signals an error.

To determine whether a *gsi_item* or *gsi_attr* represents a truth-value list, verify that the value returned by the API function *gsi_type_of()* is *GSI_LOGICAL_LIST_TAG*.

To access the contents of a *gsi_item* or *gsi_attr* that stores a truth-value list, use the API function *gsi_log_list_of()*.

Note If a *gsi_item* structure was allocated by the G2 Gateway user code through a call to the API function *gsi_make_items()*, the structure does not have a class name until you assign one to it through a call to *gsi_set_class_name()*.

You must assign a class name to the *gsi_item* structure if you intend to pass the *gsi_item* back to G2 as an object. In this case, the *gsi_item* structure must have a *class name* so that G2 can create a G2 object of that class to represent the *gsi_item*. The *class name* of the *gsi_item* must be the name of an existing G2 class.

If you are allocating a *gsi_item* that your bridge will send back to G2 as a value, you do not have to assign a class name to the *gsi_item*.

gsi_set_long

Sets the value of an item, registered item, or embedded item in an attribute to an integer value.

Synopsis

void gsi_set_long(item, long_value)

void gsi_set_long(registered_item, long_value)

void gsi_set_long(attribute, long_value)

Argument	Description
gsi_item item	An item whose value is set by this function.
gsi_registered_item registered_item	A registered item whose value is set by this function.
gsi_attr attribute	An attribute containing an embedded item whose value is set by this function.
gsi_long long_value	The long value to which this function sets the value of *item*, *registered_item*, or the item embedded in *attribute*.

Description

gsi_set_long() sets the value of an item, registered item, or embedded item in an attribute to the specified *gsi_long* long value.

This function also sets the G2 Gateway type of the *gsi_item*, *gsi_registered_item*, or item embedded in the *gsi_attr* to *GSI_LONG_TAG*.

gsi_set_name

Sets the *name* component of a specified *gsi_item* structure.

Synopsis

void gsi_set_name(item, name)

Argument	Description
gsi_item item	An item whose name is set by this function.
gsi_symbol name	The text of the new name of *item*. Specify the text of *name* in uppercase letters only.

Description

gsi_set_name() sets the *name* component of a specified *gsi_item* structure.

gsi_set_name() does not deallocate the *name* string. If your user code allocated memory for the *name* string, it can deallocate this memory after *gsi_set_name()* completes, if it has no further use for the name.

gsi_set_option

Sets a G2 Gateway global run-time option.

Synopsis

void gsi_set_option(option)

Argument	Description
gsi_int option	Symbolic constant that represents a G2 Gateway global runtime option.

Description

gsi_set_option() sets the G2 Gateway global run-time option specified by *option*.

GSI's run-time options are global settings that control G2 Gateway's own operations and communications within your G2 Gateway application. You generally manipulate these options in your application's callback function *gsi_set_up()*, though your application can set them at any time and from any part of the application after it calls the callback function *gsi_start()*.

Pass *option* as the symbolic constant that represent a G2 Gateway global run-time option, as listed in the following table:

G2 Gateway Runtime Options

Global Run-time Option	Purpose
GSI_NO_SIGNAL_HANDLERS	When set, directs G2 Gateway not to register its own signal handlers with the operating system. This can in some cases make debugging easier. When reset, directs G2 Gateway to register its own signal handlers. This is the default.
GSI_ONE_CYCLE	When set, allows control to be returned to your main function once per cycle. Refer to Processing Events through gsi_run_loop() for more information.
GSI_PROTECT_INNER_CALLS	When set, after encountering an error, G2 Gateway returns control to the caller rather than returning control to *gsi_run_loop()*.

G2 Gateway Runtime Options

Global Run-time Option	Purpose
GSI_STRING_CHECK	When set, filters out all non-ASCII characters sent to (but not *from*) G2.
GSI_SUPPRESS_OUTPUT	When set, prevents all output generated by G2 Gateway or the communications link from appearing as standard output to your screen.
GSI_TRACE_RUN_LOOP	When set, prints a message whenever *gsi_start()* or *gsi_run_loop()* are entered or exited.
GSI_TRACE_RUN_STATE	When set, prints a message whenever the flow of control enters or leaves G2 Gateway. If the *gsi_run_state_change()* callback is initialized, it prints the message before this callback is called.

gsi_set_pause_timeout

Specifies the maximum amount of time that *gsi_pause()* can pause the bridge.

Synopsis

void gsi_set_pause_timeout(max_idle_time)

Argument	Description
gsi_int max_idle_time	The timeout period, in milliseconds.

Description

gsi_pause() executes while G2 Gateway has nothing to do or until the timeout is reached, whichever happens first. By default, the timeout period is 1000 ms. You can change this default using *gsi_set_pause_timeout()*.

gsi_set_rpc_remote_return_exclude_user_attrs

Declares user-defined attributes to exclude from the returned item of the specified function handle.

Synopsis

```
void gsi_set_rpc_remote_return_exclude_user_attrs
    (function_handle, attributes)
```

Argument	Description
`gsi_function_handle_type *function_handle`	A pointer to the global variable used to identify the remote function.
`gsi_item *attributes`	A newly allocated C array of `gsi_item` structures that determine the attributes to include or exclude.

Description

All user-defined attributes except the declared attributes are included in the returned item.

gsi_set_rpc_remote_return_include_system_attrs

Declares system-defined attributes to include in the returned item of the specified function handle.

Synopsis

```
void gsi_set_rpc_remote_return_include_system_attrs
    (function_handle, attributes)
```

Argument	Description
`gsi_function_handle_type *function_handle`	A pointer to the global variable used to identify the remote function.
`gsi_item *attributes`	A newly allocated C array of `gsi_item` structures that determine the attributes to include or exclude.

Description

Only the declared system-defined attributes are included in the returned item.

gsi_set_rpc_remote_return_include_all_system_attrs_except

Declares system-defined attributes to exclude in the returned item of the specified function handle.

Synopsis

```
void gsi_set_rpc_remote_return_include_all_system_attrs_except
    (function_handle, attributes)
```

Argument	Description
`gsi_function_handle_type *function_handle`	A pointer to the global variable used to identify the remote function.
`gsi_item *attributes`	A newly allocated C array of `gsi_item` structures that determine the attributes to include or exclude.

Description

All system-defined attributes except the declared attributes are included in the returned item.

For example, this code configures a remote procedure handle in G2 Gateway to include only the system-defined attributes uuid, item-width, and item-notes:

```
gsi_item * items;
gsi_char *uuidName = gsirtl_strdup("UUID");
gsi_char *itemWidthName = gsirtl_strdup("ITEM-WIDTH");
gsi_char *itemNotesName = gsirtl_strdup("ITEM-NOTES");
items = gsi_make_items(3);
gsi_set_sym(items[0],gsi_make_symbol(uuidName));
gsi_set_sym(items[1],gsi_make_symbol(itemWidthName));
gsi_set_sym(items[2],gsi_make_symbol(itemNotesName));
gsi_rpc_declare_remote(&getitem, "GETITEM", getitem_return_receiver,
    (procedure_user_data_type)NULL, 0, 1, current_context);
gsi_set_rpc_remote_return_include_system_attrs(getitem, items);
```

gsi_set_rpc_remote_return_value_kind

Specifies how G2 returns a particular argument that a G2 Gateway bridge passed to G2 in a remote procedure call. The choices are: *COPY*, *HANDLE*, or *BY-COPY-WITH-HANDLE*.

Synopsis

void gsi_set_rpc_remote_return_value_kind (function_handle, return_value_index, kind)

Argument	Description
gsi_function_handle_type function_handle	Specify the *function_handle* argument specified in the call to *gsi_rpc_declare_remote()* that you used to declare the G2 procedure as a remote procedure.
gsi_int return_value_index	An index into the list of arguments passed to G2 through the remote procedure call. The index value of the first argument is 0. If the *return_count* argument of *gsi_rpc_declare_remote()* was not -1, specify a value between 0 and 1 less than the *return_count* argument of *gsi_rpc_declare_remote()* to designate the particular argument to which the *kind* value specified in this call applies. If the *return_count* argument of *gsi_rpc_declare_remote()* was -1, you must specify -1 for *return_value_index*. This means that the *kind* value specified in this call applies to all arguments returned to the G2 Gateway bridge.
*gsi_char *kind*	Specify one of: *COPY*, *HANDLE* (corresponds to the G2 as handle item passing grammar), or *BY-COPY-WITH-HANDLE* (corresponds to the G2 with handle item passing grammar). By default, G2 passes the return value by *COPY*.

Description

gsi_set_rpc_remote_return_value_kind() enables G2 Gateway to specify how G2 returns values to G2 Gateway.

gsi_set_run_loop_timeout

Specifies a timeout period for *gsi_run_loop()*.

Synopsis

void gsi_set_run_loop_timeout(max_run_time)

Argument	Description
gsi_int max_run_time	The timeout period, in milliseconds.

Description

gsi_run_loop() executes until G2 Gateway has nothing to do, or until the timeout period is reached, whichever happens first. By default, the timeout period is 200 ms. You can call *gsi_set_run_loop_timeout()* to specify a different timeout period for *gsi_run_loop()*.

gsi_set_status

Sets the status code associated with a registered item.

Synopsis

void gsi_set_status(registered_item, status)

Argument	Description
gsi_registered_item registered_item	A registered item whose status code is set by this function.
gsi_int status	The new status code for *item*.

Description

gsi_set_status() modifies the status of a registered item.

To access the status code of a registered item, use the API function *gsi_status_of()*.

Caution An error results if your user code attempts return a *gsi_registered_item* to G2 with a null type (*GSI_NULL_TAG*) and a *status* value of *OK* (0). For information about the values to which you can set the status of *gsi_registered_item* structures, see Using the Gsi-Variable-Status Attribute.

gsi_set_str

Sets the value of an item, registered item, or embedded item in an attribute to a text string.

Synopsis

void gsi_set_str(item, text_value)

void gsi_set_str(registered_item, text_value)

void gsi_set_str(attribute, text_value)

Argument	Description
gsi_item item	An item whose value is set by this function.
gsi_registered_item registered_item	A registered item whose value is set by this function.
gsi_attr attribute	An attribute containing an embedded item whose value is set by this function.
*gsi_char *text_value*	The text of the new value of the item. *gsi_set_str()* does not deallocate the *text_value* string. If your user code allocated memory for the *text_value* string, it can deallocate this memory after *gsi_set_str()* completes, if it has no further use for the string.

Description

gsi_set_str() sets the *value(s)* component of a *gsi_item*, *gsi_registered_item*, or *gsi_attr* structure to a text string.

This function also sets the G2 Gateway type of the item represented by the first argument to *GSI_STRING_TAG*.

Caution Symbols and text strings cannot be used interchangeably. For example, using *gsi_set_sym* on a text attribute will fail and using *gsi_set_str* on a symbol attribute will fail.

gsi_set_str_array

Changes the array of text values stored in an item or embedded item in an attribute to a new array of text values.

Synopsis

void gsi_set_str_array(item, text_values_array, count)

void gsi_set_str_array(registered_item, text_values_array, count)

void gsi_set_str_array(attribute, text_values_array, count)

Argument	Description
gsi_item item	An item whose array of text values is changed by this function.
gsi_registered_item registered_item	A registered item that points to the *gsi_item* whose array of text values is changed by this function.
gsi_attr attribute	An attribute containing an embedded item (*gsi_item*) to which this function writes the contents of a string array.
*gsi_char **text_values_array*	An array of strings representing the text values. The contents of this array are copied into the *value(s)* component of the item or embedded item. If *text_values_array* is located in a block of memory previously allocated with *malloc()*, you can deallocate this memory after *gsi_set_str_array()* completes if your user code no longer needs it.
gsi_int count	Number of elements in the *text_values_array* array.

Description

gsi_set_str_array() copies the contents of a string array into the *value(s)* component of an item or embedded item. The original and copied string arrays are represented by C arrays whose elements are of type *gsi_char**.

If the first argument to this function is neither a *gsi_item* nor a *gsi_attr*, or if the third argument is out-of-bounds relative to *text_values_array*, G2 Gateway signals an error.

If the *gsi_item* or *gsi_attr* contains a user-allocated array when *gsi_set_str_array()* is called, this function does *not* automatically free the memory for this array. In contrast, if the *gsi_item* or *gsi_attr* contains memory allocated by G2 Gateway itself (for example, as the result of item-passing), this function automatically frees that memory.

To determine whether a *gsi_item* or *gsi_attr* represents an array of text values, verify that the value returned by the API function *gsi_type_of()* is *GSI_STRING_ARRAY_TAG*.

To access the contents of a *gsi_item* or *gsi_attr* that stores an array of text values, use the API function *gsi_str_array_of()*.

Caution Symbols and text strings cannot be used interchangeably. For example, using *gsi_set_sym* on a text attribute will fail and using *gsi_set_str* on a symbol attribute will fail.

Caution If a *gsi_item* structure was allocated by the G2 Gateway user code through a call to the API function *gsi_make_items()*, the structure does not have a class name until you assign one to it through a call to *gsi_set_class_name()*.

You must assign a class name to the *gsi_item* structure if you intend to pass the *gsi_item* back to G2 as an object. In this case, the *gsi_item* structure must have a *class name* so that G2 can create a G2 object of that class to represent the *gsi_item*. The *class name* of the *gsi_item* must be the name of an existing G2 class.

If you are allocating a *gsi_item* that your bridge will send back to G2 as a value, you do not have to assign a class name to the *gsi_item*.

---

gsi_set_str_list

Sets the list of text values stored in an item or registered item embedded item in an attribute to a new list of text values

Synopsis

`void gsi_set_str_list(`*item, text_values_array, count*`)`

`void gsi_set_str_list(`*registered_item, text_values_array, count*`)`

`void gsi_set_str_list(`*attribute, text_values_array, count*`)`

Argument	Description
`gsi_item` *item*	An item whose list of text values is changed by this function
`gsi_registered_item` *registered_item*	A registered item that points to the `gsi_item` whose list of text values is changed by this function
`gsi_attr` *attribute*	An attribute containing an embedded item whose list of text values is changed by this function.
`gsi_char **`*text_values_array*	An array of pointers to strings representing the new text values. If your user code allocates memory for the *text_values_array* string, it should deallocate this memory as soon as it has no further use for it. `gsi_set_str_list()` does not itself deallocate the *text_values_array* string.
`gsi_int` *count*	Number of values in *text_values_array*

Description

`gsi_set_str_list()` changes the list of text values stored in the `value(s)` component of a `gsi_item` or `gsi_attr` to a new list, represented by a C array whose elements are of type `gsi_char*`.

If the first argument to this function is neither a `gsi_item` nor a `gsi_attr`, or if the third argument is out-of-bounds relative to *text_values_array*, G2 Gateway signals an error.

If the *gsi_item* or *gsi_attr* already contains a user-allocated list when *gsi_set_str_list()* is called, this function does *not* automatically free the memory for that list. In contrast, if the *gsi_item* or *gsi_attr* contains memory allocated by G2 Gateway itself (for example, as the result of item-passing), this function automatically frees that memory.

To determine whether a *gsi_item* or *gsi_attr* represents a list of text values, verify that the value returned by the API function *gsi_type_of()* is *GSI_STRING_LIST_TAG*.

To access the contents of a *gsi_item* or *gsi_attr* that stores a list of text values, use the API function *gsi_str_list_of()*.

Caution Symbols and text strings cannot be used interchangeably. For example, using *gsi_set_sym* on a text attribute will fail and using *gsi_set_str* on a symbol attribute will fail.

Note If a *gsi_item* structure was allocated by the G2 Gateway user code through a call to the API function *gsi_make_items()*, the structure does not have a class name until you assign one to it through a call to *gsi_set_class_name()*.

You must assign a class name to the *gsi_item* structure if you intend to pass the *gsi_item* back to G2 as an object. In this case, the *gsi_item* structure must have a *class name* so that G2 can create a G2 object of that class to represent the *gsi_item*. The *class name* of the *gsi_item* must be the name of an existing G2 class.

If you are allocating a *gsi_item* that your bridge will send back to G2 as a value, you do not have to assign a class name to the *gsi_item*.

---

gsi_set_string_converson_style

Specifies the string conversion style used to convert strings passed between a G2 Gateway application and a G2.

Synopsis

void gsi_set_string_conversion_style(style)

Argument	Description
gsi_int style	The style for string conversions that is set by this function call. For information about the supported string conversion styles, see Description below.

Description

G2 Version 5.0 uses the Unicode character set internally for all strings. G2 Gateway can use a variety different character sets, known as **string conversion styles**, to represent strings for all G2 Gateway API functions and callbacks.

You can use *gsi_set_string_conversion_style*() to set the string conversion style if the compile time switch*GSI_WIDE_STRING_API* has not been set. If this switch has been set, G2 Gateway uses the Unicode character set, regardless of the style that you specify with *gsi_set_string_conversion_style()* or the setting of any other options. The *GSI_WIDE_STRING_API* option is set automatically if you compile your G2 Gateway code with the C preprocessor switch *GSI_USE_WIDE_STRING_API* defined.

The runtime option *GSI_STRING_CHECK* determines the string conversion style if neither the runtime option *GSI_WIDE_STRING_API* is set, nor *gsi_set_string_conversion_style*() has been called to specify a style. In this case, G2 Gateway uses the ISO_8859_1 character set, also known as LATIN_1. This is the same as making a call to *gsi_set_string_conversion_style()* with the character set *GSI_CHAR_SET_ISO_8859_1*.

The default string conversion style, if neither *GSI_WIDE_STRING_API* nor *GSI_STRING_CHECK* is set, and *gsi_set_string_conversion_style()* has not been called, is the GENSYM character set, also know as *UTF_G*. This is the character set that the previous releases of G2 Gateway (GSI) used.

The following list includes all the string conversion styles that you can specify for the *style* argument of *gsi_set_string_conversion_style()*. The styles listed in each bullet are synonyms for a same style, and can be used interchangeably. For more information about these styles, see *The Unicode Standard, Version 2.0* or other relevant documentation for international standards.

- GSI_CHAR_SET_GENSYM
  GSI_CHAR_SET_UTF_G

  The default string conversion style used if neither the *GSI_STRING_CHECK* nor the *GSI_WIDE_STRING_API* runtime option is set, and *gsi_set_string_conversion_style()* has not been called.

  This is the character set that the previous releases of G2 Gateway (GSI) used.

- GSI_CHAR_SET_ISO_8859_1 GSI_CHAR_SET_LATIN_1

  The default string conversion style used if *GSI_STRING_CHECK* is set and *GSI_WIDE_STRING_API* is not set, and *gsi_set_string_conversion_style*() has not been called to specify a style.

- GSI_CHAR_SET_ISO_8859_2,
  GSI_CHAR_SET_LATIN_2
- GSI_CHAR_SET_ISO_8859_3,
  GSI_CHAR_SET_LATIN_3
- GSI_CHAR_SET_ISO_8859_4,
  GSI_CHAR_SET_LATIN_4
- GSI_CHAR_SET_ISO_8859_5,
  GSI_CHAR_SET_LATIN_CYRILLIC
- GSI_CHAR_SET_ISO_8859_6,
  GSI_CHAR_SET_LATIN_ARABIC
- GSI_CHAR_SET_ISO_8859_7,
  GSI_CHAR_SET_LATIN_GREEK
- GSI_CHAR_SET_ISO_8859_8,
  GSI_CHAR_SET_LATIN_HEBREW
- GSI_CHAR_SET_ISO_8859_9,
  GSI_CHAR_SET_LATIN_5
- GSI_CHAR_SET_ISO_8859_1,
  GSI_CHAR_SET_LATIN_6
- GSI_CHAR_SET_US_ASCII,
  GSI_CHAR_SET_ASCII,
  GSI_CHAR_SET_FILTER_TEXT_FOR_GSI,
  GSI_CHAR_SET_ISO_646_IRV
- GSI_CHAR_SET_JIS,
  GSI_CHAR_SET_JIS_X_0208
- GSI_CHAR_SET_JIS_EUC,
  GSI_CHAR_SET_JIS_X_0208_EUC

- GSI_CHAR_SET_SHIFT_JIS,
  GSI_CHAR_SET_SHIFT_JIS_X_0208,
  GSI_CHAR_SET_MS_KANJI
- GSI_CHAR_SET_KS,
  GSI_CHAR_SET_KS_C_5601
- GSI_CHAR_SET_KS_EUC,
  GSI_CHAR_SET_KS_C_5601_EUC
- GSI_CHAR_SET_UNICODE_UTF_,
  GSI_CHAR_SET_UTF_7
- GSI_CHAR_SET_ISO_2022,
  GSI_CHAR_SET_X_COMPOUND_TEXT

gsi_set_sym

Uses a text string to set the value of an item, registered item, or embedded item in an attribute with a value of the G2 Gateway type *GSI_SYMBOL_TAG*.

Sets the symbol value associated with a *gsi_item*, *gsi_registered_item*, or *gsi_attr* structure.

Synopsis

void gsi_set_sym(item-regitem-attr, symbol-value)

Argument	Description
gsi_struct item-regitem-attr	The *gsi_item*, *gsi_registered_item*, or *gsi_attr* structure whose associated symbol value is set.
gsi_symbol symbol-value	The symbol value assigned to *item-regitem-attr*.

Description

gsi_set_sym() sets the *value(s)* component of a *gsi_item*, *gsi_registered_item*, or *gsi_attr* to a value of G2 Gateway type *GSI_SYMBOL_TAG*.

This function also sets the G2 Gateway type of the first argument to *GSI_SYMBOL_TAG*.

Caution Symbols and text strings cannot be used interchangeably. For example, using *gsi_set_sym* on a text attribute will fail and using *gsi_set_str* on a symbol attribute will fail.

gsi_set_sym_array

Sets the values in the symbol array associated with a *gsi_item* or *gsi_attr* structure.

Synopsis

void gsi_set_sym_array(item-attr, symbol-values-array, count)

Argument	Description
gsi_struct item-attr	The *gsi_item* or *gsi_attr* structure whose symbol array is assigned values.
*gsi_symbol *symbol-values-array*	The array of symbol values assigned to *item-attr*.
gsi_int count	The number of values in *symbol-values-array*.

Description

gsi_set_sym_array() copies the contents of a symbol array into the *value(s)* component of an item or embedded item. The original and copied symbol arrays are represented by arrays of type *gsi_symbol*.

If the first argument to this function is neither a *gsi_registered_item*, *gsi_item*, nor a *gsi_attr*, or if the third argument is out-of-bounds relative to *symbol_values_array*, G2 Gateway signals an error.

This function does not manage memory allocated by user-written functions in your GSI application. If the *gsi_item* or *gsi_attr* already contains a user-allocated array when *gsi_set_sym_array()* is called, this function does not automatically free the memory for that array. In contrast, if the *gsi_item* or *gsi_attr* contains memory allocated by G2 Gateway itself (for example, as the result of item-passing), this function automatically frees that memory.

To determine whether a *gsi_item* or *gsi_attr* represents an array of symbol values, verify that the value returned by the API function *gsi_type_of()* is *GSI_SYMBOL_ARRAY_TAG*.

To access the contents of a *gsi_item* or *gsi_attr* that stores an array of symbol values, use the API function *gsi_sym_array_of()*.

Caution Symbols and text strings cannot be used interchangeably. For example, using *gsi_set_sym* on a text attribute will fail and using *gsi_set_str* on a symbol attribute will fail.

Note If a *gsi_item* structure was allocated by the G2 Gateway user code through a call to the API function *gsi_make_items()*, the structure does not have a class name until you assign one to it through a call to *gsi_set_class_name()*.

You must assign a class name to the *gsi_item* structure if you intend to pass the *gsi_item* back to G2 as an object. In this case, the *gsi_item* structure must have a *class name* so that G2 can create a G2 object of that class to represent the *gsi_item*. The *class name* of the *gsi_item* must be the name of an existing G2 class.

If you are allocating a *gsi_item* that your bridge will send back to G2 as a value, you do not have to assign a class name to the *gsi_item*.

gsi_set_sym_list

Sets the values in the symbol list associated with a *gsi_item* or *gsi_attr* structure.

Synopsis

void gsi_set_sym_list(item, symbol-values-array,count)

void gsi_set_sym_list(attr, symbol-values-array,count)

Argument	Description
gsi_item item	The *gsi_item* whose symbol list is assigned values.
gsi_attr attr	The *gsi_attr* whose symbol list is assigned values.
*gsi_symbol *symbol-values-array*	The list of symbol values assigned to *item-attr*.
gsi_int count	The number of values in *symbol-values-array*.

Description

gsi_set_sym_list() changes the list of symbols stored in the *value(s)* component of a *gsi_item* or *gsi_attr* to a new list, represented by an array of type *gsi_symbol*.

If the first argument to this function is neither a *gsi_item* nor a *gsi_attr* or if the third argument is out-of-bounds relative to *symbol_values_array*, G2 Gateway signals an error.

If the *gsi_item* or *gsi_attr* already contains a user-allocated array when *gsi_set_sym_list()* is called, this function does *not* automatically free the memory for that list. In contrast, if the *gsi_item* or *gsi_attr* contains memory allocated by G2 Gateway itself (for example, as the result of item-passing), this function automatically frees that memory.

To determine whether a *gsi_item* or *gsi_attr* represents a list of symbol values, verify that the value returned by the API function *gsi_type_of()* is *GSI_SYMBOL_LIST_TAG*.

To access the contents of a *gsi_item* or *gsi_attr* that stores a list of symbol values, use the API function *gsi_sym_list_of()*.

Caution Symbols and text strings cannot be used interchangeably. For example, using *gsi_set_sym* on a text attribute will fail and using *gsi_set_str* on a symbol attribute will fail.

Note If a *gsi_item* structure was allocated by the G2 Gateway user code through a call to the API function *gsi_make_items()*, the structure does not have a class name until you assign one to it through a call to *gsi_set_class_name()*.

You must assign a class name to the *gsi_item* structure if you intend to pass the *gsi_item* back to G2 as an object. In this case, the *gsi_item* structure must have a *class name* so that G2 can create a G2 object of that class to represent the *gsi_item*. The *class name* of the *gsi_item* must be the name of an existing G2 class.

If you are allocating a *gsi_item* that your bridge will send back to G2 as a value, you do not have to assign a class name to the *gsi_item*.

gsi_set_symbol_user_data

Sets the user data associated with a symbol.

Synopsis

void gsi_set_symbol_user_data(symbol,symbol_user_data)

Argument	Description
gsi_symbol symbol	The symbol with which user data is to be associated.
gsi_symbol_user_data_type symbol_user_data	The user data associated with *symbol*. **Note**: The data type *gsi_symbol_user_data_type* is defined to have the type (*void **).

Description

gsi_set_symbol_user_data() enables you to associate an arbitrary value, specified in the *symbol_user_data* argument, with a specified symbol value.

gsi_set_timestamp

Sets the timestamp of an item or registered item embedded item in an attribute.

Synopsis

`void gsi_set_timestamp(`*registered_item, timestamp_value*`)`

`void gsi_set_timestamp(`*item, timestamp_value*`)`

`void gsi_set_timestamp(`*attribute, timestamp_value*`)`

Argument	Description
`gsi_registered_item` *registered_item*	A G2 Gateway registered item, which points to the `gsi_item` structure whose timestamp is set by this function.
`gsi_item` *item*	An item whose timestamp is set by this function.
`gsi_attr` *attribute*	An attribute containing an embedded `gsi_item` structure, whose timestamp is set by this function.
`double` *timestamp_value*	On UNIX and Windows, a timestamp value representing the number of seconds since midnight, January 1, 1970, GMT.

Description

Use `gsi_set_timestamp()` to set the timestamp of a `gsi_item` that is part of data service for a GSI variable. In the call to `gsi_set_timestamp()`, you can specify the `gsi_item` directly, specify the `gsi_registered_item` that points to the `gsi_item`, or specify a `gsi_attr` in which the `gsi_item` is embedded.

You cannot use `gsi_set_timestamp()` to set timestamps on data returned to G2 through remote procedure calls.

Caution The internal G2 clock has a limit of +/- 17 years from the time that G2 is started. Using a timestamp that extends beyond this limitation may cause unexpected results. You are encouraged to do validity checking of timestamps if you suspect that this may be an issue.

gsi_set_type

Sets the G2 Gateway type of an item, registered item, or embedded item in an attribute.

Synopsis

void gsi_set_type(item, gsi_type)

void gsi_set_type(registered_item, gsi_type)

void gsi_set_type(attribute, gsi_type)

Argument	Description
gsi_item item	An item whose G2 Gateway type is set by this function.
gsi_registered_item registered_item	A registered item whose G2 Gateway type is set by this function.
gsi_attr attribute	An attribute containing an embedded item whose G2 Gateway type is set by this function.
gsi_int gsi_type	Specify one of the following G2 Gateway types: *GSI_NULL_TAG* *GSI_INTEGER_TAG* *GSI_STRING_TAG* *GSI_LOGICAL_TAG* *GSI_FLOAT64_TAG* *GSI_SEQUENCE_TAG*, *GSI_STRUCTURE_TAG* all array and list tags If you specify an array or list type for an item originally in some other type, *gsi_set_type()* sets the length of the array or list to 0.

Description

Use *gsi_set_type()* to set the type of a *gsi_item*, *gsi_registered_item*, or *gsi_attr* structure. The type determines how the *gsi_item*, *gsi_registered_item*, or *gsi_attr* structure will be interpreted by G2 and by your G2 Gateway user code.

For most purposes, it is not necessary to call *gsi_set_type()*, because the API functions that set the *value* component of a *gsi_item*, *gsi_registered_item*, or *gsi_attr* structure also set the *type* component to the type of that value. For example, *gsi_set_flt()* sets the *value* component of a *gsi_item* structure to a floating point value and also sets the *type* component of that structure to *GSI_FLOAT64_TAG*.

However, *gsi_set_type()* is the only API function that can set the value of a G2 Gateway data structure to null.

Beginning in GSI 4.0, Rev.3, *gsi_set_type()* also sets default values for G2 Gateway structures. The following table lists the default values to which *gsi_set_type()* sets G2 Gateway structures.

Default Values Set by gsi_set_type()

Value Type in G2	G2 Gateway Type Tag Name	Default Value
no value condition	*GSI_NULL_TAG*	There is no default value for the *GSI_NULL_TAG* type.
integer	*GSI_INTEGER_TAG*	0
float	*GSI_FLOAT64_TAG*	0.0
text	*GSI_STRING_TAG*	""
truth-value	*GSI_LOGICAL_TAG*	0 Halfway between *GSI_TRUE* (1000) and *GSI_FALSE* (-1000)

If you use *gsi_set_type()* with types that are not listed in the table above, your user code should use the corresponding functions in the table below, to make them compatible with GSI 4.0 Rev. 3 and subsequent versions.

The following table lists the recommended API functions for setting the types of G2 Gateway structures.

API Functions for Setting Data Structure Types

Type Tag	Function
GSI_NULL_TAG	*gsi_set_type()*
GSI_INTEGER_TAG	*gsi_set_int()*
GSI_SYMBOL_TAG	*gsi_set_sym()*

API Functions for Setting Data Structure Types

Type Tag	Function
GSI_STRING_TAG	*gsi_set_str()*
GSI_LOGICAL_TAG	*gsi_set_log()*
GSI_FLOAT64_TAG	*gsi_set_flt()*
GSI_HANDLE_TAG	*gsi_set_handle()*
GSI_INTEGER_ARRAY_TAG	*gsi_set_int_array()*
GSI_SYMBOL_ARRAY_TAG	*gsi_set_sym_array()*
GSI_STRING_ARRAY_TAG	*gsi_set_str_array()*
GSI_LOGICAL_ARRAY_TAG	*gsi_set_log_array()*
GSI_FLOAT64_ARRAY_TAG	*gsi_set_flt_array()*
GSI_ITEM_ARRAY_TAG	*gsi_set_elements()*
GSI_VALUE_ARRAY_TAG	*gsi_set_elements()*
GSI_INTEGER_LIST_TAG	*gsi_set_int_list()*
GSI_SYMBOL_LIST_TAG	*gsi_set_sym_list()*
GSI_STRING_LIST_TAG	*gsi_set_str_list()*
GSI_LOGICAL_LIST_TAG	*gsi_set_log_list()*
GSI_FLOAT64_LIST_TAG	*gsi_set_flt_list()*
GSI_ITEM_LIST_TAG	*gsi_set_elements()*
GSI_VALUE_LIST_TAG	*gsi_set_elements()*

gsi_set_unqualified_attr_name

Sets the unqualified part of an attribute's name.

Synopsis

void gsi_set_unqualified_attr_name(attribute, attribute_name)

Argument	Description
gsi_attr attribute	An attribute, for which this function sets the unqualified part of the attribute name.
gsi_symbol attribute_name	The new unqualified part of the attribute name. Specify the unqualified part of the attribute name in uppercase letters only. *gsi_set_unqualified_att_name()* does not retain the *attribute_name* string. If your user code allocated memory for the *attribute_name* string, it can deallocate this memory after *gsi_set_unqualified_att_name()* completes, if it has no further use for the string.

Description

Use *gsi_set_unqualified_attr_name()* to change the part of the *name* component of a *gsi_attr* that is not class-qualified.

This function does not manage memory allocated by user-written functions in your GSI application. In particular, any memory *malloc*'d by a user-written function to form the new unqualified part of a *gsi_attr* instance's attribute name must also be *free*'d by a user-written function.

The functions for accessing and modifying the *name* component of a *gsi_attr* are:

gsi_attr_name_of()
gsi_set_attr_name()
gsi_unqualified_attr_name_of()
gsi_set_unqualified_attr_name()
gsi_attr_name_is_qualified()
gsi_class_qualifier_of()
gsi_set_class_qualifier()

gsi_set_update_items_in_lists_and_arrays_flag

If this flag is on then the contents of *item* is used to update attribute values of items in an existing G2 array or list when returned to G2 through a call to `gsi_return_values()`.

Synopsis

`void gsi_set_update_items_in_lists_and_arrays_flag(`*item*`, `*flag*`)`

Argument	Description
`gsi_item` *item*	The item with a list or array type that you intend to return to G2 through a call to `gsi_return_values()`, or a `gsi_item` contained in the item that you return.
`gsi_int` *flag*	Specify 1 to turn the flag on, causing the values in *item* to be used to update the array or list in G2. Specify 0 to turn the flag off, causing the values not to be appended.

Description

`gsi_set_update_items_in_lists_and_arrays_flag()` sets an internal G2 Gateway flag that causes items in a G2 list or array to be updated with the attribute values of the corresponding items returned to G2 through a call to `gsi_return_values()`. Non-items in the G2 list or array are replaced with the corresponding elements in the list or array returned to G2. Items have valid G2 class names in the `class name` component.

gsi_set_user_data

Sets the *user data* component of a *gsi_registration* or *gsi_item* structure.

Synopsis

void gsi_set_user_data(registration, user_data)

void gsi_set_user_data(item, user_data)

Argument	Description
gsi_registration registration	A registration whose *user data* component is set by this function.
gsi_item item	An item whose *user data* component is set by this function.
gsi_item_user_data_type user_data	The new value of the *user data* component of *registration*.

Description

Use *gsi_set_user_data()* to set the value of the *user data* component of a *gsi_registration* or *gsi_item* structure.

The *user data* component of these structures exists for the convenience of the G2 Gateway application programmer. It can store data specific to your G2 Gateway application, and specific to a particular data-served GSI variable in the current KB of a connected G2 process.

G2 Gateway itself does not read or reclaim the *user data* component of *gsi_registration* or *gsi_item* structures.

gsi_set_usv

Sets the value of an item, registered item, or embedded item in an attribute to an unsigned short vector.

Synopsis

void gsi_set_usv(item ,usv , length)

void gsi_set_usv(reg-item ,usv , length)

void gsi_set_usv(attr ,usv , length)

Argument	Description
gsi_item item	An item whose universal unique identifier is set by this function.
gsi_registered_item reg-item	A registered item whose universal unique identifier is set by this function.
gsi_attr attr	An attribute containing an embedded item whose universal unique identifier is set by this function.
*unsigned short *usv*	The contents of the unsigned short vector.
gsi_int length	The length of the unsigned short vector.

Description

The function gsi_set_usv() sets the *value* component of a *gsi_item*, *gsi_registered_item*, or *gsi_attr* structure to an unsigned short vector.

This function also sets the G2 Gateway type of the item represented by the first argument to *GSI_UNSIGNED_SHORT_VECTOR_TAG*.

G2 5.0 enables the user to create classes that inherit from the mixin class unique-identification. All instances of any subclass of unique-identification inherit an attribute named UUID. The value of this attribute is sent to G2 Gateway as an unsigned short vector.

The universal unique identifiers are similar to strings, except that they can contain embedded zeros, and cannot undergo character set translation. For more information about universal unique identifiers, see the *G2 Reference Manual*.

Unsigned short vectors are currently used only for universal unique identifiers (UUIDs).

gsi_signal_error

Invoke the G2 Gateway error handler and pass to it user-defined error information.

Synopsis

void gsi_signal_error(origin_of_error, user_error_code, message)

Argument	Description
*gsi_char *origin_of_error*	A descriptive key documenting the location of the error.
gsi_int user_error_code	An integer (greater than 1023) that represents a user-defined error code.
*gsi_char *message*	Text that describes the user-defined error code. *gsi_signal_error()* does not retain the *message* string. If your user code allocated memory for the *message* string, it can deallocate this memory after *gsi_signal_error()* completes, if it has no further use for the string.

Description

Include *gsi_signal_error()* in user-written functions to enable these functions to access the G2 Gateway error handler. The user-written function can call *gsi_signal_error()* when needed to invoke the G2 Gateway error handler and pass to it the error code and message specified by *user_error_code* and *message*.

You can use *gsi_signal_error()* in conjunction with a customized error handler. That is, the user-written customized error handler is called by the G2 Gateway error handler as a result of calling this function. For more information about installing a customized error handler, see the *gsi_install_error_handler*.

gsi_signal_handler

Enables you to invoke the G2 Gateway default error handler to handle particular signals.

Synopsis

void gsi_signal_handler(signal_code)

Argument	Description
gsi_int signal_code	The code of the signal, as listed in the documentation of the C library function *signal*.

Description

gsi_signal_handler() invokes the G2 Gateway signal handler, which then invokes the G2 Gateway error handler.

G2 Gateway by default registers its own signal handlers with the operating system, making it unnecessary for user code to handle signals. However, in some cases, you may require different handling for some signal codes from that provided by G2 Gateway.

To handle signals through your G2 Gateway user code, follow these steps:

1. Call *gsi_set_option(GSI_NO_SIGNAL_HANDLERS)* from within *gsi_set_up()*, to prevent G2 Gateway from registering its own signal handlers with the operating system.
2. Register your own signal handlers with the operating system, using the C library function *signal*.
3. Use *gsi_signal_handler()* to invoke the G2 Gateway error handler to handle particular signals. *gsi_signal_handler()* handles the specified signals even though you disabled the default G2 Gateway signal handling by setting the *GSI_NO_SIGNAL_HANDLERS* option.

   If you want to use the default G2 Gateway signal handling for a particular signal, you must either call *gsi_signal_handler()* from within your signal handler, or cause *gsi_signal_handler()* itself to be the signal handler.

gsi_simple_content_copy

Copies a source *gsi_item* structure to a specified destination *gsi_item* structure.

Synopsis

void gsi_simple_content_copy(destination_item, source_item)

Argument	Description
gsi_item destination_item	An item to which this function copies *source*.
gsi_item source _item	The item that this function copies into *destination*.

Description

The *destination* and *source* structures are distinct at the level of the given G2 Gateway item, but share substructures, such as attributes whose values are themselves *gsi_item* structures.

To duplicate complex *gsi_item* structures, you need to create recursive routines like the examples in *gsi_misc.c*. gsi_simple_content_copy does not duplicate attribute data structures.

gsi_start

Initializes G2 Gateway, sets up network listeners, and passes control to the API function *gsi_run_loop()*.

Synopsis

void gsi_start(argc, argv)

Argument	Description
int argc	Number of arguments passed from the command line to your G2 Gateway application's process.
*char **argv*	Array of strings containing the text of the arguments passed from the command line to your G2 Gateway application's process.

Description

gsi_start() takes as input the *argc* and *argv* arguments that were passed to the *main()* function in your G2 Gateway application. The *argc* and *argv* arguments can be modified in your G2 Gateway application's *main()* routine before being passed to *gsi_start()*.

gsi_start() does the following:

- Initializes G2 Gateway internals.
- Calls the callback function *gsi_set_up()*.
- Calls the callback function *gsi_get_tcp_port()*.
- Establishes TCP/IP network listener.
- Passes control to the API function *gsi_run_loop()*.

Whether *gsi_run_loop()* returns controls to *gsi_start()* depends on whether you operate your G2 Gateway application in Continuous mode or One-Cycle mode:

- In Continuous mode (the default), *gsi_start()* calls *gsi_run_loop()*, which loops continuously, rather than returning control to its caller. Your G2 Gateway bridge process executes entirely within the *gsi_run_loop()* call tree as long as no fatal error occurs.
- In One-Cycle mode, *gsi_run_loop()* executes only once and then returns control to *gsi_start()*, which then exits and returns control to *main()*.

Note One-Cycle mode does *not* automatically cause your G2 Gateway application's process to sleep while waiting for incoming network events from the connected G2 process. To cause your bridge process to sleep in one-cycle mode, your user code must call the API function `gsi_pause()`.

Data transmitted from your G2 Gateway application to the connected G2 process need not pass through `gsi_run_loop()`.

gsi_status_of

Returns the current status code associated with a registered item.

Synopsis

`gsi_int gsi_status_of(`*registered_item*`)`

Argument	Description
`gsi_registered_item` *registered_item*	A registered item whose status code is returned by this function.

Return Value	Description
`gsi_int`	Represents the current status code of *registered_item.*

Description

Use `gsi_status_of()` to obtain the status of a registered item.

To change the status code in a registered item, use the API function `gsi_set_status()`.

gsi_string_conversion_style

Returns the currently selected string conversion style.

Synopsis

```
gsi_int gsi_string_conversion_style()
```

Return Value	Description
gsi_int	The string conversion style currently selected for all strings in G2 Gateway. For a list of the supported string conversion styles, see *gsi_set_string_converson_style*.

Description

gsi_string_conversion_style() returns the currently selected string conversion style. For information about the string conversion styles and how to set them, see *gsi_set_string_converson_style*.

gsi_str_array_of

Returns the array of text values stored in an item or registered item embedded item in an attribute

Synopsis

*gsi_char **gsi_str_array_of(registered_item)*

*gsi_char **gsi_str_array_of(item)*

*gsi_char **gsi_str_array_of(attribute)*

Argument	Description
gsi_registered_item registered_item	A registered item pointing to an item (*gsi_item)* whose array of text values is returned by this function.
gsi_item item	An item whose array of text values is returned by this function.
gsi_attr attribute	An attribute containing an item whose array of text values is returned by this function.

Return Value	Description
*gsi_char **	A one-dimensional array of C strings that represents the value of a G2 text-array item. The array of C strings persists only as long as the data structure with which it is associated. If your user code needs to keep the array of strings for longer than the life-span of the data structure, it must copy the array into memory that it has allocated independently.

Description

Use *gsi_str_array_of()* to obtain a pointer to the C array of strings in a *gsi_item* or *gsi_attr*. This C array represents the value a G2 text-array item.

If the argument to this function is neither a *gsi_item* nor a *gsi_attr*, G2 Gateway signals an error.

This function does not allocate any new memory. It copies neither this array nor any of its elements, and its return value points to the array stored in the *gsi_item* or *gsi_attr*.

To determine whether a *gsi_item* or *gsi_attr* represents an array of text values, verify that the value returned by the API function *gsi_type_of()* is *GSI_STRING_ARRAY_TAG*.

To modify a *gsi_item* or *gsi_attr* so that it stores an array of text values, use the API function *gsi_set_str_array()*.

Caution Symbols and text strings cannot be used interchangeably. For example, using *gsi_set_sym* on a text attribute will fail and using *gsi_set_str* on a symbol attribute will fail.

---

gsi_str_list_of

Returns the list of text values stored in an item or registered item embedded item in an attribute.

Synopsis

*gsi_char **gsi_str_list_of(registered_item)*

*gsi_char **gsi_str_list_of(item)*

*gsi_char **gsi_str_list_of(attribute)*

Argument	Description
gsi_registered_item registered_item	A registered item pointing to an item (*gsi_item)* whose array of text values is returned by this function.
gsi_item item	An item whose list of text values is returned by this function.
gsi_attr attribute	An attribute containing an embedded item whose list of text values is returned by this function.

Return Value	Description
*gsi_char **	A one-dimensional array of C strings that represents the value of a G2 text-list item. The array of C strings persists only as long as the data structure with which it is associated. If your user code needs to keep the array of strings for longer than the life-span of the data structure, it must copy the array into memory that it has allocated itself.

Description

Use *gsi_str_list_of()* to obtain a pointer to the list of text values in a *gsi_item* or *gsi_attr*.

If the argument to this function is neither a *gsi_item* nor a *gsi_attr*, G2 Gateway signals an error.

This function does not allocate any new memory. It copies neither this list nor any of its elements, and its return value points to the list stored in the *gsi_item* or *gsi_attr*.

To determine whether a *gsi_item* or *gsi_attr* represents a list of text values, verify that the value returned by the API function *gsi_type_of()* is *GSI_STRING_LIST_TAG*.

To modify a *gsi_item* or *gsi_attr* so that it stores a list of text values, use the API function *gsi_set_str_list()*.

Caution Symbols and text strings cannot be used interchangeably. For example, using *gsi_set_sym* on a text attribute will fail and using *gsi_set_str* on a symbol attribute will fail.

---

gsi_str_of

Returns the value of an item, registered item, or embedded item in an attribute whose G2 Gateway type is *GSI_STRING_TAG*

Synopsis

*gsi_char *gsi_str_of(item)*

*gsi_char *gsi_str_of(registered_item)*

*gsi_char *gsi_str_of(attribute)*

Argument	Description
gsi_item item	An item whose value is returned by this function.
gsi_registered_item registered_item	A registered item whose value is returned by this function.
gsi_attr attribute	An attribute containing an embedded item whose value is returned by this function.

Return Value	Description
*gsi_char**	A C string representing the value of *item*, *registered_item*, or *attribute*. The C string persists only as long as the data structure with which it is associated. If your user code needs to keep the C string for longer than the life-span of the data structure, it must copy the array into memory that it has allocated itself.

Description

Use *gsi_str_of()* to retrieve the value of a *gsi_item*, *gsi_registered_item*, or *gsi_attr* whose G2 Gateway type is *GSI_STRING_TAG* as a C string.

The G2 Gateway type of the argument must be *GSI_STRING_TAG*, or G2 Gateway signals an error.

Caution Symbols and text strings cannot be used interchangeably. For example, using *gsi_set_sym* on a text attribute will fail and using *gsi_set_str* on a symbol attribute will fail.

gsi_sym_array_of

Returns the symbol array associated with a *gsi_item*, *gsi_registered_item*, or *gsi_attr* structure.

Synopsis

*gsi_symbol *gsi_sym_array_of(item-regitem-attr)*

Argument	Description
gsi_struct item-regitem-attr	The *gsi_item*, *gsi_registered_item*, or *gsi_attr* structure for which a symbol array is returned.

Return Value	Description
*gsi_symbol**	The symbol array associated with *item-regitem-attr*.

Description

gsi_sym_array_of() obtains a pointer to the array of symbol values in a *gsi_item*, *gsi_registered_item*, or *gsi_attr* structure.

If the argument to this function is not one of these structures, G2 Gateway signals an error.

This function does not allocate any new memory. Its return value points to the array stored in the specified structure, and this function copies neither this array nor any of its elements.

To determine whether a *gsi_item*, *gsi_registered_item*, or *gsi_attr* structure represents an array of symbol values, verify that the value returned by the API function *gsi_type_of()* is *GSI_SYMBOL_ARRAY_TAG*.

To modify a *gsi_item* or *gsi_attr* so that it stores an array of symbol values, use the API function *gsi_set_sym_array()*.

Caution Symbols and text strings cannot be used interchangeably. For example, using *gsi_set_sym* on a text attribute will fail and using *gsi_set_str* on a symbol attribute will fail.

gsi_sym_list_of

Returns the symbol list associated with a *gsi_item*, *gsi_registered_item*, or *gsi_attr* structure.

Synopsis

*gsi_symbol *gsi_sym_list_of(item-regitem-attr)*

Argument	Description
gsi_struct item-regitem-attr	The *gsi_item*, *gsi_registered_item*, or *gsi_attr* structure for which a symbol list is returned.

Return Value	Description
*gsi_symbol**	The symbol list associated with *item-regitem-attr*.

Description

Use *gsi_sym_list_of()* to obtain a pointer to the list of symbol values in a *gsi_item* or *gsi_attr*. If the argument to this function is neither a *gsi_item* nor a *gsi_attr*, G2 Gateway signals an error.

This function does not allocate any new memory. Its return value points to the list stored in the *gsi_item* or *gsi_attr*, and it copies neither this list nor any of its elements.

To determine whether a *gsi_item* or *gsi_attr* represents a list of symbol values, verify that the value returned by the API function *gsi_type_of()* is *GSI_SYMBOL_LIST_TAG*.

To modify a *gsi_item*, *gsi_attr*, or *gsi_registered_item* so that it stores a list of symbol values, use the API function *gsi_set_sym_list()*.

Caution Symbols and text strings cannot be used interchangeably. For example, using *gsi_set_sym* on a text attribute will fail and using *gsi_set_str* on a symbol attribute will fail.

gsi_sym_of

Returns the symbol associated with a *gsi_item*, *gsi_registered_item*, or *gsi_attr* structure.

Synopsis

gsi_symbol gsi_sym_of(item)

gsi_symbol gsi_sym_of(regitem)

gsi_symbol gsi_sym_of(attr)

Argument	Description
gsi_item item	The *gsi_item*, for which the symbol is returned.
gsi_registered_item regitem	The *gsi_registered_item*, for which the symbol is returned.
gsi_attr attr	The *gsi_attr* structure for which the symbol is returned.

Return Value	Description
gsi_symbol	The symbol associated with the *gsi_item*, *gsi_registered_item*, or *gsi_attr*.

Description

Use *gsi_sym_of()* to retrieve the value of a *gsi_item*, *gsi_registered_item*, or *gsi_attr*, whose G2 Gateway type is *GSI_SYMBOL_TAG*, as a C string.

The type of the structure specified in the *item-regitem-attr* argument must be *GSI_SYMBOL_TAG*, or G2 Gateway signals an error.

Caution Symbols and text strings cannot be used interchangeably. For example, using *gsi_set_sym* on a text attribute will fail and using *gsi_set_str* on a symbol attribute will fail.

gsi_symbol_name

Returns the string associated with a symbol. This string must not be modified.

Synopsis

*gsi_char *gsi_symbol_name(symbol)*

Argument	Description
gsi_symbol symbol	The symbol for which the associated string is returned.

Return Value	Description
*gsi_char**	The string associated with *symbol*.

gsi_symbol_user_data

Returns the user data associated with a symbol.

Synopsis

gsi_symbol_user_data_type gsi_symbol_user_data(symbol)

Argument	Description
gsi_symbol symbol	The symbol for which user data is returned.

Return Value	Description
gsi_symbol_user_data_type	The user data associated with *symbol*. **Note**: The data type *gsi_symbol_user_data_type* is defined to have the type (*void **).

gsi_timestamp_of

Returns the timestamp for an item, registered item, or embedded item in an attribute that was set by *gsi_set_timestamp()*.

Synopsis

double gsi_timestamp_of(item)

double gsi_timestamp_of(registered_item)

double gsi_timestamp_of(attribute)

Argument	Description
gsi_item item	An item whose timestamp is returned.
gsi_registered_item registered_item	A registered item whose timestamp is returned.
gsi_attr attribute	An attribute containing an embedded item whose timestamp is returned.

Return Value	Description
double	On UNIX and Windows, a timestamp value representing the number of seconds since midnight, January 1, 1970, GMT.

Description

Use *gsi_timestamp_of()* to obtain the timestamp associated with a *gsi_item* passed through a remote procedure call, or with a *gsi_registered_item* representing a data-served GSI variable. Use the API function *gsi_decode_timestamp()* to decode a timestamp into component parts (month, day, year, and so on).

Caution The internal G2 clock has a limit of +/- 17 years from the time that G2 is started. Using a timestamp that extends beyond this limitation may cause unexpected results. You are encouraged to do validity checking of timestamps if you suspect that this may be an issue.

gsi_type_of

Returns the G2 Gateway type of the value of an item, registered item, embedded item in an attribute, or registration.

Synopsis

gsi_int gsi_type_of(item)

gsi_int gsi_type_of(registered_item)

gsi_int gsi_type_of(attribute)

gsi_int gsi_type_of(registration)

Argument	Description
gsi_item item	An item whose G2 Gateway type is returned by this function.
gsi_registered_item registered_item	A registered item whose G2 Gateway type is returned by this function.
gsi_attr attribute	An attribute containing an item whose G2 Gateway type is returned by this function.
gsi_registration registration	A registration whose G2 Gateway type is returned by this function.

Return Value	Description
gsi_int	The G2 Gateway type of a *gsi_item*, *gsi_registered_item*, *gsi_attr*, or *gsi_registration*. The return value can be any G2 Gateway type tag except *GSI_QUANTITY_TAG* and *GSI_VALUE_TAG*.

Description

Use *gsi_type_of()* to obtain the G2 Gateway type of a *gsi_item*, *gsi_registered_item*, *gsi_registration*, or *gsi_attr*.

gsi_unqualified_attr_name_of

Returns the unqualified part of an attribute's name.

Synopsis

gsi_symbol gsi_unqualified_attr_name_of(attribute)

Argument	Description
gsi_attr attribute	An attribute from which this function returns the unqualified part of the attribute's name.

Return Value	Description
gsi_symbol	A read-only symbol that contains the unqualified part of the *name* component of *attribute*. The symbol persists only as long as the data structure with which it is associated. If your user code needs to keep the symbol for longer than the life-span of the data structure, it must copy the symbol into memory that it has allocated itself either through *malloc()* or a *make_xxx* G2 Gateway function.

Description

Use *gsi_unqualified_attr_name_of()* to access the unqualified part of the *name* component of a *gsi_attr*.

This function does not allocate any new memory. Its return value points to a string stored in *attribute*, and not to a copy of that string.

The functions for accessing and modifying the name of an attribute are:

gsi_attr_name_of()
gsi_set_attr_name()
gsi_unqualified_attr_name_of()
gsi_set_unqualified_attr_name()
gsi_attr_name_is_qualified()
gsi_class_qualifier_of()
gsi_set_class_qualifier()

gsi_unwatch_fd

Causes *gsi_run_loop()* not to wake up when input or output takes place on a file descriptor.

Synopsis

void gsi_unwatch_fd(file_descriptor)

Argument	Description
long file_descriptor	A UNIX file descriptor, usually referring to a network socket (or possibly a pipe), on which input or output can occur asynchronously.

Description

gsi_unwatch_fd() undoes the effect of both *gsi_watch_fd()* and *gsi_watch_fd_for_writing()*. It stops *gsi_run_loop()* from waking up whenever read, write, or error input or output takes place on the specified file descriptor. You must call *gsi_unwatch_fd()* before closing the specified file descriptor.

Use this function on file descriptors that have been previously passed as arguments to the API function *gsi_watch_fd()* or *gsi_watch_fd_for_writing()*. Calling this function on other file descriptors is not an error, but has no effect.

Note On Windows platforms, this facility only supports socket input/output as provided by the supported third-party socket facilities — for example, Multinet and the WinSock API.

For more information, consult the section on the API function *gsi_watch_fd()*.

Related Functions

Function	Description
gsi_watch_fd()	Specifies a file descriptor that G2 Gateway watches for network read or error activity.
gsi_watch_fd_for_writing()	Specifies a file descriptor that G2 Gateway watches for network write activity.
gsi_unwatch_fd_for_writing()	Stops G2 Gateway from watching for write activity on a file descriptor.

gsi_unwatch_fd_for_writing

Stops G2 Gateway from watching for write activity on a file descriptor.

Synopsis

void gsi_unwatch_fd_for_writing(file_descriptor)

Argument	Description
long file_descriptor	A UNIX file descriptor, usually referring to a network socket (or possibly a pipe), on which input or output can occur asynchronously.

Description

gsi_unwatch_fd_for_writing() undoes the effect of *gsi_watch_fd_for_writing()*. It stops *gsi_run_loop()* from waking up whenever writing input or output takes place on the specified file descriptor.

Use this function on file descriptors that have been previously passed as arguments to the API function *gsi_watch_fd_for_writing()*. Calling this function on other file descriptors is not an error, but has no effect. This function causes *gsi_run_loop()* to not wake up when write input or output takes place on a file descriptor.

Note On Windows platforms, this facility only supports socket input/output as provided by the supported third-party socket facilities — for example, Multinet and the WinSock API.

Related Functions

Function	Description
gsi_pause()	Causes the G2 Gateway bridge process to sleep for 1 second, or until a network event occurs on a network connection to the G2 Gateway bridge process.
gsi_watch_fd()	Specifies a file descriptor that G2 Gateway watches for network read or error activity.

Function	Description
gsi_unwatch_fd ()	Causes *gsi_run_loop()* to not wake up when input or output takes place on a file descriptor.
gsi_unwatch_fd_for_writing()	Specifies a file descriptor that G2 Gateway watches for network write activity.

gsi_update_items_in_lists_and_arrays_flag

Causes contents of a *gsi_item* with a list or array type to be used to update attribute values of items in an existing G2 array or list when returned to G2 through a call to *gsi_return_values()*.

Synopsis

gsi_int gsi_update_items_in_lists_and_arrays_flag(item)

Argument	Description
gsi_item item	The item with a list or array type that you intend to return to G2 through a call to *gsi_return_values()*, or a *gsi_item* contained in the item that you return.

Return Value	Description
gsi_int	A value of 1 turns the flag on, causing the values in *item* to be used to update the array or list in G2. A value of 0 to turns the flag off, causing the values not to be appended.

Description

The function gsi_update_items_in_lists_and_arrays_flag() returns the value of an internal G2 Gateway flag that causes items in a G2 list or array to be updated with the attribute values of the corresponding items returned to G2 through a call to *gsi_return_values()*. Non-items in the G2 list or array are replaced with the corresponding elements in the list or array returned to G2. Items have valid G2 class names in the *class name* component.

gsi_user_data_of

Obtains the contents of the *user data* component of a *gsi_registration* or *gsi_item*.

Synopsis

gsi_item_user_data_type gsi_user_data_of(registration)

gsi_item_user_data_type gsi_user_data_of(item)

Argument	Description
gsi_registration registration	The registration whose *user data* component is returned by this function.
gsi_item item	The item whose *user data* component is returned by this function.

Return Value	Description
gsi_item_user_data_type	The value of the *user data* component in *registration* or *item*.

Description

Use *gsi_user_data_of()* to obtain the contents of the *user data* component of a *gsi_registration* or *gsi_item*.

gsi_usv_length_of()

A macro that calls the API function *gsi_element_count_of()*. Returns the length of an unsigned short vector.

Synopsis

gsi_int gsi_element_count_of(item-regitem-or-attr)

Argument	Description
gsi_struct item-regitem-or-attr	A *gsi_item* for which this function returns the length of the unsigned short vector, or a *gsi_registered_item* pointing to a *gsi_item* for which the length is returned, or a *gsi_attr* containing an embedded *gsi_item* for which a length is returned.

Return Value	Description
gsi_int	The length of the unsigned short vector associated with the specified item, registered item, or embedded item in an attribute.

Description

Use *gsi_usv_length_of()* to determine the length of the unsigned short vector associated with an item, an item referenced by a registered item, or an embedded item in an attribute that is a list or array. The unsigned short vector represents the universal unique identifier of the item.

G2 5.0 enables the user to create classes that inherit from the mixin class unique-identification. All instances of any subclass of unique-identification inherit an attribute named uuid. The value of this attribute is sent to G2 Gateway as an unsigned short vector.

The universal unique identifiers are similar to strings, except that they can contain embedded zeros, and cannot undergo character set translation. For more information about universal unique identifiers, see the *G2 Reference Manual*.

Unsigned short vectors are currently used only for universal unique identifiers (UUIDs).

gsi_usv_of

Returns the universal unique identifier of an item, registered item, or embedded item in an attribute of G2 Gateway type *GSI_UNSIGNED_SHORT_VECTOR_TAG*.

Synopsis

*gsi_char *gsi_usv_of(item)*

*gsi_char *gsi_usv_of(registered_item)*

*gsi_char *gsi_usv_of(attribute)*

Argument	Description
gsi_item item	An item whose universal unique identifier is returned by this function.
gsi_registered_item registered_item	A registered item whose universal unique identifier is returned by this function.
gsi_attr attribute	An attribute containing an embedded item whose universal unique identifier is returned by this function.

Return Value	Description
*gsi_char **	A C unshort short array representing the universal unique identifier of *item*, *registered_item*, or *attribute*. The C array persists only as long as the data structure with which it is associated. If your user code needs to keep the C string for longer than the life-span of the data structure, it must copy the array into memory that it has allocated itself.

Description

Use *gsi_usv_of()* to retrieve the *value* component of a *gsi_item*, *gsi_registered_item*, or *gsi_attr* whose G2 Gateway type is *GSI_UNSIGNED_SHORT_VECTOR_TAG* as a C string. The G2 Gateway type of the argument must be *GSI_UNSIGNED_SHORT_VECTOR_TAG*, or G2 Gateway signals an error.

gsi_version_information

Returns the G2 Gateway version as a structure.

Synopsis

```
gsi_int gsi_version_information(*version_id)
```

Description

To get the G2 Gateway version, create a *gsi_version_id* structure and pass a pointer to that structure into *gsi_version_information*. G2 Gateway fills in the values of the structure. The return value is always 0.

The *gsi_version_id* type has the following members:

```
typedef struct {
gsi_int major_version;
  gsi_int minor_version;
  gsi_int revision_number;
  gsi_int release_quality;
  char *build_id;
} gsi_version_id;
```

The *release_quality* structure element can return the following constants:

```
GSI_PROTOTYPE_RELEASE_QUALITY
GSI_ALPHA_RELEASE_QUALITY
GSI_BETA_RELEASE_QUALITY
GSI_FCS_RELEASE_QUALITY
```

gsi_wakeup

In a multi-threaded application, causes a *gsi_pause()* running in another thread to exit.

Synopsis

gsi_int gsi_wakeup()

Return Value	Description
gsi_int	One of the following: • 1: Success. The *gsi_pause()* function exited. • 0: Initializing call. This has no effect on *gsi_pause()*. • -1: Failure. This call to *gsi_wakeup()* did not cause the *gsi_pause()* to exit.

Description

Use *gsi_wakeup()* in a multi-threaded application to cause a *gsi_pause()* in another thread to exit, allowing that thread to wake up.

The first invocation of *gsi_wakeup()* causes it to initialize itself, and has no effect on *gsi_pause()*. For this reason, your user code should make a first initializing call to *gsi_wakeup()* before the first call to *gsi_pause()*, and make a second call to *gsi_wakeup()* after the call to *gsi_pause()*.

gsi_wakeup() can be used only by console applications (non-event-based modes), running on UNIX or Windows platforms.

gsi_watch_fd

Specifies an open file descriptor that G2 Gateway watches for network activity. Activity on the specified file descriptor awakens G2 Gateway from an interruptible sleep.

Synopsis

void gsi_watch_fd(file_descriptor)

Argument	Description
long file_descriptor	An open UNIX file descriptor, usually referring to a network socket (or possibly a pipe), on which input or output can occur asynchronously. You can specify a file descriptor on any platform that supports Berkeley sockets and the TCP/IP protocol. **Note**: On Windows, you can specify only socket descriptors.

Description

gsi_watch_fd() tells G2 Gateway to watch for network activity on a connection to a system other than G2. The *file_descriptor* argument specifies the file descriptor for that connection. G2 Gateway will not watch for write activity unless *gsi_watch_fd_for_writing()* is subsequently called.

The G2 Gateway API function *gsi_pause()* will return if any file descriptor on which *gsi_watch_fd()* has been called has:

- Data available for reading.
- An error condition.

Note This function must be called before calling the corresponding function for write watching: *gsi_watch_fd_for_writing()*.

gsi_watch_fd() is more useful in one-cycle mode than in continuous mode, because one-cycle mode is the more efficient mode for responding to events on connections to external systems. However, you can use *gsi_watch_fd()* in continuous mode if you want your bridge process to wake up as soon as there is network activity on a connection to an external system, rather than waiting for activity on a connection to a G2 process.

In one-cycle mode, you can use the API function *gsi_pause()* to pause the G2 Gateway bridge process until it detects network activity on one of the file descriptors designated by *gsi_watch_fd()*. G2 Gateway immediately wakes up whenever input arrives on that file descriptor.

Note On Windows platforms, this facility only supports socket input/output as provided by the supported third-party socket facilities — for example, Multinet and the WinSock API.

Related Functions

Function	Description
gsi_pause()	Causes the G2 Gateway bridge process to sleep for 1 second, or until a network event occurs on a network connection to the G2 Gateway bridge process.
gsi_unwatch_fd()	Causes *gsi_run_loop()* to not wake up when input or output takes place on a file descriptor.
gsi_watch_fd_for_writing()	Specifies a file descriptor that G2 Gateway watches for network write activity.
gsi_unwatch_fd_for_writing()	Stops G2 Gateway from watching for write activity on a file descriptor.

gsi_watch_fd_for_writing

Specifies a file descriptor that G2 Gateway watches for network write activity.

Synopsis

void gsi_watch_fd_for_writing (file_descriptor)

Argument	Description
long file_descriptor	A UNIX file descriptor, usually referring to a network socket (or possibly a pipe), on which input or output can occur asynchronously. You can specify a file descriptor on any platform that supports Berkeley sockets and the TCP/IP protocol.

Description

gsi_watch_fd_for_writing() tells G2 Gateway to watch for network write activity on a connection to a system other than G2. Before calling *gsi_watch_fd_for_writing()*, you must first call *gsi_watch_fd()* with a given file descriptor, otherwise, this function has no effect. The *file_descriptor* argument specifies the file descriptor for that connection.

The G2 Gateway API function *gsi_pause()* will return if any file descriptor on which *gsi_watch_fd_for_writing()* has been called has:

- Data available for reading.
- An error condition.

Note On Windows platforms, this facility only supports socket input/output as provided by the supported third-party socket facilities — for example, Multinet and the WinSock API.

Related Functions

Function	Description
gsi_pause()	Causes the G2 Gateway bridge process to sleep for 1 second, or until a network event occurs on a network connection to the G2 Gateway bridge process.
gsi_watch_fd()	Specifies a file descriptor that G2 Gateway watches for network read or error activity.
gsi_unwatch_fd ()	Causes *gsi_run_loop()* to not wake up when input or output takes place on a file descriptor.
gsi_unwatch_fd_for_writing()	Stops G2 Gateway from watching for write activity on a file descriptor.

gsi_watchdog

Calls a specified user-written function when a time-out interval expires.

Synopsis

void gsi_watchdog(watchdog_function, timeout_interval)

Argument	Description
*gsi_watchdog_function_type *watchdog_function*	Pointer to a user-written function that performs customized processing. G2 Gateway calls *watchdog_function* when *timeout_interval* expires. The name of a function, specified without parentheses, will evaluate to a pointer to that function. Setting this argument to 0 can cause unpredictable results.
gsi_int timeout_interval	An integer greater than or equal to zero, that specifies the time-out interval in seconds. If *timeout_interval* is set to zero (0), the G2 Gateway watchdog timer is disabled.

Description

After *gsi_watchdog()* is called, a watchdog timer, internal to G2 Gateway, begins counting down to zero from a number of seconds, which must be greater than or equal to zero.

Note The watchdog function runs asynchronously from the rest of the program; therefore, *timeout_interval* values that are too small can cause unpredictable results.

If *gsi_watchdog()* is called again before the time-out period expires, G2 Gateway's own internal timer is set again to *timeout_interval*, which puts off the call to the user-written watchdog function.

At any time, by calling *gsi_watchdog()* and passing it a *timeout_interval* of zero (0), your G2 Gateway application can disable G2 Gateway's internal watchdog timer.

Note *gsi_watchdog()* is supported on all platforms in continuous mode; however, we do not recommend using it in one cycle mode where delays in user code could cause the timeout interval to expire.

10

Preprocessor Flags and Runtime Options

Describes C preprocessor macros and runtime options that you can use to modify the behavior of your G2 Gateway bridge.

Introduction

This chapter describes the G2 Gateway C preprocessor flags and runtime options. These flags and options determine particular aspects of the G2 Gateway bridge's runtime behavior.

G2 Gateway C Preprocessor Flags

G2 Gateway C preprocessor flags enable you to determine aspects of G2 Gateway behavior at compile time. The header file `gsi_main.h` contains the `#defines` for these macros.

Some of the C preprocessor flags have corresponding G2 Gateway options. The C macro `GSI_SET_OPTIONS_FROM_COMPILE()` sets these options based on settings of the C preprocessor flags. See the section, Defining C Preprocessor Flags for instructions on defining the C Preprocessor flags at compile time and calling `GSI_SET_OPTIONS_FROM_COMPILE()` from `gsi_main.c`.

The following table lists the G2 Gateway C preprocessor flags that affect compilation. See the sections that follow for a more detailed description of some of these macros.

G2 Gateway C Preprocessor Flags

C Preprocessor Flag	G2 Gateway Option	Description
GSI_USE_DLL	none	You must define this option when you compile and link a G2 Gateway delivered as a DLL. This flag also causes G2 Gateway to initialize the standard GSI 4.1 callback functions and sets the appropriate version control variables.
GSI_USE_NEW_SYMBOL_API	*GSI_NEW_SYMBOL_API*	Enables API functions to access symbols efficiently. Use of this option is recommended if your user code includes calls to functions that access symbols.
GSI_USE_NON_C_CALLBACKS	*GSI_NON_C*	Enables the use of callback functions written in a language other than C.
GSI_USE_USER_DATA_FOR_CALLBACKS	*GSI_USER_DATA_FOR_CALLBACKS*	Enables the use of call identifiers and procedure user data as arguments of remote procedure calls between G2 and G2 Gateway. For information about arguments of these types, see Call Identifiers and Procedure User Data.
GSI_USE_WIDE_STRING_API	*GSI_WIDE_STRING_API*	When set, G2 Gateway defines *gsi_char* as the C type *unsigned char* to support 16-bit Unicode (wide string) characters. When not set, *gsi_char* type is defined by default to be a *char* (8 bits).

G2 Gateway C Preprocessor Flags

C Preprocessor Flag	G2 Gateway Option	Description
__GENSYM_NOALIAS__	*none*	All the function names and types defined in *gsi_main.h* begin with the prefix *gsi_*. For convenience, Gensym has included in *gsi_main.h* versions of these names without the *gsi_* prefix defined by C preprocessor macros. If *__GENSYM_NOALIAS__* is defined before *gsi_main.h* is read, then these macros are not defined.
__GENSYMKR__	none	To disable ANSI C prototypes and use Kernighan and Ritchie style function declarations, define __GENSYMKR__ before you define the header file *gsi_main.h*.

GSI_USE_NEW_SYMBOL_API

Enables API functions to access symbols efficiently. Use of this C preprocessor flag is recommended if your user code includes calls to functions that access symbols.

For a list of the API functions that access symbols, see Symbols.

Caution Symbols and text strings cannot be used interchangeably. For example, using *gsi_set_sym* on a text attribute will fail and using *gsi_set_str* on a symbol attribute will fail.

GSI_NON_C

Enables you to program your application in languages that pass arguments by reference instead of by value, such as FORTRAN or Ada. Setting this option causes all callback functions to accept pointers to their expected argument types.

Do not set this option for applications in C++.

GSI_USE_WIDE_STRING_API

Enables the use of the wide string type to represent characters in the Unicode character set. G2 uses the Unicode character set for all strings.

When *GSI_USE_WIDE_STRING_API* is in effect, G2 Gateway uses the C type *gsi_char* to support 16-bit Unicode (wide string) characters and ignores any style that you specify using *gsi_set_string_conversion_style()*.

When the *GSI_USE_WIDE_STRING_API* option is *not* in effect, *gsi_char* type is defined by default to be a *char* (8 bits). With this setting, you can:

- Use the *GSI_STRING_CHECK* compile time switch to specify how strings are converted, or
- Call *gsi_set_string_conversion_style()* to specify a string conversion style.

  A string conversion style set through a call to *gsi_set_string_conversion_style()* takes precedence over the string conversion style set by the *GSI_STRING_CHECK* compiler option.

For information about wide strings, see Wide String Type.

Defining C Preprocessor Flags

The C preprocessor flags should be defined when you compile your G2 Gateway application. You can define the C preprocessor flag in the compiler command line or in *gsi_main.c*. For example, under UNIX, you can use the following compile time switch to define *GSI_USE_WIDE_STRING_API*:

```
-DGSI_USE_WIDE_STRING_API
```

You can also define *GSI_USE_WIDE_STRING_API* by including the following statement

```
#define GSI_USE_WIDE_STRING_API
```

before the `#include "gsi_main.h"` statement in any C file that includes *gsi_main.h*.

Some of the C preprocessor flags have corresponding G2 Gateway options, as documented in the previous sections. The C macro *GSI_SET_OPTIONS_FROM_COMPILE()* sets these options based on settings of the C preprocessor flags. You should therefore include a call to *GSI_SET_OPTIONS_FROM_COMPILE()* before the call to *gsi_start()* in your *gsi_main.c main()* function.

Note The *gsi_main.c* file included with your version of G2 Gateway already includes a call to *GSI_SET_OPTIONS_FROM_COMPILE()*.

G2 Gateway Runtime Options

G2 Gateway runtime options are global settings that control operations and communications within your G2 Gateway application.

The following table lists the G2 Gateway runtime options that affect G2. See the sections that follow for a more detailed description of each.

G2 Gateway Runtime Options

Runtime Option	Description
GSI_NO_SIGNAL_HANDLERS	Prevents G2 Gateway from registering its own signal handlers with the operating system. This can in some cases make debugging easier.
GSI_ONE_CYCLE	Allows control to be returned to your main function once per cycle. Refer to Processing Events through gsi_run_loop() for more information.
GSI_PROTECT_INNER_CALLS	After encountering an error, G2 Gateway returns control to the caller rather than returning control to *gsi_run_loop()*.
GSI_STRING_CHECK	Filters out all non-ASCII characters sent to (but not *from*) G2.
GSI_SUPPRESS_OUTPUT	Prevents all output generated by G2 Gateway or the communications link from appearing as standard output to your screen.
GSI_TRACE_RUN_LOOP	Prints a message whenever *gsi_start()* or *gsi_run_loop()* are entered or exited.
GSI_TRACE_RUN_STATE	Prints a message whenever the flow of control enters or leaves G2 Gateway. If the *gsi_run_state_change()* callback is initialized, it prints the message before this callback is called.

GSI_NO_SIGNAL_HANDLERS

Prevents G2 Gateway from registering its own signal handlers with the operating system. This can in some cases make debugging easier.

In addition, user code can register its own signal handlers with the operating system, using the C library function *signal*.

Note On systems running UNIX, G2 Gateway handles *SIGALRM* even when the G2 Gateway runtime option *GSI_NO_SIGNAL_HANDLERS* is set. This is necessary for the operation of G2 Gateway on Unix systems.

The following table lists how G2 Gateway handles signals when *GSI_NO_SIGNAL_HANDLERS* is not set.

Signal	Handled
SIGQUIT, *SIGILL*, *SIGTRAP*, *SIGIOT*, *SIGEMT*, *SIGFPE*, *SIGBUS*, *SIGSEVG*, *SIGSYS*	As aborts
SIGPIPE	Ignored
SIGLOST	Causes a core dump
SIGCHLD	Special handling (unspecified). This signal is irrelevant to G2 Gateway since it does not spawn child processes.

It is necessary to set this option in the *gsi_set_up()* callback, rather than in your *main()* function, because G2 Gateway normally registers its signal handlers after *gsi_set_up()* returns.

Description

gsi_set_option(GSI_NO_SIGNAL_HANDLERS) directs G2 Gateway not to register its own signal handlers.

gsi_reset_option(GSI_NO_SIGNAL_HANDLERS) directs G2 Gateway to register its own signal handlers. This is the default.

GSI_ONE_CYCLE

Allows control to be returned to your main function once per cycle. Refer to Processing Events through gsi_run_loop() for more information.

Description

gsi_set_option(GSI_ONE_CYCLE) directs the G2 Gateway bridge to run in one-cycle mode.

gsi_reset_option(GSI_ONE_CYCLE) directs the G2 Gateway bridge to run in continuous mode. This is the default.

GSI_PROTECT_INNER_CALLS

After encountering an error, G2 Gateway returns control to the caller rather than returning control to *gsi_run_loop()*.

For example, suppose *gsi_run_loop()* calls *gsi_receive_deregistrations()*, which then calls an API function. If G2 Gateway encounters an error in the API function and *GSI_PROTECT_INNER_CALLS* is not set, it returns control to gsi_run_loop(). This can cause undesirable results if *gsi_receive_deregistrations()* is unable to complete some of its tasks, such as freeing the memory reserved for registered variables. This same scenario may be applied to housekeeping activities usually performed by *gsi_shutdown_context()*.

You should be aware that this runtime option can make G2 Gateway API functions slower, but it prevents a potential cause of G2 Gateway aborts and protocol-out-of-synchronization problems. However, using the *GSI_PROTECT_INNER_CALLS* runtime option can prevent hangs in applications that use the *gsi_run_state_change* callback.

Description

gsi_set_option(GSI_PROTECT_INNER_CALLS) returns control to the caller rather than returning control to *gsi_run_loop()* after encountering an error.

gsi_reset_option(GSI_PROTECT_INNER_CALLS) returns control to *gsi_run_loop()* after encountering an error. This is the default.

GSI_STRING_CHECK

Filters out all non-ASCII characters sent to G2.

In addition, it converts escaped characters sent from G2, as shown in the following table:

Name of Character	Encoding Used in G2 Gateway	Encoding Used in G2
Newline	'\n`	@L
Backslash	\	~\
Tilde	~	~~
At sign	@	~@

Description

gsi_set_option(GSI_STRING_CHECK) causes strings to be converted as shown in the table above.

gsi_reset_option(GSI_STRING_CHECK) causes strings not to be converted. This is the default.

GSI_SUPPRESS_OUTPUT

Directs G2 Gateway not to send information and error messages to standard output. Set this option if you do not want information and error messages displayed on the screen.

Description

gsi_set_option(GSI_SUPPRESS_OUTPUT) directs G2 Gateway not to send information and error messages to standard output.

gsi_reset_option(GSI_SUPPRESS_OUTPUT) directs G2 Gateway to send information and error messages to standard output. This is the default.

GSI_TRACE_RUN_LOOP

Prints a message whenever *gsi_start()* or *gsi_run_loop()* are entered or exited.

Description

gsi_set_option(GSI_TRACE_RUN_LOOP) causes messages to print whenever *gsi_start()* or *gsi_run_loop()* are entered or exited.

gsi_reset_option(GSI_TRACE_RUN_LOOP) disables this option. This is the default.

GSI_TRACE_RUN_STATE

Prints a message whenever the flow of control enters or leaves G2 Gateway. If the *gsi_run_state_change()* callback is initialized, it prints the message before this callback is called.

Description

*gsi_set_option(GSI_TRACE_RUN_STATE)*causes messages to be printed whenever the flow of control enters or leaves G2 Gateway.

gsi_reset_option(GSI_TRACE_RUN_STATE) disables this option.

Setting and Resetting Runtime Options

The API functions *gsi_set_option()* and *gsi_reset_option()* set and reset G2 Gateway options, using the name of an option as their single argument. *gsi_set_option()* turns on a G2 Gateway runtime option (sets the option), and *gsi_*

reset_option() turns it off (resets the option). G2 Gateway runtime options default to the reset state.

You set or reset a G2 Gateway runtime option in the bridge's user code. You generally manipulate these options in your application's callback function *gsi_set_up()*, though your application can set them at any time and from any part of the application after it calls the callback function *gsi_start()*.

For example, the following code illustrates how to set and reset the *GSI_ONE_CYCLE* option:

```
/* Run bridge in one-cycle mode */
gsi_set_option(GSI_ONE_CYCLE)

/* Run bridge in continuous mode. */
gsi_reset_option(GSI_ONE_CYCLE)
```

11

Building and Running a G2 Gateway Bridge

Describes how to compile, link, and run a G2 Gateway bridge executable image, and how to start and stop a G2 Gateway bridge process from within a G2 procedure.

Introduction

This chapter describes:

- How to compile, link and run your G2 Gateway bridge on UNIX® and Windows™ platforms.
- The configuration requirements, if any, that you must satisfy before you begin to run your G2 Gateway bridge.

Note This chapter does not describe how to install the hardware devices, network boards, and software that you need to support your G2 Gateway bridge.

G2 Gateway Files

If you have installed G2 Gateway on your computer, you should have the following files in your G2 Gateway directory:

- *gsi_main.h*, a header file that you must include in your user code.
- *gsi_main.c* and *gsi_main.o* (*gsi_main.obj* on Windows) for your main routine, can be used as part of your user code.
- *gsimmain.c* and *gsimmain.obj* (on Windows only). Required for building Windows applications; not required when building console applications.
- *gsi_misc.h*, a header file that you must include if you use the functions contained in *gsi_misc.c*.
- *gsi_misc.c* and *gsi_misc.o* (*gsi_misc.obj* on Windows). Contain functions for duplicating, displaying, and freeing memory associated with GSI items; for viewing G2 Gateway structures; for copying, printing, and deleting recursive structures; and for using *gsi_history_type_of()* and *gsi_set_history_type()* to find the type of variables or parameters that have no value or that have a value but are quantity types.
- *libgsi*, *libtcp*, *libdec*, *libnet* and *librtl*, the G2 Gateway libraries, some combination of which you must link with your user code.

  On Windows only, *gsi.lib* and *gsi.dll* are used instead of *libgsi*, *libtcp*, and *librtl* for G2 Gateway delivered as a DLL.
- The sample make files, source files, object files, and executable files to support the sample G2 knowledge bases *gsi_exam.kb*, *itempass.kb*, and *mapchar.kb*, and the sample program named *skeleton*. For example, the files for the skeleton program on UNIX are named *skeleton.c*, *skeleton.o*, and *skeleton*.

The *skeleton.c* file contains stub versions of the GSI 4.1 standard callback functions that form the basis of your G2 Gateway user code. In order to link properly, your G2 Gateway bridge code must include at least the stub version of each callback function in *skeleton.c*. For information about how to write callback functions, see Callback Functions.

The sample knowledge base *mapchar.kb*, together with the *mapchar.c* file, demonstrates how a GSI application can read and write from a standard international character set, such as Korean KSC5601, and display the international characters in G2. *mapchar.c* contains code to translate between the Gensym character set, which can encode various characters such as Japanese kanji and Korean, and international character sets.

mapchar.c contains code translating between Gensym codes and:

- Korean: KSC5601-EUC
- Japanese: JISX0208-EUC
- Japanese: JISX0208-Shift-JIS
- Latin-1: ISO-8859-1
- Latin-Cyrillic: ISO-8859-5

For more information about the Gensym character set and descriptions of translation algorithms used in this file, see the *G2 Reference Manual*.

Compiling G2 Gateway on UNIX

On computers running UNIX, the G2 Gateway bridge process is a separate process executing on the same or a separate computer from the G2 process. The two processes communicate by using TCP/IP. In each arrangement, G2 assumes the role of a client process, and G2 Gateway acts as the server.

Configuration Requirements

There are no special configuration requirements for G2 Gateway on UNIX computers. However, you must have access to a C compiler.

Compiling and Linking G2 Gateway Applications on UNIX Platforms

To compile and link your G2 Gateway application on UNIX platforms, Gensym recommends that you create a make file to handle the tasks. Gensym provides a sample make file named *makefile*, which you can modify to suit your application.

Note The information in this section is presented as a set of guidelines for preparing a make file to compile and link your G2 Gateway application. Gensym assumes that you know how to prepare and use make files.

As you prepare your make file, specify the location of the following files, if they are not present or will not be stored in the current working directory when you run the make file:

- The G2 Gateway header files *gsi_main.h* and *gsi_misc.h*.
- The G2 Gateway libraries (use *libgsi.a*, *librtl.a*, and one of *libtcp.a* or *libnet.a*).
- Platform-specific libraries (such as *-ldl* on Solaris), and the system math library (such as *-lm*) that you will need for your application.
- The source files.
- The object modules.
- The executable files.

Specify only the G2 Gateway protocol libraries that you need.

Make certain that you specify the optimize flag (*-O*) before you specify the locations of the files.

When you are ready to run the make file, enter the following syntax at the command line:

make -f makefile-name

where *makefile-name* is the name of your make file.

Running the Bridge

To run your bridge executable at the command line, use the following syntax:

executable [listener [-network {tcpip | all}]]

where:

executable must be the first argument. Specifies the name of the bridge executable file.

listener must be the second argument (if used). Specifies the alphanumeric identifier for a network listener.

-network {tcpip|all} must be the final argument. Specifies the network protocol of the connection to listen on. The keyword *all* specifies that G2 Gateway looks for all types of network connections. This argument is also known as the **transport argument**.

If you do not specify a *listener* on the command line, TCP/IP connections are opened using the TCP/IP port number specified by:

- G2 Gateway callback function *gsi_get_tcp_port()*
- The default TCP/IP port number *22041* if *gsi_get_tcp_port()* is left as a stub.

For information about *gsi_get_tcp_port()*, see *gsi_get_tcp_port*.

If you do not specify a network protocol on the command line, G2 Gateway assumes that the *listener* is a TCP/IP port number if it contains only numbers and no letters.

The following example shows how to override G2 Gateway's algorithm for selecting a port:

```
#my_bridge 11111 -network all
```

In the example, the name of the executable is *my_bridge*, and one listener is opened at port *11111*.

Compiling G2 Gateway on Windows

On computers running Windows, the G2 Gateway bridge process is a separate process that can execute on the same computer as the G2 process or on a different computer. The two processes communicate by using TCP/IP. In each arrangement, G2 acts as the client process, and G2 Gateway acts as the server.

G2 Gateway runs on Windows platforms. However, it runs as both a Windows application and a console application (that is, it can link with *GUIxxx* libraries as well as *CONxxx* ones).

To compile a G2 Gateway application on a Windows platform, you must use the Microsoft Visual C++ compiler, Version 6.0 or later, which includes Visual Studio .NET 2003.

Caution If you compile your user code with a compiler other than the Microsoft Visual C++ compiler Version 6.0 or later, or Intel® C++ Compiler 8.0 for Windows, you may not be able to link your object files with G2 Gateway libraries.

For a DLL bridge, link in *gsi.lib*.

G2 Gateway provides a Microsoft Visual C++ project file that you can use to build your bridge. The project file consists of a project workspace called *bridges.dsw* and eight projects, one for each bridge. Similar to the *makefile*, each project uses *gsimmain.c*, so they are all Windows applications. The projects also use *libgsi.lib*, rather than *gsi.dll*.

Configuration Requirements

There are no special configuration requirements for G2 Gateway applications running on Windows platforms.

Compiling and Linking G2 Gateway on Windows

On Windows platforms, G2 Gateway can be delivered either as a DLL or as a non-DLL library. Windows is the only platform on which G2 Gateway can be delivered as a DLL.

The following libraries are provided for G2 Gateway on Windows: *libgsi.lib*, *librtl.lib*, *libtcp.lib*, *libmmt.lib*, *libicrmt.lib*, and *gsi.lib*. Which of these libraries you link in depends on whether your G2 Gateway bridge is a DLL or non-DLL:

- For a non-DLL bridge, link in: *libgsi.lib*, *librtl.lib*, *libtcp.lib*, *libmmt.lib*, and *libicrmt.lib*.
- For a DLL bridge, link in: *gsi.lib*.

If you are creating a new G2 Gateway application, delivering G2 Gateway as a DLL can result in a smaller executable and save disk space. If you are upgrading your G2 Gateway application, delivering G2 Gateway as a non-DLL does not produce a significantly smaller executable.

You can compile and link your G2 Gateway application on Windows in the following ways, as a Windows application and as a console application.

As a Windows application:

- Using *gsi_main.c* and *gsimmain.c* without editing them.

  gsimmain.c performs special initializations required only on Windows platforms, and then calls the *main()* function that you define in the *gsi_main.c* file. You do not need to make any changes to *gsimmain.c*.
- Editing the *main()* and *WinMain()* procedures before compiling and linking *gsi_main.c* and *gsimmain.c*. Use this option if your application includes Windows code, which you place in *WinMain()*.

As a console application:

- Using *gsi_main.c* without editing it.
- Include the *main()* procedure from *gsi_main.c* within your own application with whatever edits it requires.

Note GSI 4.x supported GSI only as a console application. G2 Gateway 5.0 Rev. 0 supported only Windows applications. Beginning with 5.0 Rev. 1, G2 Gateway supports both windows and console applications.

You can compile and link a G2 Gateway as a DLL only when you define the C preprocessor flag *GSI_USE_DLL*. See Using Standard Callback Functions for more information.

Note Linking G2 Gateway causes files named *skeleton.lib* and *skeleton.exp* to be created. These files are not needed, and you can delete them when you finish linking.

Using gsi_main.c and gsimmain.c without Modification (Windows Applications)

To link with the G2 Gateway DLL, using `gsi_main.c` and `gsimmain.c`:

1. Compile *gsi_main.c*, with the C preprocessor flag *GSI_USE_DLL* defined.
2. Compile *gsimmain.c*.
3. Link using *$(guilflags)*, *gsi_main.obj*, *gsimmain.obj*, *gsi.lib*, and *$(guilibsmt)*.

Using gsi_main.c without Modification (Console Applications)

To link with the G2 Gateway DLL, using `gsi_main.c`:

1. Compile *gsi_main.c*, with the C preprocessor flag *GSI_USE_DLL* defined.
2. Link using *$(guilflags)*, *gsi_main.obj*, *gsi.lib*, and *$(guilibsmt)*.

Using a Customized WinMain() and main()

To link with the G2 Gateway DLL, using a user-written `main()` and `WinMain()`:

1. Make sure that your G2 Gateway user code includes a *WinMain()* procedure. If your code does not have a *WinMain()* procedure, create one using *gsimmain.c* as an example.
2. Call *gsi_initialize_for_win32()* in your *WinMain()*, as in this example from *gsimmain.c*:

   ```
   gsi_initialize_for_win32(hInstance, lpCmdLine);
   ```

   gsi_initialize_for_win32() performs Windows-specific initialization and looks for a *-log* argument in the command line. If the *-log* argument is present, it redirects console output to a file whose name is specified by the argument following *-log*. To do this, it needs certain handles from the operating system, so in the call to *gsi_initialize_for_win32()*, the first argument must correspond to the first parameter of *WinMain()*, and the second argument must correspond to the third parameter of *WinMain()*. If you do not intend to use *-log* from your bridge, you could just pass *NULL* for the arguments, but you still must call the function.

3 Call *gsi_set_include_file_version()* to specify the major, minor, and revision file version of the include file, as follows:

```
gsi_set_include_file_version(GSI_INCLUDE_MAJ_VER_NUM,
GSI_INCLUDE_MIN_VER_NUM, GSI_INCLUDE_REV_VER_NUM);
```

Note For a G2 Gateway delivered as a DLL, you cannot use the variables *gsi_include_file_major_version* to specify the major include file version, *gsi_include_file_minor_version* to specify the minor include file version, or *gsi_include_file_revision_version* to specify the revision.

---

4 Declare and install any callback functions that you may use. See Using Standard Callback Functions for instructions.

5 Include a call to *GSI_SET_OPTIONS_FROM_COMPILE()* before the call to *gsi_start()* in your *main()* function.

6 Make any desired calls to *gsi_set_option()*.

7 Call *gsi_start(argc, argv)*. You must supply *argc* and *argv*, possibly by parsing *lpCmdLine*.

8 Link using *$(guilflags)* and *$(guilibsmt)*.

Using a Customized main()

To link with the G2 Gateway DLL, using a user-written `main()`:

1 Make sure that your G2 Gateway user code includes a *main()* procedure. If your code does not have a *main()* procedure, create one using *gsi_main.c* as an example.

2 Call *gsi_initialize_for_win32()* in your *main()*, as in this example from *gsi_main.c:*

```
gsi_initialize_for_win32(NULL, NULL);
```

gsi_initialize_for_win32() performs Windows-specific initialization. In a Windows application, it also allows you to specify a log file, but that functionality is not supported for console applications.

Compiling and Linking G2 Gateway Applications on Windows Platforms

The steps that you must follow to compile and link a G2 Gateway application on an Windows platform are similar to the steps that you must follow on a Unix platform.

To compile and link your G2 Gateway application on Windows platforms, Gensym recommends that you create a make file to handle the tasks. Gensym

provides a sample make file named *makefile*, which you can modify to suit your application.

As you prepare your make file, specify the location of the following files, if they are not present or will not be stored in the current working directory when you run the make file:

- The G2 Gateway header files *gsi_main.h* and *gsi_misc.h*
- The G2 Gateway libraries *libgsi.lib*, *libtcp.lib*, and *librtl.lib*

  If G2 Gateway is delivered as a DLL (possible only on Windows), it is delivered in the *gsi.dll* and *gsi.lib* (rather than *libgsi.lib*, *libtcp.lib*, and *librtl.lib*).
- Platform-specific libraries that you will need for your application
- The source files
- The object modules
- The executable files

When you are ready to run the make file, enter the following syntax at the command line:

```
nmake -f makefile-name
```

where *makefile-name* is the name of your make file.

Compiling and Linking a Windows Application

You compile and link your G2 Gateway Windows application on an Intel-based Windows platform by using the following command:

```
cl source-file(s).c gsi_main.obj gsimmain.obj libgsi.lib libtcp.lib
librtl.lib libc.lib kernel32.lib advapi32.lib user32.lib
wsock32.lib gdi32.lib
    -Feexecutable.exe
    -Iinclude-path -link -subsystem:windows,4.0
    -entry:WinMainCRTStartup
```

where:

source-file(s).*c* are the source file or files of your G2 Gateway bridge

executable.exe is the name that you give to the executable that you create when you compile and link the bridge

include-path is the search path for an include file. Use the *include-path* syntax as shown below:

```
c:\Program Files\gensym\g2-2015\gsi
```

```
cl skeleton.c gsi_main.obj gsimmain.obj libgsi.lib libtcp.lib
librtl.lib libc.lib kernel32.lib advapi32.lib user32.lib
wsock32.lib gdi32.lib -Feskeleton.exe
-Ic:\Program Files\gensym\g2-2015\gsi -link
-subsystem:windows,4.0 -entry:WinMainCRTStartup
```

Note When using a make file with Visual Studio .NET 2003, you must add `/nodefaultlib:libc` to the `-link` line. When using a Project file with Visual Studio .NET 2003, choose Project > Properties > Configuration Properties > Linker > Input and change `Ignore Specific Library = libc`.

Compiling and Linking a Console Application

You can also compile and link your G2 Gateway console application on an Intel-based Windows platform by using the following command:

```
cl source-file(s).c gsi_main.obj libgsi.lib libtcp.lib librtl.lib
libc.lib kernel32.lib advapi32.lib user32.lib wsock32.lib
gdi32.lib
    -Feexecutable.exe
    -Iinclude-path -link -subsystem:console
    -entry:mainCRTStartup
```

where:

source-file(s)`.c` are the source file or files of your G2 Gateway bridge

executable.exe is the name that you give to the executable that you create when you compile and link the bridge

include-path is the search path for an include file. Use the *include-path* syntax as shown below:

```
c:\Program Files\gensym\g2-2015\gsi
```

```
cl skeleton.c gsi_main.obj libgsi.lib libtcp.lib librtl.lib
libc.lib kernel32.lib advapi32.lib user32.lib wsock32.lib
gdi32.lib -Feskeleton.exe -Ic:\Program Files\gensym\
g2-2015\gsi -link -subsystem:console -entry:mainCRTStartup
```

Note When using a make file with Visual Studio .NET 2003, you must add */nodefaultlib:libc* to the *-link* line. When using a Project file with Visual Studio .NET 2003, choose Project > Properties > Configuration Properties > Linker > Input and change *Ignore Specific Library = libc*.

Running the Bridge

To run your bridge executable at the command line, use the following syntax:

executable [listener [-network tcpip]]

where:

executable must be the first argument specified, and is the name of the bridge executable file.

listener must be the second argument specified (if used), and is the alphanumeric identifier for a network listener.

-network tcpip must be the final argument specified.

An example is:

```
#my_bridge 11111 -network tcp/ip
```

In this example, the name of the executable is *my_bridge*, and 11111 is the TCP/IP listener that is opened.

If you omit *listener* from the command line, a TCP/IP connection is opened using the TCP/IP port number specified by:

- The G2 Gateway callback function *gsi_get_tcp_port()*, or
- The default for TCP/IP (*22041*), if *gsi_get_tcp_port()* is left as a stub.

For information about *gsi_get_tcp_port()*, see *gsi_get_tcp_port*.

Command-Line Options and Arguments

In the command line that starts a G2 Gateway bridge process, you can optionally include one or more of the following command line options:

- General command line options: *help*, *rgn1lmt*, *rgn2lmt*, *tcpipexact*, and *tcpport*.
- Command line switches for initiating connections to G2: *nolistener*, *noconnect*, *connect*.
- Command line arguments for initiating connections to G2: *connect-interface-name interface_name*, *connect-class-name class_name*,

connect-network *network*, *connect-host* *host_name*, *connect-port* *port_number*, *connect-initialization-string* *string*.

- Command line options for secure communication: *secure* and *cert*.

You can also set command-line options in your user code, using the *GSI_SET_OPTIONS_FROM_COMPILE*() C macro. *GSI_SET_OPTIONS_FROM_COMPILE()* is called from *gsi_main.c* to generate the code that sets runtime G2 Gateway options. It calls *gsi_set_option()* to set these options, based on settings of the G2 Gateway C preprocessor macros. For information about this function, see *Defining C Preprocessor Flags*.

The following sections describe these options in detail.

cert

Specifies the SSL server certificate to use.

Platforms

All platforms

Syntax

Windows:

`-cert name`

name: The Common Name (CN) of the SSPI certificate in the local machine's `my` certificate store.

UNIX:

`-cert file`

file: The name of the OpenSSL server certificate to use, where *file* is a file containing a private key and a certificate in PEM format, which consists of the DER format base64 encoded with additional header and footer lines.

Equivalent Environment Variable

`G2_CERT`

Description

You specify the certificate when encrypting communication, using the `-secure` command-line option.

help

Directs a new G2 Gateway bridge process to output the text of the syntax of all G2 Gateway bridge command line options to the window from which the process was launched.

Platforms

All platforms

Syntax

```
-help
```

Equivalent Environment Variable

None.

Description

This option directs the new G2 Gateway bridge process to output text that describes the syntax of each support command line option, then to exit.

The G2 Gateway bridge writes the help text to the launch window.

Example

This command directs the G2 Gateway bridge process to output the text of the syntax of all G2 command line options, then exit. Unlike all other options, which must come after the listener (port name or number) on the command line, help must be the first or last option after the bridge:

```
g2dbbridge tw -help
```

log

Redirects command window output to a specified file.

Platforms

All platforms

Syntax

-log file

Equivalent Environment Variable

None.

Description

This option redirects the command window output to the file whose pathname is specified by the *file* argument following *-log*.

On Windows, this option is only supported if it is passed to *gsi_initialize_for_win32* as part of the *lpCommandLine*.

rgn1lmt

Specifies the initial supply of available memory for G2 Gateway data other than symbols and graphics images.

Platforms

All platforms

Syntax

`-rgn1lmt` *number-of-bytes*

number-of-bytes: The integer 800,000 or higher, up to the maximum per-process allocation determined by your platform's operating system settings. The only exception to this minimum is the Compaq Tru64 Unix operating system, which requires 2,900,000 bytes.

Equivalent Environment Variable

`G2RGN1LMT=<`*number-of-bytes*`>`

where *number-of-bytes* has the same requirements as `-rgn1lmt`.

If set, this environment variable determines the initial supply of available memory in the Region 1 memory pool of G2, TeleWindows, and G2 Gateway. You can override this setting for G2 Gateway by issuing the `env` command before the command to invoke G2 Gateway or G2 (whichever is more convenient). For example:

`env G2RGN1LMT=<`*number-of-bytes*`> bridge-invocation-command`

This changes the setting for all G2, TeleWindows, and G2 Gateway processes running on your system.

Description

G2 Gateway maintains its supply of available memory in three regions. This option controls the initial supply of available memory in its Region 1 memory pool. G2 Gateway uses its Region 1 memory pool to store all data other than symbols and graphics images.

The new G2 Gateway process allocates Region 1 memory when it is launched. G2 Gateway's standard output message at start-up reflects this fact.

For a G2 Gateway process the default amount of Region 1 memory is 800,000 bytes. The default for the Compaq Tru64 Unix operating system is 3,500,000 bytes.

Special Considerations

If your `-rgn1lmt` option specifies less than the minimum number of bytes, G2 Gateway displays a warning standard output message and supplies the minimum amount. Do not include commas when specifying *number-of-bytes*.

Example

This command starts a new G2 Gateway process and directs it to allocate 8,500,000 bytes as its initial supply of Region 1 memory:

```
g2dbbridge -tcpport 22041 -rgn1lmt 8500000
```

G2 Gateway attempts to allocate more Region 1 memory as is required by the connected G2's processing.

Additional Information

For information about G2 memory management, see the chapter on memory management in the *G2 Reference Manual*.

rgn2lmt

Specifies the initial supply of available memory for G2 Gateway symbol data.

Platforms

All platforms

Syntax

`-rgn2lmt` *number-of-bytes*

number-of-bytes: The integer 300,000 or higher, up to the maximum per-process allocation determined by your platform's operating system settings.

Equivalent Environment Variable

`G2RGN2LMT=<`*number-of-bytes*`>`

where *number-of-bytes* has the same requirements as `-rgn2lmt`.

If set, this environment variable determines the initial supply of available memory in the Region 2 memory pool of G2, TeleWindows, and G2 Gateway. You can override this setting for G2 Gateway by issuing the `env` command before the command to invoke G2 Gateway or G2 (whichever is more convenient). For example:

`env G2RGN2LMT=<`*number-of-bytes*`> bridge-invocation-command`

This changes the setting for all G2, TeleWindows, and G2 Gateway processes running on your system.

Description

G2 Gateway maintains its supply of available memory in three regions. This option controls the initial supply of available memory in its Region 2 memory pool. G2 Gateway uses its Region 2 memory pool to store symbols and related internal data.

The new G2 Gateway process allocates Region 2 memory when it is launched. G2 Gateway's standard output message at start-up reflects this fact.

The default amount of Region 2 memory is 500,000 bytes.

Special Considerations

If your `-rgn2lmt` option specifies less than the minimum number of bytes, G2 Gateway displays a warning standard output message and supplies the minimum amount. Do not include commas when specifying *number-of-bytes*.

Example

This command starts a new G2 Gateway process and directs it to allocate 4,500,000 bytes as its initial supply of Region 2 memory:

```
g2dbbridge -tcpport 22041 -rgn1lmt 4500000
```

G2 Gateway attempts to allocate more Region 2 memory as is required by the connected G2's processing.

Additional Information

For information about G2 memory management, see the chapter on memory management in the *G2 Reference Manual*.

secure

Platforms

All platforms

Syntax

-secure

Use SSL on all TCP/ICP connections.

Equivalent Environment Variable

None

Description

This command-line option causes G2 Gateway to use SSL on all TCP/ICP connections. G2 Gateway uses SSPI on Windows and OpenSSL on UNIX.

To access SSL, you need to include the following libraries, depending on your platform:

UNIX	Windows
libgsec.a	*libgsec.lib*

If you do not want to use SSL, you need to include the following libraries instead:

UNIX	Windows
libnogsec.a	*libnogsec.lib*

Failure to include one of these libraries or attempts to include both results in link errors.

In addition, you must also provide the following platform-specific libraries:

- Windows: *crypt32.lib*, available with your Microsoft compiler.
- Solaris, Linux, HP-UX, IBM AIX: *libssl.a* and *libcrypto.a*, which are supplied with G2 Gateway. Note that you must supply these two libraries in exactly this order; failure to do so will result in link errors.
- HP-UX: You must also include *libgcc.a*, also provided with G2 Gateway.

On the Windows platforms, the default *gsi.dll* is linked without SSL support; a separate library *gsi_ssl.dll* is provided to include SSL support as a DLL.

Currently, G2 Gateway does not support SSL on the alphaosf platform, but *libnogsec.a* must be linked in anyway. The example is not present.

The example *makefile* for G2 Gateway compiles most of the examples without SSL support. The *skeleton_ssl* example includes SSL support.

Attempting to give the *-secure* option to a G2 Gateway bridge that has not been linked with SSL support results in a warning message; however, the bridge will start up normally, but without SSL support.

Upon startup, a bridge gives the port number with /SSL appended when *-secure* is requested and available. For example:

```
GSI Version 8.3 Rev. 0 IBM POWERstation (JA28)
2007-01-30 15:00:05   Waiting to accept a connection on:
2007-01-30 15:00:05   TCP_IP:cs-aix4:22000/SSL
```

To establish a secure connection and test the secure status, use these procedures:

- *gsi_establish_secure_listener*.
- *gsi_initiate_secure_connection*.
- *gsi_initiate_secure_connection_with_user_data*.
- *gsi_context_is_secure*.
- *gsi_current_context_is_secure*.

Note that if G2 is not listening for secure connections, this connection fails and G2 Gateway becomes inoperative. We recommend that you determine whether G2 is listening securely before executing either of these procedures.

To establish a GSI connection with security, use the secure yes option in the gsi-connection-configuration attribute, after the host and port number. For example:

```
tcp-ip host "localhost" port-number 22044 secure yes
```

For G2-G2 connections, use the icp-connection-specification attribute.

Specifying the secure yes option attempts to make a secure connection to the port number on the specified host. Note that if the host is not listening for secure connections on the specified port, this connection fails and G2 becomes inoperative. If no host is listening at the port, then the connection simply fails.

In addition, the gsi-interface class defines the gsi-interface-is-secure attribute, and the g2-to-g2-interface class defines the interface-is-secure attribute, whose value

is yes or no, which determines whether or not security was established on the connection from the remote system.

Note that you cannot make a secure G2-to-G2 connection to the same G2. This condition is detected, and an insecure connection is created instead, with a warning on the logbook.

For OpenSSL copyright information, see the *readme-g2.html* file.

tcpipexact

Prohibits the new G2 Gateway bridge process from attempting to open a network connection to any TCP/IP port other than that specified by the accompanying *-tcpport* command line option.

Platforms

All platforms

Syntax

```
-tcpipexact
```

Equivalent Environment Variable

None.

Description

This option directs the new G2 Gateway bridge process to exit before completing its startup, if it cannot open a network connection to the TCP/IP port specified by the *-tcpport* command line option.

This option requires that you also include the *-tcpport* option in the command line that launches this G2 Gateway bridge process. The G2 Gateway bridge ignores this option unless the command line also includes the *-tcpport* option.

Example

This command starts a G2 Gateway bridge process that attempts to open a network connection to the TCP/IP port *22041*:

```
g2dbbridge -tcpport 22041 -tcpipexact
```

If this attempt is not successful, the bridge process does *not* attempt to open a network connection to another TCP/IP port, and automatically exits.

tcpport

Directs the new G2 Gateway bridge process to open a network connection to the specified TCP/IP port, with additional attempts to connect to other TCP/IP ports as necessary.

Platforms

All platforms

Syntax

`-tcpport` *tcpip-port-number*

tcpip-port-number: A positive integer; however, TCP/IP port numbers under 1000 are often reserved by your platform and should be avoided for use with a G2 Gateway bridge process.

Equivalent Environment Variable

None.

Description

This option directs the new G2 Gateway bridge process to open a network connection to a TCP/IP port. Specify the port's name as an argument, following the option.

If the new G2 Gateway bridge process cannot open a connection to the specified port, this option also directs the bridge process to attempt to open a connection to the default TCP/IP port `22041`. If this is not successful, the G2 Gateway bridge process increments the port's last digit (to `22042`), attempts to connect to that port, and so on. The bridge process stops after trying to connect to TCP/IP port `22140` — that is, after incrementing the default TCP/IP port number 100 times.

Special Considerations

Including the `-tcpipexact` option in the command line modifies how the new G2 Gateway bridge process attempts to open a TCP/IP network connection. See the section `tcpipexact`.

The first argument after the bridge must be the port number. The only exception for this is if you are using the -help option, in which case it must come first.

Example

This command starts a G2 Gateway bridge process that attempts to open a network connection to TCP/IP port *22041*:

```
g2dbbridge -tcpport 22041
```

Command Line Switches for Initiating Connections to G2

The following table describes G2 Gateway command line switches that support the ability of G2 Gateway to initiate connections to G2.

G2 Gateway Command-Line Switches

Switch	Description
-nolistener	Inhibits establishment of listener(s). Suppresses the G2 Gateway default launch behavior of automatically creating TCP/IP listeners based on the specification on the remainder of command line. You can use the API function *gsi_establish_listeners()* to create the specified listeners later -- for example, after you log in or confirm that your system is ready. This has the additional benefit of preventing a *gsi-interface* from connecting on the G2 side until your application has decided it is ready. This switch does not affect the ability of G2 Gateway to establish a connection to a G2.

G2 Gateway Command-Line Switches

Switch	Description
–*noconnect*	Use this switch to specify that no initial connection to G2 is established. Any arguments specified in the G2 Gateway command line provide default values for the call to *gsi_initiate_connection()*.
-connect	When you use this switch, an initial connection to G2 is established, before *gsi_set_up()* is called. The connection is established using values specified in command line arguments. Default values are used for any unspecified arguments. The connection is established without a call to the API function *gsi_initiate_connection()*. However, your G2 Gateway bridge process can subsequently call *gsi_initiate_connection()* to establish additional connections to G2 processes.

Note If you specify neither *-connect* or *-noconnect*, G2 Gateway nevertheless establishes a connection to G2 if you specify at least one of the G2 Gateway connection command line arguments (see below).

If you specify neither the *-connect* switch nor the G2 Gateway connection command line arguments, the bridge does not establish a connection.

Command-Line Arguments for Initiating Connections to G2

The following table describes the G2 Gateway command line arguments that support the ability of G2 Gateway to establish a connection to a G2 process.

G2 Gateway Command-Line Arguments

Argument	Description
-connect-interface-name interface_name	The name of a GSI interface that G2 Gateway creates in the G2 knowledge base to which G2 Gateway establishes a connection. G2 uses this GSI interface to communicate with the G2 Gateway that establishes the connection. **Note:** Specify *interface_name* using all upper case, for example: *-connect-interface-name MY-INTERFACE*
-connect-class-name class_name	The name of an existing class definition in the G2 knowledge base. The GSI interface *interface_name* is created as an instance of this class. If you omit the *-connect-class-name* argument, G2 Gateway creates the GSI interface as an instance of the G2 class gsi-interface. **Note:** Specify *class_name* using all upper case, for example: *-connect-class-name MY-CLASS*

G2 Gateway Command-Line Arguments

Argument	Description
-connect-network network	The network protocol used on the connection to the G2. Specify *TCP-IP*. You need to specify only the first letter (T), in upper or lower case. The value that you specify for *network* is the default for the API function *gsi_initiate_connection()*.
-connect-host host_name	Specify the host name of the computer where the G2 is listening for a connection to this G2 Gateway. The value that you specify for *host_name* is the default for the API function *gsi_initiate_connection()*.
-connect-port port_number	Specify the number of the port on which the G2 process is listening for a connection to this G2 Gateway. The value that you specify for *port_number* is the default for the API function *gsi_initiate_connection()*. The first argument after the bridge must be the port number. The only exception for this is if you are using the *-help* option, in which case it must come first.
-connect-initialization-string string	A string that is written to the remote-process-initialization-string attribute of the GSI interface created by this G2 Gateway. When the connection is established, G2 Gateway passes this string to the callback function *gsi_initialize_context()*.

Starting a G2 Gateway Bridge from within G2

Typically, you start the G2 Gateway bridge process before you start the G2 process. When the G2 is launched successfully, the GSI interface used by G2 to connect to the bridge process then becomes active, which causes G2 to attempt to establish a connection to the bridge. Thereafter, you deactivate, then activate, the GSI interface each time you want G2 to attempt to make a connection to the bridge.

However, you can also start and stop your G2 Gateway bridge process from a procedure that you have written in G2. Using the approach described in this appendix, you start the G2 process first, start the bridge process next, and then activate the GSI interface. You can activate the GSI interface separately from the G2 launch if you place it on an activatable subworkspace.

To start the bridge process from G2, you perform the following tasks:

1. Choose an object on whose subworkspace you will place the GSI interface used by G2 to connect to the bridge
2. Create G2 parameters to represent the bridge process pathname, TCP/IP port number, and unique identification (so that G2 can identify the process)
3. Create a procedure that starts the bridge process
4. Create a procedure that stops the bridge process
5. Create one or two procedures that:
   a. Activate the subworkspace on which you have placed the GSI interface
   b. Deactivate the subworkspace containing the GSI interface

These tasks are described in their own selections procedures described are provided as guidelines for your own procedures, which may vary from those presented here. See the *G2 Reference Manual* for instructions about creating and using G2 procedures.

Note The G2 Gateway process that you spawn using the system procedure g2-spawn-process-to-run-command-line() must be on the local machine.

You can start a G2 Gateway process on a remote machine only if that machine has a Telewindows session into the G2 from which you are starting GSI. In this case, you can start the G2 Gateway process on the remote machine using the system procedure g2-spawn-remote-process().

Placement of the GSI Interface

Place the GSI interface that you use to connect to the bridge on the subworkspace of another object in G2. If the class has this capability, its subworkspaces are deactivated automatically when G2 starts, but can be activated with the activate action. In addition, the subworkspaces that you have activated can be deactivated with the deactivate action.

The GSI interface must be on a deactivated subworkspace when G2 starts, so that G2 does not automatically look for the connection to the bridge as soon as G2 is launched (based on the information provided by the GSI interface).

Representing the Bridge Process Information

Create G2 parameters to represent the following information:

- Create a text parameter to represent the pathname of the bridge executable file. For its initial-value attribute, specify the pathname of the executable file. An example is:

  "*/home/gsi/gsi-app*"

- For TCP/IP, create an integer parameter to represent the bridge process port number. For its Initial-value attribute, specify the port number. An example is:

  22040

- Create a floating point variable to represent a unique bridge process identification. Its value will be returned to it by a procedure you write. A floating point number is used because process IDs frequently use all 32 bits of a long word, whereas G2 integers can support only 30-bit integers.

  See the *G2 Reference Manual* for information about creating G2 parameters.

Stopping G2 Gateway from within G2

To stop the bridge process from within G2, you must write two procedures that stop the bridge process and deactivate the subworkspace containing the GSI interface.

You can also write a local procedure in G2 Gateway which, when called as a remote procedure by G2, cleans up and exits.

Part III

Appendixes

Appendix A: Functions by Argument and Return Type

Lists the API and callback functions provided by G2 Gateway, grouped by the data types of their arguments and their return values.

Appendix B: Constants

Lists symbolic constants defined in G2 Gateway header files.

Appendix C: G2 Gateway Error Messages

Lists and describes the standard error messages returned by G2 Gateway.

Appendix D: G2 Gateway Data Types

Describes the data types defined for use in G2 Gateway user code.

Appendix E: Limits and Ranges

Describes limits and ranges applicable in G2 Gateway.

Appendix F: How G2 and G2 Gateway Exchange Data

Provides a brief summary of techniques for exchanging data between a G2 Gateway bridge and a G2 KB.

Appendix G: Upgrading G2 Gateway Applications

Describes how to upgrade existing GSI applications to the current version of G2 Gateway.

A

Functions by Argument and Return Type

Lists the API and callback functions provided by G2 Gateway, grouped by the data types of their arguments and their return values.

Introduction

This chapter lists the APIs and callback functions grouped by the data types of their arguments and their return values.

Functions by Argument Type

The following table lists API and callback functions grouped by the types of their arguments. In each group, the arguments of the given type appears in **bold** characters.

Note The curly braces ({}) around a function name indicate that this function is a callback function defined by the user. The brackets ([]) around an argument indicate that this argument is supplied if the C preprocessor macro *GSI_USE_USER_DATA_FOR_CALLBACKS* is defined and is not supplied otherwise.

Functions Grouped by Argument Type

HANDLE

*gsi_initialize_for_win32(***hInstance***, lpCmdLine)*

char *

gsi_initialize_for_win32(hInstance, **lpCmdLine***)*
*gsi_convert_string_to_unicode(***string***, style)*
*gsi_initialize_callbacks(***name1***, ...)*

char **

gsi_start(argc, **argv***)*

double *

gsi_set_flt_array(item_or_attr, **doubles_array***, gsi_int count)*
gsi_set_flt_list(item_or_attr, **doubles_array***, count)*
gsi_set_history(item_or_attr, values, **timestamps***, count, type,*
maximum_count, maximum_age, min_interval)

double **

gsi_extract_history
(item, values_address, **timestamps_address***, type_address)*

double

gsi_decode_timestamp
*(***timestamp***, year_address, month_address, day_address,*
hour_address, minute_address, second_address)
gsi_set_flt(item_or_regitem_or_attr, **float_value***)*
gsi_set_timestamp(item_or_attr, **timestamp***)*

gsi_attr *

*gsi_reclaim_attrs(***attributes***)*
*gsi_reclaim_attrs_with_items(***attributes***)*
gsi_return_attrs(registered_item, **attributes***, count, context)*
gsi_return_timed_attrs(registered_item, **attributes***, count,*
context)
gsi_set_attrs(item_or_attr, **new_attributes***, count)*

Functions Grouped by Argument Type

gsi_attr

*gsi_attr_is_transient(***attribute***)*
*gsi_attr_name_is_qualified(***attribute***)*
*gsi_attr_name_of(***attribute***)*
*gsi_class_qualifier_of(***attribute***)*
*gsi_item_of_attr(***attribute***)*
gsi_set_attr_by_name(item_or_attr, search_name, **source_attr***)*
*gsi_set_attr_is_transient(***attribute***, new_value)*
*gsi_set_attr_name(***attribute***, attribute_name)*
*gsi_set_class_qualifier(***attribute***, attribute_name)*
*gsi_set_item_of_attr(***attribute***, source_item)*
*gsi_set_unqualified_attr_name(***attribute***, attribute_name)*
*gsi_unqualified_attr_name_of(***attribute***)*

gsi_call_identifier_type

gsi_rpc_call
*(function_handle, arguments, [***call_identifier***], context)*
gsi_rpc_call_with_count
*(function_handle, arguments, count,[***call_identifier***], context)*
gsi_rpc_return_error_values
(arguments, count, **call_identifier***, context)*
gsi_rpc_return_values
(arguments, count, **call_identifier***, context)*
{gsi_rpc_local_fn}
([procedure_user_data], rpc_arguments, count, **call_identifier***)*
{gsi_rpc_receiver_fn}
*([procedure_user_data], rpc_arguments, count, [***call_identifier***])*

Functions Grouped by Argument Type

gsi_char *

gsi_establish_listener(**network**, **port**, *exact*)
gsi_initiate_connection
(**interface_name**, **interface_class_name**,
keep_connection_on_g2_reset, **network**, **host**, **port**,
remote_process_initialization_string)
gsi_initiate_connection_with_user_data
(**interface_name**, **interface_class_name**,
keep_connection_on_g2_reset, **network**, **host**, **port**,
remote_process_initialization_string, *context_user_data*)
gsi_make_symbol(**name**)
gsi_return_message(**message**, *context*)
gsi_rpc_declare_local(*local_function*, *[procedure_user_data]*,
g2_function_name)
gsi_rpc_declare_remote
(*function_handle*, **g2_function_name**, *receiver_function*,
[procedure_user_data], *argument_count*, *return_count*, *context*)
gsi_rpc_declare_remote_with_error_handler_and_user_data
(*function_handle*, **g2_function_name**, *receiver_function*,
error_receiver_function, *[procedure_user_data]*,
argument_count, *return_count*, *context*)
gsi_set_rpc_remote_return_value_kind
(*function_handle*, *return_value_index*, **kind**)
gsi_set_str(*item_or_regitem_or_attr*, **text_value**)
gsi_signal_error
(**originating_function_name**, *user_error*, **message**)
{gsi_error_handler}
(*error_context*, *error_code*, **error_message**)
{gsi_initialize_context}
(**remote_process_initialization_string**, *length*)
{gsi_missing_procedure_handler}(**name**)
{gsi_receive_message}(**message**, *length*)

gsi_char **

gsi_set_str_array
(*item_or_attr*, **text_values_array**, *gsi_int count*)
gsi_set_str_list
(*item_or_attr*, **text_values_array**, *gsi_int count*)

Functions Grouped by Argument Type

gsi_context_user_data_type

gsi_initiate_connection_with_user_data
(interface_name, interface_class_name,
keep_connection_on_g2_reset, network, host, port,
remote_process_initialization_string, **context_user_data***)*
gsi_set_context_user_data(context, **context_user_data***)*

gsi_function_handle_type *

gsi_rpc_declare_remote
*(***function_handle***, g2_function_name, receiver_function,*
[procedure_user_data], argument_count, return_count, context)
gsi_rpc_declare_remote_with_error_handler_and_user_data
*(***function_handle***, g2_function_name, receiver_function,*
error_receiver_function, [procedure_user_data],
argument_count, return_count, context)

gsi_function_handle_type

gsi_rpc_call
*(***function_handle***, arguments, [call_identifier], context)*
gsi_rpc_call_with_count
*(***function_handle***, arguments, count, [call_identifier],*
context)
*gsi_rpc_start(***function_handle***, arguments, context)*
gsi_rpc_start_with_count
*(***function_handle***, arguments, count, context)*
gsi_set_rpc_remote_return_value_kind
*(***function_handle***, return_value_index, kind)*
gsi_set_rpc_remote_return_exclude_user_attrs
*(***function_handle***, attributes)*
gsi_set_rpc_remote_return_include_system_attrs
*(***function_handle***, attributes)*
gsi_set_rpc_remote_return_include_all_system_attrs_except
*(***function_handle***, attributes)*

Functions Grouped by Argument Type

gsi_int *

gsi_decode_timestamp
(timestamp, **year_address**, **month_address**, **day_address**,
hour_address, **minute_address**, **second_address***)*
gsi_extract_history
(item, values_address, timestamps_address, **type_address***)*
gsi_extract_history_spec
(item, **maximum_count_address**, **maximum_age_address**,
minimum_interval_address*)*
*gsi_initialize_error_variable(***error_variable_address***)*
gsi_set_int_array(item_or_attr, **integers_array**, *count)*
gsi_set_int_list(item_or_attr, **integers_array**, *count)*
gsi_set_log_array
(item_or_attr, **truth_values_array**, *gsi_int count)*
gsi_set_log_list
(item_or_attr, **truth_values_array**, *gsi_int count)*

Functions Grouped by Argument Type

gsi_int

*gsi_convert_string_to_unicode(string, **style**)*
*gsi_convert_unicode_to_string(string, **style**)*
*gsi_convert_unicode_to_wide_string(string, **style**)*
*gsi_convert_wide_string_to_unicode(string, **style**)*
*gsi_context_received_data(**context**)*
*gsi_context_socket(**context**)*
*gsi_context_user_data(**context**)*
*gsi_encode_timestamp(**year**, **month**, **day**, **hour**, **minute**, **second**)*
*gsi_error_message(**error_code**)*
*gsi_establish_listener(network, port, **exact**)*
*gsi_flush(**context**)*
gsi_identifying_attr_of(registration,
*   **identifying_attribute_index**)*
gsi_initiate_connection
*   (interface_name, interface_class_name,*
*   **keep_connection_on_g2_reset**, network, host, port,*
*   remote_process_initialization_string)*
gsi_initiate_connection_with_user_data
*   (interface_name, interface_class_name,*
*   **keep_connection_on_g2_reset**, network, host, port,*
*   remote_process_initialization_string, context_user_data)*
gsi_item_of_identifying_attr_of(registration,
*   **identifying_attribute_index**)*
*gsi_kill_context(**context**)*
*gsi_make_array(**count**)*
*gsi_make_attrs(**count**)*
*gsi_make_attrs_with_items(**count**)*
*gsi_make_items(**count**)*
*gsi_make_registered_items(**count**)*
*gsi_option_is_set(**option_index**)*
*gsi_registration_of(**item_handle**, **context**)*
*gsi_registration_of_handle(**item_handle**, **context**)*
*gsi_reset_option(**option_index**)*
*gsi_return_attrs(registered_item, attributes, **count**, context)*
*gsi_return_attrs(registered_item, attributes, count, **context**)*
*gsi_return_message(message, **context**)*
gsi_return_timed_attrs
*   (registered_item, attributes, **count**, **context**)*
*gsi_return_timed_values(registered_items, **count**, **context**)*

gsi_int *(continued)*

gsi_return_values(registered_items, **count**, **context***)*
gsi_rpc_call
(function_handle, arguments, [call_identifier], **context***)*
gsi_rpc_call_with_count
(function_handle, arguments, **count**, *[call_identifier],*
context*)*
gsi_rpc_declare_remote
(function_handle, g2_function_name, receiver_function,
[procedure_user_data], **argument_count**, **return_count**, **context***)*
gsi_rpc_declare_remote_with_error_handler_and_user_data
(function_handle, g2_function_name, receiver_function,
error_receiver_function, [procedure_user_data],
argument_count, **return_count**, **context***)*
gsi_rpc_return_error_values
(arguments, **count**, *call_identifier,* **context***)*
gsi_rpc_return_values
(arguments, **count**, *call_identifier,* **context***)*
gsi_rpc_start(function_handle, arguments, **context***)*
gsi_rpc_start_with_count
(function_handle, arguments, **count**, **context***)*
gsi_set_attr_count(item_or_regitem_or_attr_or_reg, **count***)*
gsi_set_attr_is_transient(attribute, **new_value***)*
gsi_set_attrs(item_or_attr, new_attributes, **count***)*
*gsi_set_context_limit(***limit***)*
*gsi_set_context_user_data(***context**, *context_user_data)*
gsi_set_element_count(item, **count***)*
gsi_set_elements
(item_or_attr, elements_array, **count**, **type_tag***)*
gsi_set_flt_list(item_or_attr, doubles_array, **count***)*
gsi_set_handle(item_or_regitem, **handle_value***)*
gsi_set_history
(item_or_attr, values, timestamps, **count**, **type**,
maximum_count, **maximum_age**, **min_interval***)*
*gsi_set_include_file_version(***major**, **minor***)*
gsi_set_int(item_or_regitem_or_attr, **integer_value***)*
gsi_set_int_array(item_or_attr, integers_array, **count***)*
gsi_set_int_list(item_or_attr, integers_array, **count***)*
gsi_set_interval(registered_item, **interval***)*
gsi_set_item_append_flag(item, **append_flag***)*
gsi_set_item_reference_flag(item, **reference_flag***)*

Functions Grouped by Argument Type

gsi_int *(continued)*

gsi_set_log(item_or_regitem_or_attr, **truth_value***)*
*gsi_set_option(***option_index***)*
*gsi_set_pause_timeout(***max_idle_time***)*
gsi_set_rpc_remote_return_value_kind
(function_handle, **return_value_index***, kind)*
*gsi_set_run_loop_timeout(***max_run_time***)*
gsi_set_status(registered_item, **status***)*
*gsi_set_string_conversion_style(***style***)*
gsi_set_type(item_or_regitem_or_attr, **gsi_type***)*
gsi_set_update_items_in_lists_and_arrays_flag
(item, **immediate_flag***)*
gsi_set_usv(item_or_regitem_or_attr, us_vector, **length***)*
gsi_signal_error
(originating_function_name, **user_error***, message)*
gsi_watchdog(user_watchdog_function, **timeout_interval***)*
*{gsi_close_fd}(***fd***)*
*{gsi_error_handler}(***error_context***,* **error_code***, error_message)*
{gsi_get_data}(registered_items, **count***)*
{gsi_initialize_context}
(remote_process_initialization_string, **length***)*
*{gsi_open_fd}(***fd***)*
*{gsi_read_callback}(***context***,* **state***)*
{gsi_receive_deregistrations}(registered_items, **count***)*
{gsi_receive_message}(message, **length***)*
{gsi_rpc_local_fn}
([procedure_user_data], rpc_arguments, **count***,*
call_identifier)
{gsi_rpc_receiver_fn}
*([procedure_user_data], rpc_arguments, <***count***>,*
[call_identifier])
*{gsi_run_state_change}(***direction_code***,* **type_code***, char * name)*
{gsi_set_data}(registered_items, **count***)*
*{gsi_write_callback}(***context***,* **state***)*

Functions Grouped by Argument Type

gsi_item *

*gsi_reclaim_array(***array***)*
*gsi_reclaim_items(***items***)*
gsi_rpc_call
(function_handle, **arguments***, [call_identifier], context)*
gsi_rpc_call_with_count(function_handle, **arguments***, count,*
[call_identifier], context)
*gsi_rpc_return_error_values(***arguments***, count, call_identifier,*
context)
*gsi_rpc_return_values(***arguments***, count, call_identifier,*
context)
gsi_rpc_start(function_handle, **arguments***, context)*
gsi_rpc_start_with_count(function_handle, **arguments***, count,*
context)
gsi_set_elements(item_or_attr, **elements_array***, count, type_tag)*
{gsi_rpc_local_fn}
([procedure_user_data], **rpc_arguments***, count,*
call_identifier)
{gsi_rpc_receiver_fn}
([procedure_user_data], **rpc_arguments***, count,*
[call_identifier])
gsi_set_rpc_remote_return_exclude_user_attrs
(function_handle, **attributes***)*
gsi_set_rpc_remote_return_include_system_attrs
(function_handle, **attributes***)*
gsi_set_rpc_remote_return_include_all_system_attrs_except
(function_handle, **attributes***)*

Functions Grouped by Argument Type

gsi_item

gsi_extract_history
*(**item**, values_address, timestamps_address, type_address)*
gsi_extract_history_spec
*(**item**, maximum_count_address, maximum_age_address,*
minimum_interval_address)
*gsi_is_item(**item**)*
*gsi_item_append_flag(**item**)*
*gsi_item_reference_flag(**item**)*
*gsi_reclaim_item(**item**)*
*gsi_set_attr_by_name(item_or_attr, search_name, **source_item**)*
*gsi_set_element_count(**item**, count)*
*gsi_set_item_append_flag(**item**, append_flag)*
*gsi_set_item_of_attr(attribute, **source_item**)*
gsi_set_item_of_attr_by_name
*(item_or_attr, search_name, **source_item**)*
*gsi_set_item_reference_flag(**item**, reference_flag)*
*gsi_set_name(**item**, name)*
gsi_set_update_items_in_lists_and_arrays_flag
*(**item**, immediate_flag)*
*gsi_simple_content_copy(**destination**, **source**)*
*gsi_update_items_in_lists_and_arrays_flag(**item**)*

Functions Grouped by Argument Type

gsi_item or gsi_attr

gsi_attr_count_of(**item_or_attr**)
gsi_attrs_of(**item_or_attr**)
gsi_class_name_of(**item_or_attr**)
gsi_class_type_of(**item_or_attr**)
gsi_elements_of(**item_or_attr**)
gsi_history_count_of(**item_or_attr**)
gsi_history_type_of(**item_or_attr**)
gsi_timestamp_of(**item_or_attr**)
gsi_set_flt_list(**item_or_attr**, *doubles_array, count*)
gsi_set_flt_array(**item_or_attr**, *doubles_array, gsi_int count*)
gsi_set_elements(**item_or_attr**, *elements_array, count, type_tag*)
gsi_set_int_array(**item_or_attr**, *integers_array, count*)
gsi_set_int_list(**item_or_attr**, *integers_array, count*)
gsi_set_class_name(**item_or_attr**, *name*)
gsi_set_attrs(**item_or_attr**, *new_attributes, count*)
gsi_attr_by_name(**item_or_attr**, *search_name*)
gsi_attr_by_name(**item_or_attr**, *search_name*)
gsi_item_of_attr_by_name(**item_or_attr**, *search_name*)
gsi_set_attr_by_name(**item_or_attr**, *search_name, source_attr*)
gsi_set_attr_by_name(**item_or_attr**, *search_name, source_item*)
gsi_set_item_of_attr_by_name
(**item_or_attr**, *search_name, source_item*)
gsi_set_sym_array
(**item_or_attr**, *symbol_values_array, gsi_int count*)
gsi_set_sym_list
(**item_or_attr**, *symbol_values_array, gsi_int count*)
gsi_set_str_array
(**item_or_attr**, *text_values_array, gsi_int count*)
gsi_set_str_list
(**item_or_attr**, *text_values_array, gsi_int count*)
gsi_set_timestamp(**item_or_attr**, *timestamp*)
gsi_set_log_array
(**item_or_attr**, *truth_values_array, gsi_int count*)
gsi_set_log_list
(**item_or_attr**, *truth_values_array, gsi_int count*)
gsi_set_history
(**item_or_attr**, *values, timestamps, count, type,*
maximum_count, maximum_age, min_interval)

gsi_item or gsi_registered_item

gsi_set_handle(**item_or_regitem**, *handle_value*)

Functions Grouped by Argument Type

gsi_item, gsi_registered_item, or gsi_attr

*gsi_clear_item(***item_or_regitem_or_attr***)*
*gsi_element_count_of(***item_or_regitem_or_attr***)*
*gsi_flt_array_of(***item_or_regitem_or_attr***)*
*gsi_flt_list_of(***item_or_regitem_or_attr***)*
*gsi_flt_of(***item_or_regitem_or_attr***)*
*gsi_int_array_of(***item_or_regitem_or_attr***)*
*gsi_int_list_of(***item_or_regitem_or_attr***)*
*gsi_int_of(***item_or_regitem_or_attr***)*
*gsi_log_array_of(***item_or_regitem_or_attr***)*
*gsi_log_list_of(***item_or_regitem_or_attr***)*
*gsi_log_of(***item_or_regitem_or_attr***)*
*gsi_str_array_of(***item_or_regitem_or_attr***)*
*gsi_str_list_of(***item_or_regitem_or_attr***)*
*gsi_str_of(***item_or_regitem_or_attr***)*
*gsi_sym_array_of(***item_or_regitem_or_attr***)*
*gsi_sym_list_of(***item_or_regitem_or_attr***)*
*gsi_sym_of(***item_or_regitem_or_attr***)*
*gsi_usv_length_of(***item_or_regitem_or_attr***)*
*gsi_usv_of(***item_or_regitem_or_attr***)*
*gsi_set_flt(***item_or_regitem_or_attr***, float_value)*
*gsi_set_type(***item_or_regitem_or_attr***, gsi_type)*
*gsi_set_int(***item_or_regitem_or_attr***, integer_value)*
*gsi_set_sym(***item_or_regitem_or_attr***, symbol_value)*
*gsi_set_str(***item_or_regitem_or_attr***, text_value)*
*gsi_set_log(***item_or_regitem_or_attr***, truth_value)*
*gsi_set_usv(***item_or_regitem_or_attr***, us_vector, length)*

gsi_item, gsi_registered_item, gsi_attr, or gsi_registration

*gsi_name_of(***item_or_regitem_or_attr_or_reg***)*
*gsi_owner_of(***item_or_regitem_or_attr_or_reg***)*
*gsi_type_of(***item_or_regitem_or_attr_or_reg***)*
*gsi_set_attr_count(***item_or_regitem_or_attr_or_reg***, count)*

gsi_item, gsi_registered_item, or gsi_registration

*gsi_handle_of(***item_or_regitem_or_reg***)*
*gsi_registration_of_item(***item_or_regitem_or_reg***)*

gsi_item_user_data_type

gsi_set_user_data(registration, **item_user_data***)*

Functions Grouped by Argument Type

gsi_procedure_user_data_type

gsi_rpc_declare_local
*(local_function, [**procedure_user_data**], g2_function_name)*
gsi_rpc_declare_remote
(function_handle, g2_function_name, receiver_function,
*[**procedure_user_data**], argument_count, return_count, context)*
gsi_rpc_declare_remote_with_error_handler_and_user_data
(function_handle, g2_function_name, receiver_function,
*error_receiver_function, [**procedure_user_data**],*
argument_count, return_count, context)
{gsi_rpc_local_fn}
*([**procedure_user_data**], rpc_arguments, count,*
call_identifier)
{gsi_rpc_receiver_fn}
*([**procedure_user_data**], rpc_arguments, count,*
[call_identifier])

gsi_registered_item *

*gsi_reclaim_registered_items(**registered_items**)*
*gsi_return_timed_values(**registered_items**, count, context)*
*gsi_return_values(**registered_items**, count, context)*

gsi_registered_item

*gsi_item_of_registered_item(**registered_item**)*
*gsi_return_attrs(**registered_item**, attributes, count, context)*
gsi_return_timed_attrs
*(**registered_item**, attributes, count, context)*
*gsi_set_interval(**registered_item**, interval)*
*gsi_set_status(**registered_item**, status)*
*gsi_status_of(**registered_item**)*

gsi_registered_item_array

*{gsi_get_data}(**registered_items**, count)*
*{gsi_receive_deregistrations}(**registered_items**, count)*
*{gsi_set_data}(**registered_items**, count)*

gsi_registered_item or gsi_registration

*gsi_interval_of(**regitem_or_reg**)*

Functions Grouped by Argument Type

gsi_registration

gsi_identifying_attr_of
*(***registration***, identifying_attribute_index)*
gsi_item_of_identifying_attr_of
(registration, identifying_attribute_index)
*gsi_set_user_data(***registration***, item_user_data)*
*gsi_user_data_of(***registration***)*
*{gsi_receive_registration}(***registration***)*

gsi_rpc_local_fn_type *

gsi_rpc_declare_local
*(***local_function***, [procedure_user_data], g2_function_name)*

gsi_rpc_receiver_fn_type *

gsi_rpc_declare_remote
(function_handle, g2_function_name, ***receiver_function****,*
[procedure_user_data], argument_count, return_count, context)
gsi_rpc_declare_remote_with_error_handler_and_user_data
(function_handle, g2_function_name, ***receiver_function****,*
error_receiver_function, [procedure_user_data],
argument_count, return_count, context)
gsi_rpc_declare_remote_with_error_handler_and_user_data
(function_handle, g2_function_name, receiver_function,
error_receiver_function*, [procedure_user_data],*
argument_count, return_count, context)

gsi_symbol *

gsi_set_sym_array
(item_or_attr, ***symbol_values_array****, gsi_int count)*
gsi_set_sym_list
(item_or_attr, ***symbol_values_array****, gsi_int count)*

Functions Grouped by Argument Type

gsi_symbol

gsi_attr_by_name(item_or_attr, **search_name***)*
gsi_item_of_attr_by_name(item_or_attr, **search_name***)*
gsi_set_attr_by_name(item_or_attr, **search_name***, source_item)*
gsi_set_attr_name(attribute, **attribute_name***)*
gsi_set_class_name(item_or_attr, **name***)*
gsi_set_class_qualifier(attribute, **attribute_name***)*
gsi_set_item_of_attr_by_name
(item_or_attr, **search_name***, source_item)*
gsi_set_name(item, **name***)*
gsi_set_sym(item_or_regitem_or_attr, **symbol_value***)*
*gsi_set_symbol_user_data(***sym***, symbol_user_data)*
gsi_set_unqualified_attr_name(attribute, **attribute_name***)*
*gsi_symbol_name(***sym***)*
*gsi_symbol_user_data(***sym***)*

gsi_symbol_user_data_type

gsi_set_symbol_user_data(sym, **symbol_user_data***)*

gsi_version_id

gsi_version_information(gsi_version_id)

gsi_watchdog_function_type *

*gsi_watchdog(***gsi_watchdog_function***, timeout_interval)*

int

*gsi_signal_handler(***signal_code***)*
*gsi_start(***argc***, argv)*

long

*gsi_unwatch_fd(***file_descriptor***)*
*gsi_watch_fd(***file_descriptor***)*

short *

*gsi_convert_unicode_to_string(***string***, style)*
*gsi_convert_unicode_to_wide_string(***string***, style)*
*gsi_convert_wide_string_to_unicode(***string***, style)*

unsigned short *

gsi_set_usv(item_or_regitem_or_attr, **us_vector***, length)*

Functions Grouped by Argument Type

void * *gsi_set_history* *(item_or_attr,* **values**, *timestamps, count, type,* *maximum_count, maximum_age, min_interval)*
void ** *gsi_extract_history* *(item,* **values_address, *timestamps_address, type_address)*

Functions by Type of Return Value

The following table lists API and callback functions grouped by the types of their return values.

Note The curly braces ({}) around a function name indicate that this function is a callback function defined by the user. The brackets ([]) around an argument indicate that this argument is supplied if the C preprocessor macro *GSI_USE_USER_DATA_FOR_CALLBACKS* is defined and is not supplied otherwise.

Functions Grouped by Return Type

char *   *convert_unicode_to_string()*
double *   *gsi_flt_array_of()*
double   *gsi_encode_timestamp()* *gsi_flt_of()* *gsi_interval_of()* *gsi_timestamp_of()*
gsi_attr *   *gsi_attrs_of()* *gsi_make_attrs()* *gsi_make_attrs_with_items()*
gsi_attr   *gsi_attr_by_name()*

Functions Grouped by Return Type

gsi_call_identifier_type *gsi_last_error_call_handle()*
gsi_char ** *gsi_str_array_of()* *gsi_str_list_of()*
gsi_char * *gsi_error_message()* *gsi_last_error_message()* *gsi_str_of()* *gsi_symbol_name()*
gsi_context_user_data_type *gsi_context_user_data()*
gsi_int * *gsi_int_array_of()* *gsi_int_list_of()* *gsi_log_array_of()* *gsi_log_list_of()*

Functions Grouped by Return Type

gsi_int

{gsi_get_tcp_port}()
{gsi_initialize_context}()

gsi_attr_count_of()
gsi_attr_is_transient()
gsi_attr_name_is_qualified()
gsi_class_type_of()
gsi_context_received_data()
gsi_context_socket()
gsi_current_context()
gsi_element_count_of()
gsi_establish_listener()
gsi_extract_history()
gsi_extract_history_spec()
gsi_handle_of()
gsi_history_count_of()
gsi_history_type_of()
gsi_initiate_connection()
gsi_initiate_connection_with_user_data()
gsi_int_of()
gsi_is_item()
gsi_item_append_flag()
gsi_item_reference_flag()
gsi_last_error()
gsi_listener_socket()
gsi_log_of()
gsi_option_is_set()
gsi_owner_of()
gsi_status_of()
gsi_string_conversion_style()
gsi_type_of()
gsi_update_items_in_lists_and_arrays_flag()
gsi_usv_length_of()
gsi_version_information()
gsi_wakeup()

gsi_item *

gsi_elements_of()
gsi_make_array()
gsi_make_items()

Functions Grouped by Return Type

gsi_item

gsi_attr_by_name()
gsi_identifying_attr_of()
gsi_item_of_attr()
gsi_item_of_attr_by_name()
gsi_item_of_identifying_attr_of()
gsi_item_of_registered_item()
gsi_make_item()

gsi_item_user_data_type

gsi_user_data_of()

gsi_registered_item *

gsi_make_registered_items()

gsi_registration

gsi_registration_of()
gsi_registration_of_handle()
gsi_registration_of_item()

gsi_symbol *

gsi_sym_array_of()
gsi_sym_list_of()

gsi_symbol

gsi_attr_name_of()
gsi_class_name_of()
gsi_class_qualifier_of()
gsi_make_symbol()
gsi_name_of()
gsi_sym_of()
gsi_unqualified_attr_name_of()

gsi_symbol_user_data_type

gsi_symbol_user_data()

short *

gsi_convert_string_to_unicode()
gsi_convert_unicode_to_wide_string()
gsi_convert_wide_string_to_unicode()

Functions Grouped by Return Type

unsigned short *

gsi_usv_of()

void

gsi_clear_item()
gsi_clear_last_error()
gsi_close_listeners()
gsi_decode_timestamp()
gsi_flush()
gsi_initialize_callbacks()
gsi_initialize_error_variable()
gsi_initialize_for_win32()
gsi_install_error_handler()
gsi_install_missing_procedure_handler()
gsi_install_run_state_change()
gsi_kill_context()
gsi_pause()
gsi_reclaim_array()
gsi_reclaim_attrs()
gsi_reclaim_attrs_with_items()
gsi_reclaim_item()
gsi_reclaim_items()
gsi_reclaim_registered_items()
gsi_reset_option()
gsi_return_attrs()
gsi_return_message()
gsi_return_timed_attrs()
gsi_return_timed_values()
gsi_return_values()
gsi_rpc_call()
gsi_rpc_call_with_count()
gsi_rpc_declare_local()
gsi_rpc_declare_remote()
gsi_rpc_declare_remote_with_error_handler_and_user_data()
gsi_rpc_return_error_values()
gsi_rpc_return_values()
gsi_rpc_start()
gsi_rpc_start_with_count()
gsi_run_loop()
gsi_set_attr_by_name()

void *(continued)*

gsi_set_attr_by_name()
gsi_set_attr_count()
gsi_set_attr_is_transient()
gsi_set_attr_name()
gsi_set_attrs()
gsi_set_class_name()
gsi_set_class_qualifier()
gsi_set_context_limit()
gsi_set_context_user_data()
gsi_set_element_count()
gsi_set_elements()
gsi_set_flt()
gsi_set_flt_array()
gsi_set_flt_list()
gsi_set_handle()
gsi_set_history()
gsi_set_include_file_version()
gsi_set_int()
gsi_set_int_array()
gsi_set_int_list()
gsi_set_interval()
gsi_set_item_append_flag()
gsi_set_item_of_attr()
gsi_set_item_of_attr_by_name()
gsi_set_item_reference_flag()
gsi_set_log()
gsi_set_log_array()
gsi_set_log_list()
gsi_set_name()
gsi_set_option()
gsi_set_pause_timeout()
gsi_set_rpc_remote_return_value_kind()
gsi_set_rpc_remote_return_exclude_user_attrs ()
gsi_set_rpc_remote_return_include_system_attrs()
gsi_set_rpc_remote_return_include_all_system_attrs_except()
gsi_set_run_loop_timeout()
gsi_set_status()
gsi_set_str()
gsi_set_str_array()
gsi_set_str_list()
gsi_set_string_conversion_style()
gsi_set_sym()
gsi_set_sym_array()
gsi_set_sym_list()

Functions Grouped by Return Type

void *(continued)*

gsi_set_symbol_user_data()
gsi_set_timestamp()
gsi_set_type()
gsi_set_unqualified_attr_name()
gsi_set_update_items_in_lists_and_arrays_flag()
gsi_set_user_data()
gsi_set_usv()
gsi_signal_error()
gsi_signal_handler()
gsi_simple_content_copy()
gsi_start()
gsi_unwatch_fd()
gsi_watch_fd()
gsi_watchdog()
{gsi_close_fd}()
{gsi_error_handler}()
{gsi_g2_poll}()
{gsi_get_data}()
{gsi_missing_procedure_handler}()
{gsi_open_fd}()
{gsi_pause_context}()
{gsi_read_callback}()
{gsi_receive_deregistrations}()
{gsi_receive_message}()
{gsi_receive_registration}()
{gsi_resume_context}()
{gsi_rpc_local_fn}()
{gsi_rpc_receiver_fn}()
{gsi_run_state_change}()
{gsi_set_data}()
{gsi_set_up}()
{gsi_shutdown_context}()
{gsi_write_callback}()
{gsi_watchdog_function}()

Functions with No Arguments

The following table lists API and callback functions that have no arguments:

API and Callback Functions with No Arguments

gsi_clear_last_error()
gsi_close_listeners()
gsi_current_context()
gsi_last_error()
gsi_last_error_call_handle()
gsi_last_error_message()
gsi_listener_socket()
gsi_make_item()
gsi_pause()
gsi_run_loop()
gsi_string_conversion_style()
gsi_wakeup()
{gsi_g2_poll}()
{gsi_get_tcp_port}()
{gsi_pause_context}()
{gsi_resume_context}()
{gsi_set_up}()
{gsi_shutdown_context}()
{gsi_watchdog_function}()

Constants

Lists symbolic constants defined in G2 Gateway header files.

Introduction

The following table lists the constants provided by Gensym for use in your G2 Gateway application. These constants are defined in the header file `gsi_main.h`. Files that use the API functions of G2 Gateway must include the `gsi_main.h` file.

G2 Gateway Constants

Constant	Value
NO_ERR	0
NULL_PTR	0
TRUE	1
FALSE	0
GSI_ACCEPT	0
GSI_REJECT	1
GSI_UNDEFINED_CONTEXT	-1
GSI_VOID_INDEX	-1
GSI_NULL_TAG	0
GSI_INTEGER_TAG	1

G2 Gateway Constants

Constant	Value
GSI_IO_UNBLOCKED	1
GSI_IO_BLOCKED	0
GSI_CALL_HANDLE_OF_START	0
GSI_SYMBOL_TAG	3
GSI_STRING_TAG	4
GSI_LOGICAL_TAG	5
GSI_FLOAT64_TAG	6
GSI_ITEM_TAG	7
GSI_VALUE_TAG	8
GSI_HANDLE_TAG	9
GSI_QUANTITY_TAG	10
GSI_INTEGER_ARRAY_TAG	17
GSI_SYMBOL_ARRAY_TAG	19
GSI_STRING_ARRAY_TAG	20
GSI_LOGICAL_ARRAY_TAG	21
GSI_FLOAT64_ARRAY_TAG	22
GSI_ITEM_ARRAY_TAG	23
GSI_VALUE_ARRAY_TAG	24
GSI_ITEM_OR_VALUE _ARRAY_TAG	25
GSI_QUANTITY_ARRAY_TAG	26
GSI_INTEGER_LIST_TAG	33
GSI_SYMBOL_LIST_TAG	35
GSI_STRING_LIST_TAG	36
GSI_LOGICAL_LIST_TAG	37

G2 Gateway Constants

Constant	Value
GSI_FLOAT64_LIST_TAG	38
GSI_ITEM_LIST_TAG	39
GSI_VALUE_LIST_TAG	40
GSI_ITEM_OR_VALUE_LIST_TAG	41
GSI_QUANTITY_LIST_TAG	42
GSI_SEQUENCE_TAG	49
GSI_STRUCTURE_TAG	50
GSI_PORT_NUM	22041
GSI_TRUE	1000
GSI_FALSE	–1000
MAX_G2_INTEGER	536870911
MIN_G2_INTEGER	–536870912

G2 Gateway Error Messages

Lists and describes the standard error messages returned by G2 Gateway.

Introduction

The following table lists the standard severe G2 Gateway errors:

Errors

Value	Error Symbol	Text of Error
1	*GSI_INVALID_PROTOCOL*	"*Invalid network protocol* ***protocol-name*** *- please specify tcpip*"
2	*GSI_INVALID_NETWORK_COMBO*	"*Duplication of network protocol* ***protocol-name*** *- please specify one of each*"
3	*GSI_INVALID_TCP_PORT_NUMBER*	"*Invalid TCP port number* ***port-number*** *- please specify an integer number*"
5	*GSI_INVALID_NETWORK_ADDRESS*	"*Invalid network address* ***network-address*** *- does not conform to tcpip*"
6	*GSI_ITEM_DEFINITIONS_ARE_READ_ONLY*	"*Item registrations are read-only - make modifications through G2*"

Errors

Value	Error Symbol	Text of Error
7	*GSI_STRUCTURE_HAS_NO_HANDLE*	"*Access function received structure with no handle element*"
8	*GSI_STRUCTURE_HAS_NO_IDENTIFYING_ATTRS*	"*Access function received structure with no identifying attribute elements*"
9	*GSI_STRUCTURE_HAS_NO_INSTANCE*	"*Access function received an incompatible structure as an argument*"
10	*GSI_STRUCTURE_HAS_NO_TIMESTAMP*	"*Access function received structure with no timestamp element*"
11	*GSI_STRUCTURE_HAS_NO_TIMED_FLAG*	"*Access function received structure with no is-timed element*"
12	*GSI_STRUCTURE_HAS_NO_INTERVAL*	"*Access function received structure with no interval element*"
13	*GSI_STRUCTURE_HAS_NO_STATUS*	*Access function received structure with no status element*"
14	*GSI_STRUCTURE_HAS_NO_ATTRIBUTE_NAME*	"*Access function expected an attribute structure*"
15	*GSI_MISSING_INSTANCE_STRUCTURE*	"*Nothing was found where an embedded value or instance structure was expected*"
16	*GSI_INCOMPATIBLE_TYPE*	"*Type mismatch - value of type* ***data-type*** *passed to this function*"
17	*GSI_LOCAL_FUNCTION_UNDEFINED*	"*GSI could not find function* ***function-name*** *which G2 attempted to invoke*"
18	*GSI_INVALID_CALL_HANDLE*	"*Call handle specified was not for an outstanding RPC*"

Errors

Value	Error Symbol	Text of Error
19	*GSI_REMOTE_DECLARATION_NOT_FOUND*	"*A remote procedure declaration could not be found. Ensure gsi_rpc_declare_remote() called*"
20	*GSI_INCORRECT_ARGUMENT_COUNT*	"*Remote procedure* ***function-name*** *passed/received* ***number-of-args*** *arguments - number-of-args were expected*"
21	*GSI_UNEXPECTED_OPERATION*	"*This operation not thought to be useful and may imply a misunderstanding of GSI*"
22	*GSI_INTERNAL_MEMORY_INCONSISTENCY*	"*Internal memory inconsistency encountered in GSI - please contact customer support*"
23	*GSI_BAD_MAGIC_NUMBER*	"*Bad magic number - address received was not the head of an array*"
24	*GSI_INVALID_CONTEXT*	"*Context* ***context-number*** *is invalid or inactive*"
25	*GSI_IDENTIFYING_ATTRIBUTE_INDEX_OUT_OF_RANGE*	"*Identifying attribute index* ***attribute-index*** *is out of the acceptable range (1..**limit**)*"
26	*GSI_NO_CLASS_IN_REGISTERED_ITEM*	"*The class name is in the registration corresponding to the handle of this registered item*"
27	*GSI_G2_LOGICAL_OUT_OF_BOUNDS*	"*Logical* ***logical-name*** *is out of bounds. Logicals in GSI may range from -1000 to 1000 inclusive*"
28	*GSI_MISSING_ATTRIBUTE_NAME*	"*Requested attribute name where there was none*"
29	*GSI_COUNT_OUT_OF_ARRAY_BOUNDS*	"*Count* ***element-count*** *out of bounds for array of size* ***array-size***"

Errors

Value	Error Symbol	Text of Error
30	*GSI_INCOMPATIBLE_STRUCTURE*	"*Received null pointer argument, or a structure type incompatible with requested operation*"
31	*GSI_STRUCTURE_HAS_NO_CLASS_QUALIFIER*	"*Requested class qualifier where there was none*"
32	*GSI_NO_CLASS_NAME_SPECIFIED*	"*Attempt to pass item without specifying class name*"
33	*GSI_ARGUMENT_COUNT_TOO_BIG*	"*Argument count* ***argument-count*** *too large for provided argument array of length* ***array-size***"
34	*GSI_RPC_ERROR_FROM_G2*	"*Remote procedure invocation failed with level* ***severity*** *error:* ***text***"
35	*GSI_RPC_ERROR_FROM_G2_PROCEDURE*	"*Invocation of remote procedure* ***function-name*** *failed with level* ***severity*** *error:****text***"
36	*GSI_RESERVED_ERROR_CODE*	"*Reserved error code* ***error-code*** *signalled by user. Codes below 1024 are reserved*"
37	*GSI_CUSTOM_USER_ERROR*	"*The following custom error has been signalled:* ***error-code***"
38	*GSI_CONNECTION_LOST*	"*Network layer reports connection was lost or ICP protocol error occurred:* ***error-message***"
39	*GSI_OWNEROUS_RECLAMATION_VIOLATION*	"*Attempted to reclaim a structure not owned (allocated) by user*"
40	*GSI_NULL_POINTER_ARGUMENT*	"*Access function received null pointer argument*"

Errors

Value	Error Symbol	Text of Error
41	*GSI_ITEM_HAS_NO_VALUE*	"*Attempted to access value when none present - see GSI_NULL_TAG*"
42	*GSI_NOOP_CODE_ACCESSED*	"*Support for desired network transport type (TCP) not linked in to your executable*"
43	*GSI_ILLEGAL_NESTING_OF_RUN_LOOP*	"*Attempt to call gsi_run_loop from within the extent of a previous call to gsi_run_loop*"
44	*GSI_STRUCTURE_HAS_NO_HISTORY*	"*Attempted to access history when none present - see gsi_history_count_of()*"
45	*GSI_INCOMPATIBLE_HISTORY_SPEC*	"*Type code* ***type-code*** *is invalid - see documentation for valid type codes*"
46	*GSI_INVALID_TYPE*	"*Type code* ***type-code*** *is invalid for return to data served variable*"
47	*GSI_TYPE_FOR_DATA_SERVICE*	"*Attempt to load* ***number-of-values*** *values into a history whose spec allows for no more than* ***maximum-values***"
48	*GSI_BAD_MAGIC_NUMBER_IN_QUERY*	"*Attempted to access non-existent data - perhaps reclaimed or never allocated?*"
49	*GSI_CONTEXT_UNDEFINED*	"*Context -1 is the undefined context - operation attempted outside of any context*"
50	*GSI_INVALID_HANDLE*	"*The handle of the registered_item is invalid*"
51	*GSI_INVALID_HANDLE_OF_NTH*	"*The handle of registered_items[****array-index****] is invalid*"

Errors

Value	Error Symbol	Text of Error
52	*GSI_TIMESTAMP_OUT_OF_BOUNDS*	"*Out-of-range timestamp seen: timestamp: timestamp-value Valid range: 0.0 < t < 2145934799*" (*2145934799* means midnight, December 31, 2037)
53	*GSI_INVALID_TIMESTAMP*	"*Invalid timestamp seen in timestamped object.*"
54	*GSI_INVALID_HANDLE_FOR_CONTEXT*	"*The handle handle is not known in context context-number*"
55	*GSI_NO_NAME_IN_REGISTERED_ITEM*	"*The name is in the registration corresponding to the handle of this registered item*"
56	*GSI_INCOMPATIBLE_STRUCTURE2*	"*Received null pointer argument, or incompatible type for second structure argument*"
57	*GSI_ICP_MESSAGE_TOO_LONG*	"*ICP message series too long -- please call Gensym customer support*"
58	*GSI_ICP_MESSAGE_OUT_OF_SYNCH_CASE_2*	"*Protocol out-of-synch (case 2)*"
59	*GSI_MAXIMUM_TEXT_STRING_LENGTH_ERROR*	"*Attempting to allocate number-of-elements element string, which is beyond the established maximum of maximum-elements.*"
60	*GSI_EXTEND_CURRENT_TEXT_STRING_ERROR*	"*Trying to write a string longer than 1MB!*"
61	*GSI_UNDEFINED_STRUCTURE_METHOD_ERROR*	"*Call to structure method were none was defined.*"
62	*GSI_INDEX_SPACE_FULL_ERROR*	"*Index space full.*"

Errors

Value	Error Symbol	Text of Error
63	*GSI_CIRCULARITY_NOT_SUPPORTED*	"*Self referencing items may not yet be sent to G2 - sorry*" **Note**: This error occurs only when G2 Gateway 5.0 is communicating with a G2 4.0.
64	*GSI_NO_MORE_RPC_IDENTIFIERS*	"*The limit of outstanding calls to G2 procedures has been reached*"
65	*GSI_BAD_RPC_STATE*	"*Internal state of RPC identifier is unknown.*"
66	*GSI_MALFORMED_SOCKET*	"*GSI socket malformed-extension data missing*" *Causes the API function to exit.*
67	*GSI_UNKNOWN_TYPE_TAG*	"*GSI structure contains unknown type tag.*"
68	*GSI_INVALID_HISTORY_TYPE*	"*GSI found an invalid value type for this history.*"
69	*GSI_UNCAUGHT_EXCEPTION*	*Not used.*
70	*GSI_CANNOT_ESTABLISH_LISTENER*	"*Could not make exact ICP connection.*"
71	*GSI_MAXIMUM_CONTEXTS_EXCEEDED*	"*Connection rejected - GSI bridge context limit* ***maximum-contexts*** *exceeded.*" Closes the connection.
72	*GSI_CONNECTION_DENIED*	"*Connection denied - the G2 at* ***protocol-host-port*** *has disallowed connections from GSI*" Closes the connection.

Errors

Value	Error Symbol	Text of Error
73	*GSI_UNKNOWN_STRING_CONVERSION_STYLE*	"*Unknown string conversion style*"
74	*GSI_ERROR_IN_CONNECT*	"*Error during connection attempt: error-message*" Closes the connection.
75	*GSI_INVALID_RETURN_VALUE_INDEX*	"*Invalid return value index*"
76	*GSI_INVALID_RETURN_VALUE_KIND*	"*Invalid return value kind*"
77	*GSI_INVALID_ATTRIBUTE_TYPE*	"*Invalid attribute type*"
78	*GSI_RESUMABLE_ICP_ERROR*	Not used.
79	*GSI_UNKNOWN_CALLING_PROCEDURE_INDEX*	"*Unknown calling procedure index.*" Closes the connection.
80	*GSI_ILLEGAL_ERROR_ARGUMENT*	"*Illegal error arguments: error-description .*" Closes the connection.
81	*GSI_INTEGER_TOO_LARGE*	"*gsi_int argument was larger than 536870911.*" The number 536870911 is $2^{29} - 1$.
82	*GSI_INTEGER_TOO_SMALL*	"*gsi_int argument was smaller than −536870912.*" The number –536870912 is -2^{29}.
83	*GSI_NOT_A_SYMBOL*	"*Invalid gsi_symbol argument (use gsi_make_symbol).*"
84	*GSI_NO_LICENSE*	*No license is available for a connection over connection*

Errors

Value	Error Symbol	Text of Error
85	*GSI_CONTEXT_HAS_BEEN_KILLED*	*Context connection has been killed*
86	*GSI_CONTEXT_HAS_BEEN_SHUT_DOWN*	*Context connection has been shut down*
87	*GSI_CALLBACK_NAME_NOT_FOUND*	*Invalid GSI callback name: callback*
88	*GSI_INVALID_OPTION_INDEX*	*Invalid GSI option index: index*
89	*GSI_APPLICATION_LOAD_FAILED*	*Failed to successfully load the application application*
90	*GSI_NETWORK_ERROR*	*Network error event: socket-handle is socket, event-status is status*

Warnings

Value	Error Symbol	Text of Error
256	*GSI_ATTRIBUTE_NAME_NOT_FOUND*	"*Could not find attribute with qualified name of attribute-name*"
257	*GSI_TIMESTAMP_NOT_FOUND*	"*No timestamp specified. The 0.0 value returned does not construe a valid timestamp*"
258	*GSI_RPC_ASYNC_ABORT*	"*This call has been aborted by the client - no values returned*"

Warnings

Value	Error Symbol	Text of Error
259	*GSI_RPC_LATE_ASYNC_ABORT*	"*An RPC has been aborted by the client while return en route*"
1024	*GSI_BASE_USER_ERROR_CODE* User-defined error codes can use any value greater than 1024. Your user code can signal user-defined errors, using the API *gsi_signal_error()*.	

D

G2 Gateway Data Types

Describes the data types defined for use in G2 Gateway user code.

Introduction

This appendix describes the data types supported by G2 Gateway, the G2 data types and G2 Gateway type tags, and the G2 Gateway data types for RPC arguments.

Data Types Supported by G2 Gateway

Some types are more efficient to process than others. When you choose a data type to hold a value, select the type based on the purpose you have for the value. Each of the following data types is defined for you in G2 Gateway.

Floats

Floating point numbers require more processing than integers because of the manipulation involving the floating point. Use floats only if you need floating point accuracy.

Always use the *FLOAT64_TAG* data type, which is a standard IEEE 64-bit floating point number on UNIX platforms.

Integers

Integers are the most efficient data type. If you are returning whole numbers to G2, use integers instead of floats.

In your G2 Gateway user code, integers must be declared as type *gsi_int*.

Note Upon entering G2 Standard, integers have their two most significant bits removed. Upon entering G2 Standard, integers have their three most significant bits removed. If you require integers with full 64-bit range, use long integers.

Long integers

Long integers are efficient data type with native 64-bit computing capabilities. If you are returning signed 64-bit integers from/to G2, use long instead of integer or floats.

In your G2 Gateway user code, long integers must be declared as type *gsi_long*.

Null

Use the *Null* type when you want the value of a variable to be undefined.

Logicals

G2 uses fuzzy logicals that are integers ranging in value from -1000 (100% false) to 1000 (100% true). When a logical value is sent from G2 to G2 Gateway it is sent as a long integer. Since most external systems use standard 0/1 logic, values must be converted accordingly on their way into and out of G2.

Strings

Strings are null-terminated character strings that can include any characters when all are specified within double quotation marks (""). Use strings to represent data whose value changes frequently or whose value is likely to be used only a few times, as when you want to inform the operator with an error message concerning some condition in the external system. Strings are evaluated by G2 character by character, so long strings will take slightly longer to process than short ones.

Strings passed between G2 and G2 Gateway are limited in length only by the size of the G2 Gateway network buffers (64K bytes).

G2 Version 5.0 uses the Unicode character set internally for all strings. G2 Gateway can use a variety of different character sets, known as **string conversion styles**, to represent strings for all G2 Gateway API functions and callbacks. You can set the current string conversion style by calling the function *gsi_set_string_conversion_style()*. The function *gsi_string_conversion_style()* returns a value indicating which string conversion style is currently in use. You can apply string conversion styles to individual strings using the functions *convert_string_to_unicode()*, *convert_wide_string_to_unicode()*, *convert_unicode_to_string()*, and *convert_unicode_to_wide_string()*. For information about these functions, see API Functions.

Symbols

Symbols are null-terminated character strings that function similarly to enumerated types. Symbols are usually specified as upper-case alphanumeric characters, excluding spaces and other special characters, although other characters can be included through the use of escape characters. For information about the proper format of G2 symbols, see the *G2 Reference Manual*.

G2 allocates space for the symbols in an internal table for looking up as needed. After space is allocated for the symbol, G2 evaluates it by looking up its location in the table. This evaluation is faster than string evaluation, which is performed on a character-by-character basis. For this reason, it is good practice to use the symbol type for data values that are evaluated frequently, or for data values used in comparisons.

If you are using API functions that operate on symbols, it is good practice to compile your G2 Gateway application with the *GSI_USE_NEW_SYMBOL_API* preprocessor macro defined, or to set the runtime option *GSI_NEW_SYMBOL_API*. These options enable the API functions to access the symbols efficiently. When you use *GSI_USE_NEW_SYMBOL_API* or *GSI_NEW_SYMBOL_API*, a symbol value is equivalent to a *void **; otherwise, it is equivalent to a *gsi_char **.

To set the symbol value associated with a *gsi_item*, *gsi_registered_item*, or *gsi_attr* structure, use the function *gsi_set_sym()*. This function sets the type of the structure on which it operates to *_TAG*.

Symbol values returned by *gsi_sym_of()* are of the type *gsi_symbol*.

Symbols passed between G2 and G2 Gateway are limited in length only by the size of the G2 Gateway network buffers (64K bytes).

Note Symbol values that your G2 Gateway user code passes to G2 should be expressed in uppercase letters only. This enables G2 to interpret the symbol values correctly.

The following table summarizes the API functions that operate on symbols:

API Functions that Operate on Symbols

API Function	Description
gsi_attr_by_name	Invokes gsi_item_of_attr_by_name().
gsi_attr_name_of	Returns the name of an attribute.
gsi_class_name_of	Returns the name of the G2 class of an item.
gsi_class_qualifier_of	Returns the part of the name component of an attribute that is the class qualifier.
gsi_item_of_attr_by_name	Returns the gsi_item structure contained in a specified gsi_attr structure.
gsi_make_symbol	Returns a symbol, given the name of that symbol.
gsi_name_of	Returns a symbol representing the name of a specified GSI item.
gsi_set_attr_by_name	Invokes gsi_set_item_of_attr_by_name().
gsi_set_attr_name	Changes the name of an attribute.
gsi_set_class_name	Sets the value of the class name component of an item or an embedded item in an attribute.
gsi_set_class_qualifier	Changes the part of the name of an attribute that specifies the G2 class that defines the attribute.
gsi_set_item_of_attr_by_name	Sets the gsi_item embedded in a specified gsi_attr attribute structure.
gsi_set_long	Sets the name component of a specified gsi_item structure.

API Functions that Operate on Symbols

API Function	Description
gsi_set_sym	Sets the symbol value associated with a gsi_item, gsi_registered_item, or gsi_attr structure.
gsi_set_sym_array	Sets the values in the symbol array associated with a gsi_item, gsi_registered_item, or gsi_attr structure.
gsi_set_symbol_user_data	Sets the user data associated with a symbol.
gsi_set_sym_list	Sets the values in the symbol list associated with a gsi_item or gsi_attr structure.
gsi_set_unqualified_attr_name	Sets the unqualified part of an attribute's name.
gsi_sym_array_of	Returns the symbol array associated with a gsi_item, gsi_registered_item, or gsi_attr structure.
gsi_symbol_name	Returns the string the corresponds to a symbol. The string must not be modified.
gsi_symbol_user_data	Returns the user data associated with a symbol.
gsi_sym_list_of	Returns the symbol list associated with a gsi_item, gsi_registered_item, or gsi_attr structure.
gsi_sym_of	Returns the symbol associated with a gsi_item, gsi_registered_item, or gsi_attr structure.
gsi_unqualified_attr_name_of	Returns the unqualified part of an attribute's name.

Sequence and Structure Types

GSI_SEQUENCE_TAG

The G2 Gateway data structure that corresponds to a sequence in G2 is a *gsi_item* having the type tag *GSI_SEQUENCE_TAG*. The elements of the sequence can be accessed with the functions:

```
gsi_element_count_of()
gsi_elements_of()
gsi_set_elements()
```

GSI_STRUCTURE_TAG

The G2 Gateway data structure that corresponds to a structure in G2 is a *gsi_item* having the type tag *GSI_STRUCTURE_TAG*. The components of the structure can be accessed with the functions:

```
gsi_attr_count_of()
gsi_attrs_of()
gsi_set_attrs()
```

Wide String Type

G2 Gateway 5.0 uses the C type *gsi_char* to support either 8-bit or 16-bit Unicode (wide string) characters.

All the G2 Gateway API functions and callback functions that in earlier versions of G2 Gateway accepted or returned *char** now accept or return *gsi_char**, except for *gsi_start()*, which retains a *char* argument (for the *argv* argument passed to it from the command line) and for *gsi_initialize_callbacks()*.

To make *gsi_char* support wide string characters, compile your G2 Gateway user code with the C preprocessor macro *GSI_USE_WIDE_STRING_API* defined. When *GSI_USE_WIDE_STRING_API* is not defined, the *gsi_char* type is defined by default to be a *char* (8 bits).

You can define the C preprocessor macro *GSI_USE_WIDE_STRING_API* in your compiler command line. For example, under UNIX, you can use the following compile time switch:

```
-DGSI_USE_WIDE_STRING_API
```

You can also define *GSI_USE_WIDE_STRING_API* by including the following statement

```
#define GSI_USE_WIDE_STRING_API
```

before the *#define "gsi_main.h"* statement in any C file that includes *gsi_main.h*.

When you compile your user code with the *GSI_USE_WIDE_STRING_API* preprocessor macro defined, the *main()* function in *gsi_main.c* automatically

calls *gsi_set_option(GSI_WIDE_STRING_API)*. The *GSI_WIDE_STRING_API* runtime option must be set in order to define the *gsi_char* type as a *short*.

Note If you use a C file for your *main()* function other than the *gsi_main.c* provided with G2 Gateway, and you need to define the *gsi_char* type as a *short*, you must include in your *main()* C file a call to *gsi_set_option(GSI_WIDE_STRING_API)*.

G2 Data Types and G2 Gateway Type Tags

G2 Gateway data structures are assigned data type tags that enable API functions to perform type-checking on the data structures and to handle the data structures appropriately to their type. For information about how G2 Gateway sets and uses data type tags, see Type Tags of G2 Gateway Data Structures.

For information about how C and C++ data types are converted to G2 types, see the *G2 Reference Manual*.

The following tables list G2 data types and classes and the corresponding G2 Gateway type tags.

G2 Data Type	G2 Gateway Type Tag
No value	*GSI_NULL_TAG*
float	*GSI_FLOAT64_TAG*
integer	*GSI_INTEGER_TAG*
long	*GSI_LONG_TAG or GSI_INT64_TAG*
truth-value	*GSI_LOGICAL_TAG*
symbol	*GSI_SYMBOL_TAG*
text	*GSI_STRING_TAG*
sequence	*GSI_SEQUENCE_TAG*
structure	*GSI_STRUCTURE_TAG*

G2 Class Type	G2 Gateway Type Tag
float-variable	*GSI_FLOAT64_TAG* or *GSI_NULL_TAG*
integer-variable	*GSI_INTEGER_TAG* or *GSI_NULL_TAG*

G2 Class Type	G2 Gateway Type Tag
quantitative-variable	*GSI_INTEGER_TAG* or *GSI_FLOAT64_TAG* or *GSI_NULL_TAG*
logical-variable	*GSI_LOGICAL_TAG* or *GSI_NULL_TAG*
symbolic-variable	*GSI_SYMBOL_TAG* or *GSI_NULL_TAG*
text-variable	*GSI_STRING_TAG* or *GSI_NULL_TAG*
float-parameter	*GSI_FLOAT64_TAG* or *GSI_NULL_TAG*
integer-parameter	*GSI_INTEGER_TAG* or *GSI_NULL_TAG*
long-parameter	*GSI_LONG_TAG or GSI_NULL_TAG*
quantitative-parameter	*GSI_INTEGER_TAG or GSI_LONG_TAG or GSI_FLOAT64_TAG or GSI_NULL_TAG*
logical-parameter	*GSI_LOGICAL_TAG or GSI_NULL_TAG*
symbolic-parameter	*GSI_SYMBOL_TAG or GSI_NULL_TAG*
text-parameter	*GSI_STRING_TAG or GSI_NULL_TAG*
integer-array	*GSI_INTEGER_ARRAY_TAG*
symbol-array	*GSI_SYMBOL_ARRAY_TAG*
text-array	*GSI_STRING_ARRAY_TAG*
truth-value-array	*GSI_LOGICAL_ARRAY_TAG*
float-array	*GSI_FLOAT64_ARRAY_TAG*
quantity-array	*GSI_QUANTITY_ARRAY_TAG*
item-array	*GSI_ITEM_ARRAY_TAG*
value-array	*GSI_VALUE_ARRAY_TAG*
g2-array	*GSI_ITEM_OR_VALUE_ARRAY_TAG*
integer-list	*GSI_INTEGER_LIST_TAG*
symbol-list	*GSI_SYMBOL_LIST_TAG*
text-list	*GSI_STRING_LIST_TAG*

G2 Class Type	G2 Gateway Type Tag
truth-value-list	*GSI_LOGICAL_LIST_TAG*
float-list	*GSI_FLOAT64_LIST_TAG*
quantity-list	*GSI_QUANTITY_LIST_TAG*
item-list	*GSI_ITEM_LIST_TAG*
value-list	*GSI_VALUE_LIST_TAG*
g2-list	*GSI_ITEM_OR_VALUE_LIST_TAG*
All other classes	*GSI_NULL_TAG*

The following table summarizes the G2 variable and parameter types that correspond to the return values of *gsi_class_type_of()* and *gsi_history_type_of()*.

G2 Variable or Parameter Type	G2 Gateway Type Tag
float-variable	*GSI_FLOAT64_TAG*
integer-variable	*GSI_INTEGER_TAG*
long-variable	*GSI_LONG_TAG or GSI_INT64_TAG*
quantitative-variable	*GSI_QUANTITY_TAG*
logical-variable	*GSI_LOGICAL_TAG*
symbolic-variable	*GSI_SYMBOL_TAG*
text-variable	*GSI_STRING_TAG*
float-parameter	*GSI_FLOAT64_TAG*
integer-parameter	*GSI_INTEGER_TAG*
quantitative-parameter	*GSI_QUANTITY_TAG*
logical-parameter	*GSI_LOGICAL_TAG*
symbolic-parameter	*GSI_SYMBOL_TAG*
text-parameter	*GSI_STRING_TAG*

G2 Gateway Data Types for RPC Arguments

The following table lists data types for API function arguments:

G2 Gateway Data Types for Function Arguments

Type	Description
gsi_call_identifier_type	The type of a **call identifier**, which distinguishes remote procedure function calls from each other. When data is returned to the place where the remote procedure call originated (either G2 or a G2 Gateway bridge), the call identifier identifies the original call. To enable the use of call identifiers, you must compile your G2 Gateway code with the *GSI_USE_USER_DATA_FOR_CALLBACKS* preprocessor macro defined or use the corresponding compile time switch. For more info see Call Identifiers for Remote Procedure Calls. When the use of call identifiers is enabled: • Calls to the API functions *gsi_rpc_call()* and *gsi_rpc_call_with_count()* must include a call identifier argument. The programmer must provide a value for this argument. • A G2 Gateway receiver function receives a call identifier value from G2, which identifies the original call made by G2 Gateway to the G2 function. • A call identifier is the return value of the function *gsi_last_error_call_handle()*. G2 always generates a call identifier value for G2 Gateway local functions that it invokes. A local function invoked by G2 can use the call identifier as an argument to *gsi_rpc_return_values()* or *gsi_rpc_return_error_values()* if it returns a value or signals an error to G2.

G2 Gateway Data Types for Function Arguments

Type	Description
gsi_context_user_data_type	The type of **context user data**, a value provided by the programmer to identify a particular context. Context user data can be useful for identifying connections between different locations, or connections used for different purposes. A context user data value can be associated with a connection in either or two ways: • As an argument to a call to *gsi_initiate_connection_with_user_data()* that initiates the connection. • As an argument to *gsi_set_context_user_data()*, to associate context user data with an existing connection. The API function *gsi_context_user_data()* returns the context user data associated with a specified connection.
gsi_function_handle_type	The type of **function handle** arguments, which identify G2 procedures in calls to the following API functions: *gsi_rpc_declare_remote()*, *gsi_rpc_declare_remote_with_error_handler_and_user_data()*, *gsi_rpc_start()*, *gsi_rpc_start_with_count()*, *gsi_rpc_call()*, *gsi_rpc_call_with_count()*, *gsi_set_rpc_remote_return_value_kind()*.
gsi_item_user_data_type	The type of the *user_data* argument of *gsi_set_user_data()* and of the return value of *gsi_user_data_of()*. This datatype is aliased to void *, and can therefore be a pointer to any type of data. *gsi_item_user_data_type* provides a way for you to manage the data that you are passing to and from API functions, and thus gets passed back to G2 Gateway in callback functions. You would typically use a cast operation with *gsi_item_user_data_type* to access certain types of data, such as structure members.

G2 Gateway Data Types for Function Arguments

Type	Description
gsi_procedure_user_data_type	The type of **procedure user data**. A procedure user data argument can be the first argument of local functions, receiver functions, and error receiver functions in a G2 Gateway application. To enable the use of call identifiers, you must compile your G2 Gateway code with the GSI_USE_USER_DATA_FOR_CALLBACKS preprocessor macro defined or use the corresponding compile time switch. For more info see Call Identifiers for Remote Procedure Calls. When you use procedure user data, you must include a *procedure_user_data* argument in the calls to *gsi_rpc_declare_local()* and *gsi_rpc_declare_remote()* that you use to declare the local and remote functions. For more information about procedure user data, see Procedure User Data for Remote Procedure Calls.
gsi_rpc_receiver_fn_type	The type of a receiver function in your G2 Gateway code. The API functions *gsi_rpc_declare_remote()* and *gsi_rpc_declare_remote_with_error_handler_and_user_data()* must include an argument that is a pointer to a function of this type.
gsi_rpc_local_fn_type	The type of a local function in your G2 Gateway code. The API function *gsi_rpc_declare_local()* must include an argument that is a pointer to a function of this type.
gsi_symbol_user_data_type	The type of **symbol user data**, which you can associate with symbol values. The API function *gsi_set_symbol_user_data()* associates symbol user data with a symbol value, and the function *gsi_symbol_user_data()* returns the symbol data associated with a function. The data type *gsi_symbol_user_data_type* is defined to have the type (*void **).

G2 Gateway Data Types for Function Arguments

Type	Description
gsi_struct	The type of a function argument that can represent one or more of the following structures, depending on the function: *gsi_item*, *gsi_registered_item*, *gsi_attr*, or *gsi_registration*.
handle	The type of the first argument of *gsi_initialize_for_win32()*.

E

Limits and Ranges

Describes limits and ranges applicable in G2 Gateway.

Introduction

This appendix describes the limits and ranges applicable in G2 Gateway for:

- Contexts, objects, attributes, and error codes.
- G2 data types.
- Callback functions.
- API functions.
- Remote procedure calls.

Limits on Contexts, Objects, Attributes, and Error Codes

Limits on Contexts, Objects, Attributes, and Error Codes

G2 Feature	Limits
contexts	By default, a single G2 Gateway process can support a maximum of 50 contexts. You can change this default by calling *gsi_set_context_limit()*.
objects	There is no limit to the number of GSI variables per context in G2 5.0 and later. In G2 4.1 and earlier versions, the maximum number of GSI variables per context is 65,534 if no G2 Gateway RPCs have been made available to G2. This maximum number is 3 less for each G2 Gateway function that G2 calls as a remote procedure, and is 2 less for each G2 Gateway function that G2 starts as a remote procedure.
identifying attributes	A GSI interface can have a maximum of 6 identifying attributes. Strings that specify identifying attributes can have a maximum of 64K characters.
attribute names	The maximum length of attribute names is 64K characters.
error codes	Error codes must be positive integers not greater than 65,537.

Limits on G2 Data Types

Limits on G2 Data Types

G2 Data Type	Limits
integer	In G2 Standard (32-bit), the G2 integer type is a signed 30-bit integer, between -536870912 and 536870911, inclusive. In G2 Enterprise (64-bit), the G2 integer type is a signed 61-bit integer, between -1152921504606846976 and 1152921504606846975, inclusive. Both 32-bit and 64-bit GSI libraries are provided in G2 Enterprise. GSI communications between 32-bit GSI bridge and 64-bit G2 Server, or 64-bit GSI bridge and 32-bit G2 Server, should consider the range of G2 integer type is a signed 30-bit integer, between -536870912 and 536870911, inclusive. Otherwise there will be data lost. G2 Gateway applications should send large numeric values to G2 quantitative-variable variables, rather than to integer-variable variables; G2 then automatically assigns any values exceeded integer ranges to floating-variable variables.
long	The G2 long type is a signed 64-bit integer, between -9223372036854775808 and 9223372036854775807, inclusive.
float	The G2 float type is an IEEE double float (64-bit).
text	To allow for internal header information, text strings should contain no more than 60 thousand characters.
symbol	To allow for internal header information, symbols should contain no more than 60 thousand characters.

Limits on Callback Functions

Limits on Callback Functions

Callback Function	Limits
gsi_initialize_context()	The maximum length of the string passed to this function as the remote process initialization string argument is 64K.
gsi_receive_message()	This maximum length of the string received by this function is 64K, including the null terminator.
gsi_set_data()	The longest string that this function can send to G2 Gateway is 64K characters.

Limits on API Functions

Limits on API Functions

API Function	Limits
gsi_return_attrs()	The maximum length of strings that this function can send to G2 is 64K.
gsi_return_message()	The maximum length of strings that this function can return to G2 is 64K.
gsi_return_timed_attrs()	The maximum length of strings that this function can send to G2 is 64K.
gsi_return_timed_values()	The maximum length of strings that this function can send to G2 is 64K.
gsi_return_values()	The maximum number of values that this function can send to G2 in a single call is 200. The maximum length of strings that this function can send to G2 is 64K.

The maximum number of bytes that G2 Gateway can send to G2 in a single API call is 60K.

Limits on Remote Procedure Calls

Note the following limits on remote procedure calls in G2 Gateway:

- The maximum number of G2 procedures that G2 Gateway can declare and invoke as remote procedures is 64K.
- The maximum number of arguments in a remote procedure call, from G2 Gateway to G2 or from G2 to G2 Gateway, is 1024.
- The maximum size of arguments in a remote procedure call, from G2 Gateway to G2 or from G2 to G2 Gateway, is 64K.
- The maximum number of arguments that a G2 procedure, invoked by G2 Gateway as a remote procedure, can return to a G2 Gateway return handler function, is 64K.
- The maximum length of remote procedure names is 64K characters.
- There is no limit on the number of simultaneously executing remote procedure calls. (In GSI 4.0, the maximum number of simultaneously executing remote procedure calls is 4,096.)

F

How G2 and G2 Gateway Exchange Data

Provides a brief summary of techniques for exchanging data between a G2 Gateway bridge and a G2 KB.

Introduction

This appendix contains figures that illustrate how to implement the following kinds of data exchange between G2 and a G2 Gateway bridge:

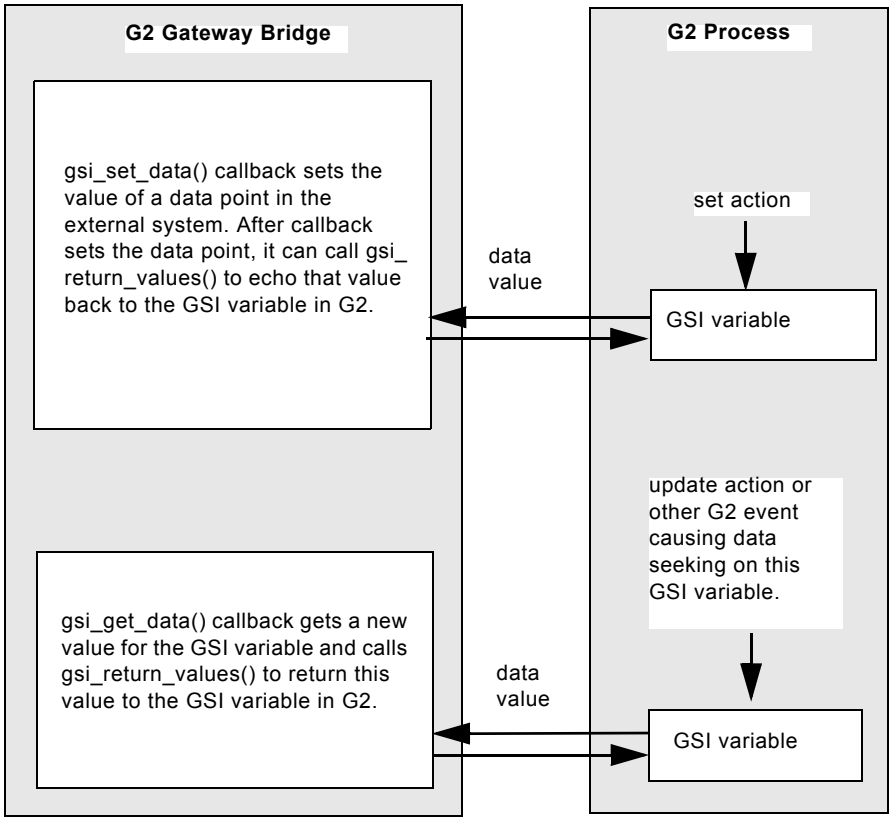
- Setting the value of a data point in an external system with the value of a GSI variable in G2, and updating a GSI variable in G2 with the value of an external data point.
- Receiving unsolicited data from the G2 Gateway bridge.
- Starting and calling remote procedures in the G2 Gateway bridge, from G2.
- Starting and calling G2 procedures, from the G2 Gateway bridge.
- Sending text messages from G2 to the G2 Gateway bridge, and from the G2 Gateway bridge to G2.

Setting an External Data Point and Updating a GSI Variable

The following figure illustrates how to:

- Run a set action on a GSI variable to update the value of a data point in an external system, using the value of the GSI variable.
- Run an update action on a GSI variable to update the value of that variable, using a value retrieved from a data point in an external system.

Setting an External Data Point and Updating a GSI Variable



For more information, see:

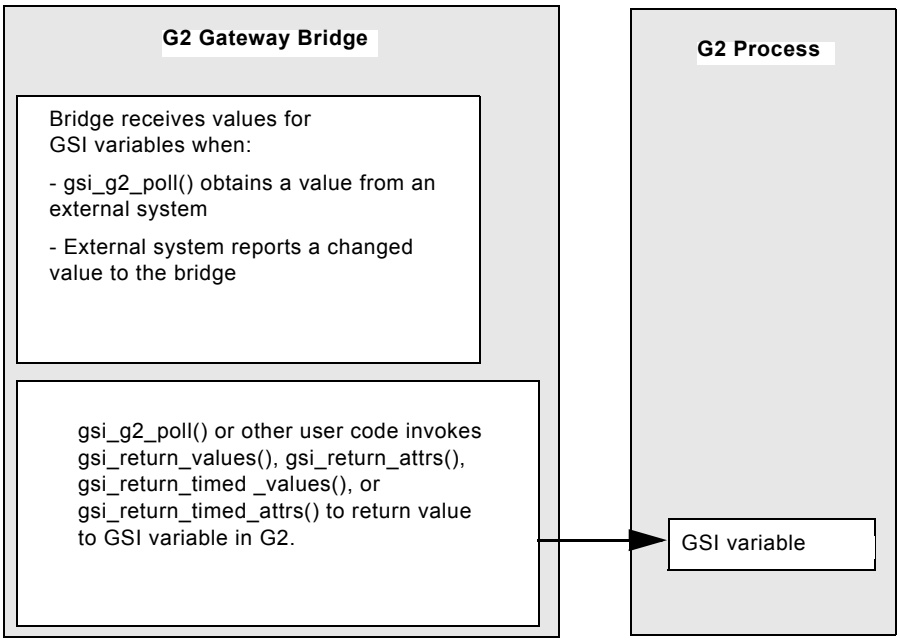
- Implementing Data Service in G2 Gateway.
- *gsi_get_data*.
- *gsi_set_data*.
- *gsi_return_values*.

- Returning Solicited Data to G2, for information about when G2 seeks data for GSI variables.
- *G2 Reference Manual*, for information about the set and update actions, and about data seeking for GSI variables.

Receiving Unsolicited Data from a G2 Gateway Bridge

The following figure illustrates how a G2 Gateway bridge returns unsolicited data to G2:

Receiving Unsolicited Data from a G2 Gateway Bridge



For more information, see:

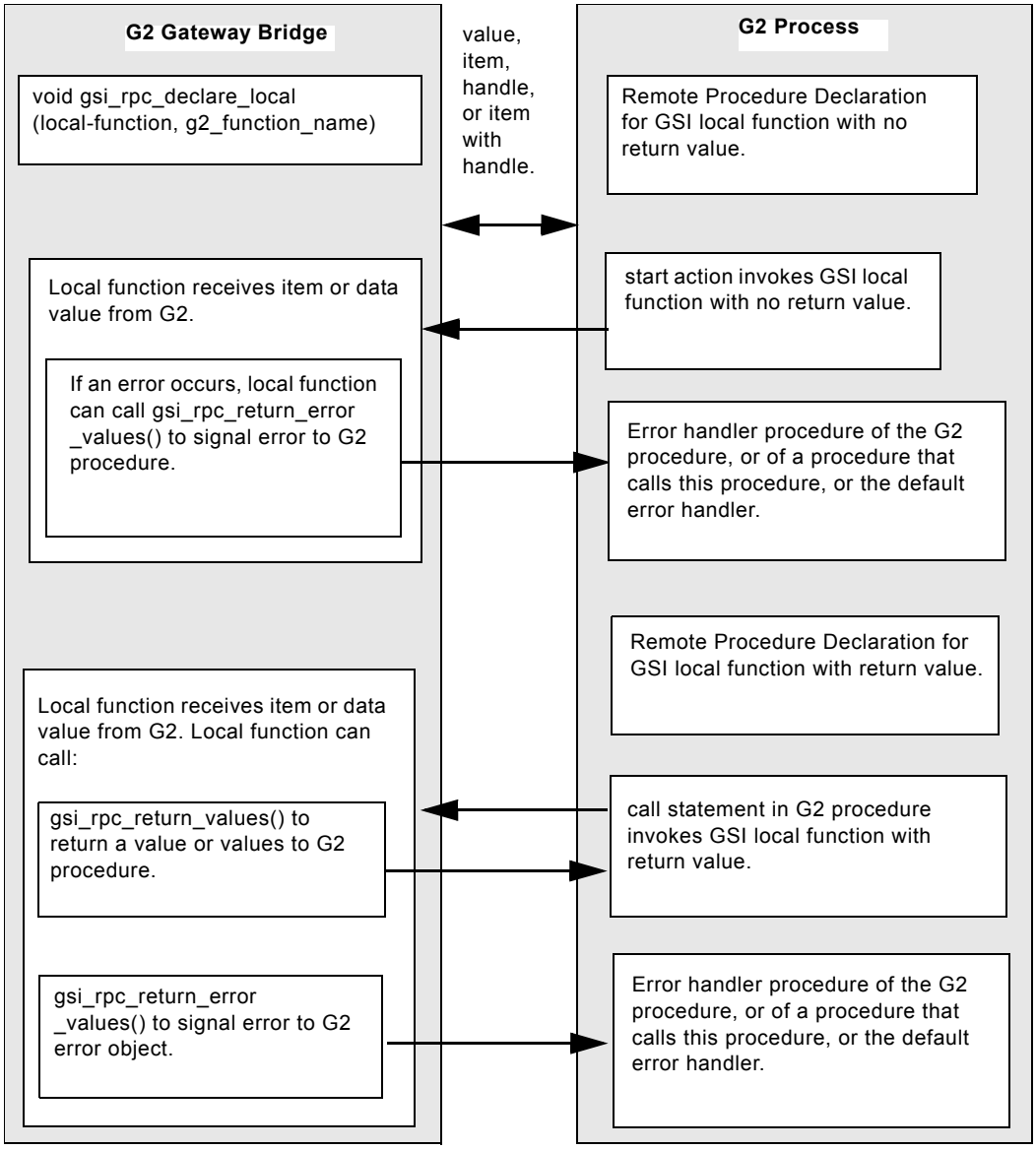
- Sending Unsolicited Data to G2.
- *gsi_g2_poll*.
- *gsi_return_attrs*.
- *gsi_return_timed_attrs*.
- *gsi_return_timed_values*.
- *gsi_return_values*.

Invoking a Local Function in a G2 Gateway Bridge from G2

The following figure illustrates how to:

- Invoke a G2 Gateway local function as a remote procedure using the start action in G2.
- Invoke a G2 Gateway local function as a remote procedure using the call statement in a G2 procedure.

Invoking G2 Gateway Local Functions From G2



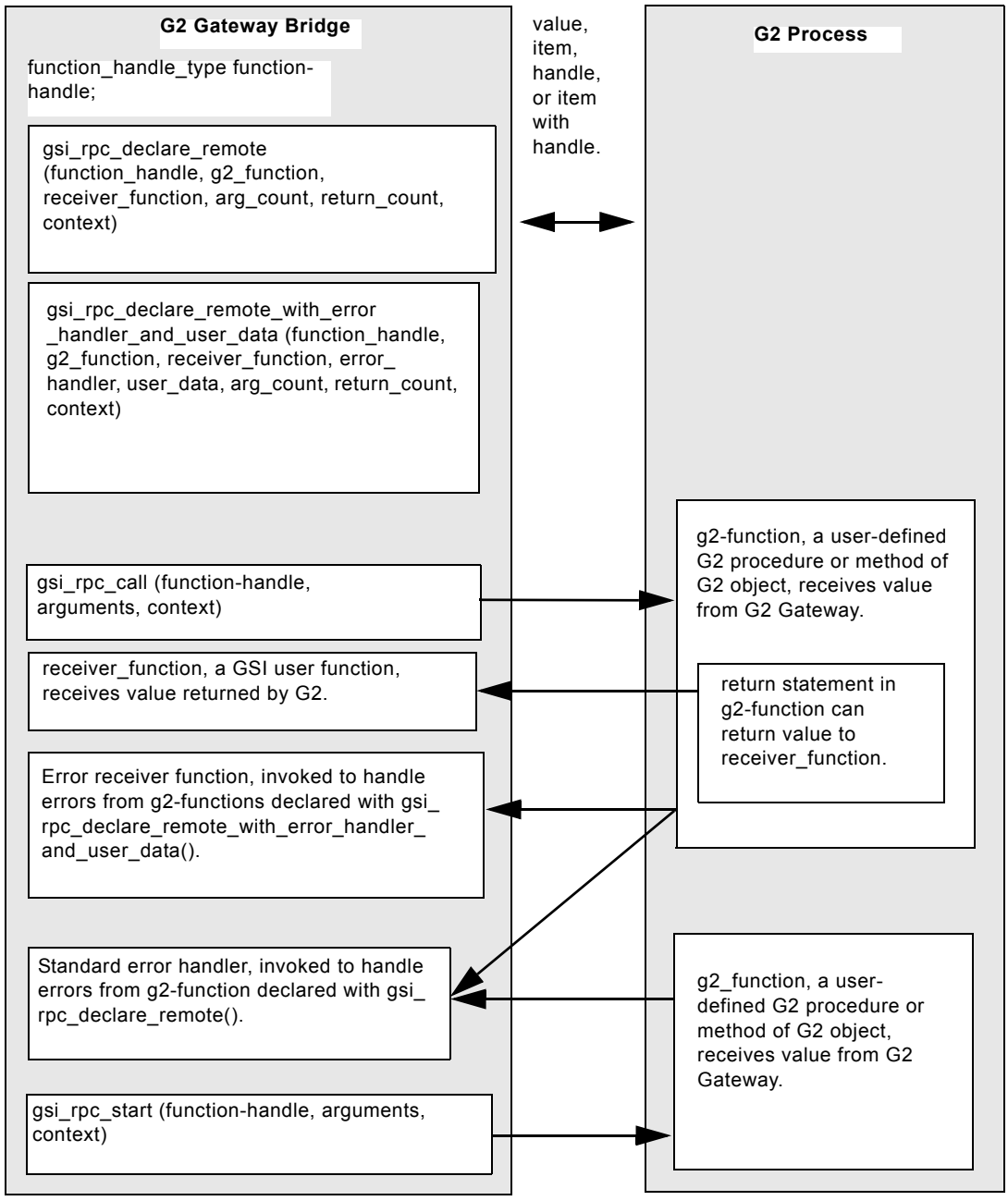
For more information, see:

- Making Remote Procedure Calls from G2 to the G2 Gateway Bridge.
- *gsi_rpc_declare_local*.
- *gsi_rpc_return_values*.
- *G2 Reference Manual*, for information about the start action and the call procedure statement.

Invoking G2 Procedures and Methods from a G2 Gateway Bridge

The following figure illustrates how to call and start a G2 procedure or a method of a G2 object from the G2 Gateway bridge:

Invoking G2 Procedures or Methods from a G2 Gateway Bridge



For more information, see:

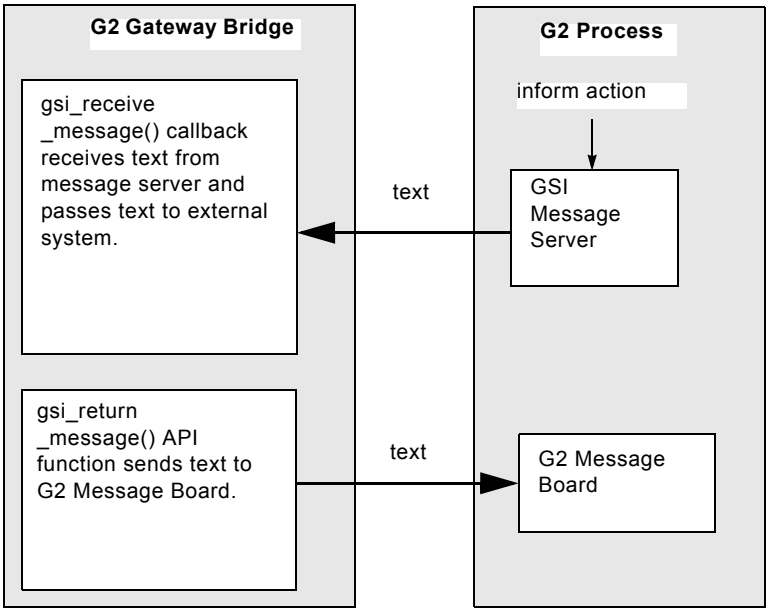
- Making Remote Procedure Calls from a G2 Gateway Bridge to G2.
- *gsi_rpc_declare_remote*.
- *gsi_rpc_call*.
- *gsi_rpc_start*.
- *G2 Reference Manual*, for information about the call procedure statement and about remote procedure declarations.

Exchanging Text Messages Between G2 and a G2 Gateway Bridge

The following figure illustrates how to:

- Send a text message to a G2 Gateway bridge.
- Send a text message to the G2 Message Board.

Exchange of Text Messages Between a G2 Gateway Bridge and G2



For more information, see:

- Message Passing.
- *gsi_receive_message*.

- *gsi_return_message*.
- *G2 Reference Manual*, for information about the inform action.

Upgrading G2 Gateway Applications

Describes how to upgrade existing GSI applications to the current version of G2 Gateway.

Introduction

This chapter describes the changes introduced by G2 Gateway Version 6.0 from the previous release of this product, GSI Version 5.1. It also provides upgrade procedures from GSI 4.1 or G2 Gateway 5.0 to G2 Gateway 7.0.

You can use the information in this chapter as a guideline when you modify applications based on versions of the product earlier than G2 Gateway 6.0.

Support of Earlier GSI Versions

Gensym does not support versions of GSI prior to G2 5.1 Rev. 9. This means that Gensym will not fix GSI bugs, produce patches, or add functionality to a version of GSI earlier than G2 5.1 Rev. 9. Customers running earlier versions of GSI should upgrade to a supported version. Gensym neither recommends nor supports new bridge development in an unsupported version of G2 Gateway.

All versions of GSI, however, regardless of age, do remain compatible with newer supported versions of G2. The oldest GSI executable can still run with all subsequent versions of G2. This means you can run bridges built and compiled on earlier versions of GSI with G2 5.1 Rev. 0 and higher.

Do not expect, however, your GSI bridge executable to use features from a version of G2 developed after the GSI version you are using was developed. Object passing, for example, was introduced in G2 4.0, so a GSI 3.2 bridge running with G2 4.0 will not have object passing capability.

GSI 4.1 Support Policy

You can still run your GSI 4.1 bridge executables with G2 7.0.

Gensym will no longer ship GSI 4.1, nor will Gensym fix GSI 4.1 bugs, produce patches, or develop new features in the GSI 4.1 sources in response to developer's requests.

If you report problems with your GSI 4.1 bridge running on a supported version of G2, Gensym will evaluate your problem and support fixes required in the G2 sources, but not in the GSI 4.1 sources.

New G2 Gateway 6.0 Features

G2 Gateway Version 6.0 introduced the following changes:

- Support for the Windows 2000/XP and Linux operating systems.
- Increase in the maximum text size from 65,533 to 1 million.
- The ability to return large amounts of data to a G2 array.

In previous versions of G2 Gateway, the limit for returning a G2 array was 65,533. If data exceeded this amount, G2 Gateway returned the array unappended and without generating a warning.

In G2 Gateway 6.0, the limit for returning a G2 array has increased to 1 million. If data is within this limit, G2 Gateway appends the updated values to the end of the array. If data exceeds this limit, G2 Gateway generates a warning.

New API Functions

gsi_print_backtrace

Prints a backtrace to the console on Sun4 and Solaris platforms.

Synopsis

```
void gsi_print_backtrace()
```

Description

gsi_print_backtrace() prints a backtrace to the console. If the G2 Gateway executable is stripped, G2 Gateway prints a numeric bracktrace; if it is not, G2 Gateway prints a symbolic backtrace. This function is useful for debugging.

New Runtime Options

GSI_PROTECT_INNER_CALLS

After encountering an error, G2 Gateway returns control to the caller rather than returning control to *gsi_run_loop()*.

For example, suppose *gsi_run_loop()* calls *gsi_receive_deregistrations()*, which then calls an API function. If G2 Gateway encounters an error in the API function and GSI_PROTECT_INNER_CALLS is not set, it returns control to gsi_run_loop(). This can cause undesirable results if *gsi_receive_deregistrations()* is unable to complete some of its tasks, such as freeing the memory reserved for registered variables. This same scenario may be applied to housekeeping activities usually performed by *gsi_shutdown_context()*.

You should be aware that this runtime option can make G2 Gateway API functions slower, but it prevents a potential cause of G2 Gateway aborts and protocol-out-of-synchronization problems. However, using the *GSI_PROTECT_INNER_CALLS* runtime option can prevent hangs in applications that use the *gsi_run_state_change* callback.

This runtime option was introduced in G2 Gateway 5.0 but not previously documented.

Description

gsi_set_option(GSI_PROTECT_INNER_CALLS) returns control to the caller rather than returning control to *gsi_run_loop()* after encountering an error.

gsi_reset_option(GSI_PROTECT_INNER_CALLS) returns control to *gsi_run_loop()* after encountering an error. This is the default.

GSI_TRACE_RUN_LOOP

Prints a message whenever *gsi_start()* or *gsi_run_loop()* are entered or exited.

Description

gsi_set_option(GSI_TRACE_RUN_LOOP) causes messages to print whenever *gsi_start()* or *gsi_run_loop()* are entered or exited.

gsi_reset_option(GSI_TRACE_RUN_LOOP) disables this option. This is the default.

This runtime option was introduced in G2 Gateway 5.0 but not previously documented.

Changes to G2 Gateway 6.0

Make File Changes

Some of the libraries with which G2 Gateway 6.0 applications link are different than the ones with which G2 Gateway 5.1 linked.

On Windows platforms, G2 Gateway 6.0 no longer uses the *$(conlibs*) library and links with *$(guilibsmt)* in place of the *$(guilibs)* library. See Compiling and Linking G2 Gateway on Windows for more information.

On Solaris platforms, the dynamic loader library (*-ldl*) should be included in your make file when linking your G2 Gateway application. This switch is now included in the sample make file provided with G2 Gateway.

gsi_main.c Changes

The *gsi_main.c* file included with your version of G2 Gateway now includes a call to *GSI_SET_OPTIONS_FROM_COMPILE()*, which is a C macro that sets options based on settings of the C preprocessor flags. If you are not using the standard *gsi_main.c* file, you should include a call to *GSI_SET_OPTIONS_FROM_COMPILE()* before the call to *gsi_start()* in your *gsi_main.c main()* function.

gsi_misc.h Changes

The *gsi_misc.h* file has been updated so that you no longer need to add the extern C declaration when linking with C++ applications.

Superseded Practices

VMS is not supported for G2 Gateway 6.0 Rev. 0 and higher.

32-bit and 64-bit Support for G2 Gateway

Gensym is continually updating support for G2 Gateway to operate in 32-bit and 64-bit environments. See the Release Notes for the most up-to-date information about running G2 Gateway in your environment.

The following table lists the Application Binary Interface (ABI) with which G2 Gateway can operate for each supported operating system.

G2 Gateway Processor Support

Operating System	ABI
Compaq Tru64 Unix	Alpha (ECOFF)
HP HP-UX 11.00 and HP-UX 11i V2	HP9000 (PA-RISC 2.0)
HP HP-UX 11.23	HP Itanium
IBM AIX	RS/6000 (32-bit XCOFF)
Microsoft Windows	NT4/2000/XP (32-bit COFF)
RedHat Linux	Intel (32-bit ELF)
Sun Solaris	Sparc (32-bit ELF)

For compatibility with G2 Gateway libraries, you must compile and link your applications in the modes listed in the above table. If your compiler is setup for some other mode, you must set the flags necessary to conform to these settings. The flags to set differ depending on the operating system and your system's configuration. Consult the documentation for your system to determine which flags to set.

G2 Gateway provides a separate component called G2 Gateway (GSI) for HP-UX Itanium, which is available on HP platforms only. This component installs into the `gsi-itanium` directory in the G2 bundle installation directory, a parallel directory to the `gsi` directory. The GSI libraries and examples in the `gsi-itanium` directory have been compiled on an HP Itanium machine running HP-UX 11.23, and are in the native ELF format. All other files in the G2 bundle on the HP platform are compiled and linked on a PA-RISC machine running HP-UX 11.00 and will run in compatibility mode on the HP Itanium machine under HP-UX 11.23.

Previously Undocumented Changes in 5.0

Beginning with G2 Gateway 5.0 Rev. 0, handle assignments begin with the number 1. Prior to 5.0 Rev. 0, handle assignments began with the number 2.

A handle is an integer that uniquely identifies items passed between G2 to G2 Gateway. A handle is assigned to all registered variables and can be assigned to items passed as arguments to remote procedure calls between G2 and G2 Gateway. If G2 Gateway is version 5.0 or higher, G2 sets the first handle assignment to 1. If the G2 Gateway (or GSI) version is prior to 5.0, G2 sets the first handle assignment to 2.

In versions prior to 5.0, G2 Gateway deregistered large numbers of GSI variables in batches. One call to *gsi_receive_deregistrations()* could derigister up to 48 variables. Beginning with 5.0, G2 Gateway deregisters variables one at time. In other words, each variable requires a separate call to *gsi_receive_deregistrations()*. This change should not affect performance. Although incorporated in 5.0, this change of functionality was not previously documented.

Changes to API Functions in G2 Gateway 5.0

This section lists the G2 Gateway API functions whose functionality or calling signature is different from what it was in previous versions.

gsi_watch_fd() functionality change. Starting with G2 Gateway 5.0, you can only call *gsi_watch_fd()* on open file descriptors. This functionality change was not documented in 5.0.

gsi_unwatch_fd() functionality change. Starting with G2 Gateway 5.0, you must also call *gsi_unwatch_fd()* before closing the file descriptor. This functionality change was not documented in 5.0.

Upgrading from GSI 4.1 to G2 Gateway to 7.0

After upgrading from GSI 4.0 to G2 Gateway 7.0, you must update your current files so that they are compatible with the new version. If you do not, your applications will not run due to version inconsistencies. If you have already upgrade from Version 4.1 to Version 6.0, these steps are not necessary.

To update your files:

1. Replace all G2 Gateway libraries with the 7.0 versions of these files.
2. Replace the header files *gsi_main.h* and *gsi_misc.h* with the G2 Gateway 7.0 versions.
3. Incorporate the changes from *gsi_main.c* and *gsi_misc.c* into your applications.

   If you have not made changes to these files, you can simply compile and link your applications with the 7.0 versions of the files. You should be aware that there is a new version control variable, *GSI_INCLUDE_REV_VER_NUM*, in *gsi_main.c*. The syntax in *gsi_main.c*:

```
gsi_set_include_file_version(
   GSI_INCLUDE_MAJ_VER_NUM,
   GSI_INCLUDE_MIN_VER_NUM,
   GSI_INCLUDE_REV_VER_NUM);
```

If you have incorporated the contents of *gsi_main.c* or *gsi_misc.c* into your source code or if you have additional information added to them, you should update them with the changes in the new files. The syntax to add the new version control variable, *GSI_INCLUDE_REV_VER_NUM*, is:

```
gsi_include_file_revision_version =
   GSI_INCLUDE_REV_VER_NUM;
```

4 Edit your code so that it agrees with any changes made to the signature of the G2 Gateway API functions. See the section, Changes to API Functions in G2 Gateway 5.0 for a list of changes.

Upgrading from G2 Gateway 5.0 to 7.0

After upgrading from G2 Gateway 5.0 to 7.0, you must update your current files so that they are compatible with the new version. If you have already upgraded from Version 5.0 to Version 6.0, and you are now upgrading to Version 7.0, these steps are not necessary. If you do not make these changes, your applications will not run due to version inconsistencies. To update your files:

1 Replace all G2 Gateway libraries with the 7.0 versions of these files.

2 Replace the header files *gsi_main.h* and *gsi_misc.h* with the G2 Gateway 7.0 versions.

3 Incorporate the changes from *gsi_main.c* and *gsi_misc.c* into your applications.

If you have not made changes to these files, you can simply compile and link your applications with the 7.0 versions of the files.

If you have incorporated the contents of *gsi_main.c* or *gsi_misc.c* into your source code or if you have additional information added to them, you should update them with the changes in the new files.

4 Edit your code so that it agrees with any changes made to the signature of the G2 Gateway API functions. See the section, Changes to API Functions in G2 Gateway 5.0 for a list of changes.

Glossary

A

access functions: A library of API functions provided with G2 Gateway that enable your user code to access information in the internal G2 Gateway data structures `gsi_registration`, `gsi_registered_item`, `gsi_item`, and `gsi_attr`. For information about the access functions provided with G2 Gateway, see API Functions.

application programmer interface (API): A formally defined programming language interface. G2 Gateway provides its own library of API functions, which you need in order to develop the user code of your G2 Gateway application. For information about the API functions provided with G2 Gateway, see API Functions.

The external system to which you are building the bridge may also provide an API to enable a bridge process to communicate with the external system. If you use this API, you must link it to the bridge, usually as an object library. The API of your external system is an optional part of a G2 Gateway application (not required for it to function). It is not provided by Gensym. Most APIs are developed on site or by a third-party vendor.

B

bridge: A software application that provides an interface between G2 and some external application, device, or system. If a bridge is not provided by Gensym for your external application, you can create your own bridge, using G2 Gateway to complete the bridge source code. The bridge source code includes G2 Gateway libraries and standard G2 Gateway functions for transmitting data values and text messages, and for starting, pausing, resuming, and stopping the external system. See also *bridge process* and *stub functions*.

bridge process: A running executable that connects the G2 process to the external system. The bridge process consists of G2 Gateway libraries of API functions, G2 Gateway callback functions that you complete to suit your application, and an optional API to an external system, provided by you or by a third-party developer. See also *application programmer interface*, *G2 process*, *G2 Standard Interface*, and *user code*.

C

call: An action in G2 that can be used to invoke a remote procedure in the bridge process and to return values from that procedure. This action is also used from within G2 procedures to invoke other G2 procedures. See also *start* and *remote procedure call*.

call handle: An integer that G2 generates to identify a particular remote procedure call to a G2 Gateway local function, within the current context. The API function `gsi_rpc_return_values()` references the call handle to indicate which outstanding remote procedure call, within the specified context, to return values to in G2.

callback functions: Functions that form the basis of your user code. G2 Gateway invokes callback functions *automatically*, when network events occur on a connection to a G2 KB. Each callback is invoked to respond to a particular event, such as the activation of a GSI interface or a request by G2 for a value for a GSI variable. Your user code never needs to invoke callback functions explicitly, and should not attempt to do so.

Callback functions are invoked only while your G2 Gateway bridge process is executing under the control of `gsi_run_loop()`, the API function that establishes the main event-processing loop of your G2 Gateway bridge process.

See also *skeleton file* and *stub functions*.

communications link: The physical link that enables the G2 process and the G2 Gateway bridge process to communicate with each other. The link includes a transport layer protocol (TCP/IP) and Gensym's Intelligent Communications Protocol (ICP). The transport layer protocol used depends on the platform that you are using. See also *Intelligent Communications Protocol* and *TCP/IP*.

context: A connection between an instance of a GSI interface residing in a G2 knowledge base and a bridge process. You can have multiple contexts between a single bridge process and one or more GSI interfaces in one or more G2 processes. See also *current context*.

Continuous Mode: The default operational mode of a G2 Gateway bridge process. When a bridge is running in continuous mode, `gsi_start()` invokes the API function `gsi_run_loop()` repeatedly. `gsi_run_loop()` returns control only when a fatal error occurs.

Continuous mode is the better mode for polling an external system for data. The bridge can poll the external system using the callback function `gsi_g2_poll()`, which is invoked by G2 Gateway approximately once per second.

See also *One-Cycle Mode* and *G2 Gateway options*.

current context: The one connection (out of many possible connections) between the G2 Gateway bridge process and a G2 KB that is being served by the G2 Gateway bridge process at the present time. G2 Gateway uses an integer (`gsi_`

int) to identify each context. The API function *gsi_current_context()* returns the number of the current context. You can use a macro, *current_context*, to access *gsi_current_context()*. Note that macros are not available if you define *__GENSYM_NOALIAS__* when you compile your G2 Gateway bridge. See also *context*.

D

data point: A value in an external system, represented in a G2 knowledge base by a GSI variable. Your G2 KB can use the GSI variable to read from and write to the data point.

data seeking: Providing values for variables. Data seeking occurs when a GSI variable has the value gsi-data-server in its data-server attribute, and either of the following conditions is true: the variable's default-update-interval expires, or G2 requires a value from the last-recorded-value attribute of the variable and the value of this attribute has expired or is set to no value. Data seeking causes the function *gsi_get_data()* to be called within the bridge.

data server: Supplies the G2 process with data from a real, simulated, or recorded environment. G2 Gateway is one possible data server for objects in G2.

In addition, data-server is the name of an attribute of a g2-variable that specifies the data server from which the variable receives its values. If the data-server attribute is specified as gsi-data-server, the variable receives a value from an external system through G2 Gateway. The data server of a GSI variable must be gsi-data-server. See also *G2 Gateway* and *GSI variable*.

E

echo: The return to G2 of a value that has been set in the external system. When G2 executes a set action (for example, within a rule or procedure), a value is set for an external variable that corresponds to a GSI variable in G2. The value is first transferred over to G2 Gateway. G2 Gateway calls the function *gsi_set_data()* in the bridge to get the value resulting from the set action to the external system. After the value is set in the external system, the function *gsi_return_values()* can be used to return, or "echo", the value to the GSI variable that corresponds to the external variable. To ensure that the GSI variable is also set to the new value, the value must be echoed back to G2.

external system: An application or system being directly affected or controlled by the G2 Gateway bridge process, such as a database or a PLC. Also, equipment or sensors dependent upon the system (external data points are often embodied by such equipment or sensors).

F

function handle: An integer used by G2 Gateway to identify the G2 procedure to which you are making a remote procedure call. In your G2 Gateway user code, you must declare a function handle as a variable of type `gsi_int`; as the name of this variable, you can specify the name of the G2 procedure, so that you can know which G2 procedure the handle refers to. You must specify the function handle variable as the first argument of `gsi_rpc_declare_remote()` when you call this API function to declare the G2 procedure.

G

G2 application: A system that uses one or more G2 processes to provide real-time reasoning and data processing.

G2 Gateway standard interface (GSI): A network-oriented toolkit that enables you to develop a bridge between G2 and an external system.

G2 process: A running G2 executable. A G2 process that uses G2 Gateway contains a GSI interface configured for communication with a G2 Gateway bridge process. The G2 process contains objects that exchange data with an external system through the G2 Gateway bridge. See also *bridge*, *bridge process*, *GSI interface*, and *GSI variable*.

Grouping: Using the values of the identifying attributes of GSI variables to indicate how requests for data (to update the values of the variables) are to be sorted and grouped. Requests are grouped according to the values of the identifying attributes that the variables have in common. You specify the identifying attributes by which to group requests in the grouping-specification attribute of a GSI interface used by the variables. G2 packs data requests together according to the specified attributes. See also *GSI variable* and *identifying attributes*.

gsi-interface class: A G2 standard class (gsi-interface). You can create a GSI interface as an instance of this class, or as an instance of a class that inherits from gsi-interface. See also *GSI interface* and *GSI variable*.

GSI interface: A user-created instance of the gsi-interface class, or of a class that inherits from gsi-interface. This object is configured for use by GSI variables and remote procedure calls to exchange data values and text messages with an external system.

You must create and configure a GSI interface for each connection between a G2 knowledge base and a G2 Gateway bridge process. You edit attributes of the GSI interface to configure one connection between G2 and a G2 Gateway bridge.

A GSI interface identifies a G2 Gateway bridge process with which the G2 process will attempt to establish a network connection, indicates whether the G2 Gateway bridge will receive unsolicited data from the external system, specifies whether G2 or the G2 Gateway bridge determines when data is passed from the bridge to G2, designates the identifying attributes of classes of GSI variables, and records

the current status of this particular connection between G2 and the G2 Gateway bridge process.

See also *identifying attributes*, *GSI interface class* and *GSI variable*.

GSI Message Server: A user-defined G2 object or message class that includes gsi-message-service as one of its direct superior classes. You run an inform action on a GSI message server to send a message to an external system. See also *inform*.

G2 Gateway options: Modes of operation that affect the runtime behavior of a G2 Gateway bridge process. You can set (turn on) or reset (turn off) the runtime options in the `gsi_set_up()` callback function, using the macros `gsi_set_option()` and `gsi_reset_option()`. All options are turned off (reset) by default.

GSI variable: A user-created instance of a GSI variable class, used by a G2 KB to read from and write to a data point in an external system. A G2 KB can use GSI variables to send values to and read values from data points in an external system. The current (last received) value of a GSI variable is stored in its last-recorded-value attributives information about G2 variables, see the *G2 Reference Manual*. See also *data server*, *GSI interface*, *GSI variable class*, and *identifying attributes*.

GSI variable class: A user-defined class definition, which includes among its direct superior classes one of the standard variable classes (integer-variable, float-variable, quantitative-variable, logical-variable, symbolic-variable, text-variable, or sensor), as well as the G2 mixin class gsi-data-service.

You create instances of GSI variable classes to represent individual data points in external systems. See also *data server*, *GSI interface*, *GSI variable*, and *identifying attributes*.

H

handle: See *call handle* and *item handle*.

history: The past values of a variable or parameter. Each value is stored with the date and collections time.

I

identifying attributes: A group of one to six attributes of a GSI variable, used by the G2 Gateway bridge to maintain a one-to-one mapping between that variable and the G2 Gateway data structure that represents the variable in the bridge. Identifying attributes can also be used by G2 to group data requests for GSI variables.

Identifying attributes provide a unique identifier for each GSI variable. The values of the identifying attributes of each GSI variable must distinguish that GSI variable from all other variables in the KB.

You specify which attributes of a GSI variable class are to be used as identifying attributes in the identifying-attributes attribute of the GSI interface. The gsi-interface-name attribute of each GSI variable must refer to the GSI interface that designates the identifying attributes for that variable's class.

The identifying attributes of a GSI variable must be *user-defined attributes* — either class-specific-attributes of the GSI variable's own class, or class-specific-attributes that the GSI variable inherits from a superior class.

Data should not be returned to identifying attributes of GSI variables.

inform: An action in G2 that sends a text message to a specified destination in an external system. When you run an inform action on a GSI message server in G2, G2 Gateway calls the callback function *gsi_receive_message()* to receive the message from G2 and send it to the external system. See also *GSI Message Server*.

Intelligent Communications Protocol (ICP): Gensym's proprietary communications protocol, which enables G2 Gateway to communicate with an external system over a network. ICP is used to share information and distribute control among one or more G2 processes, G2 Gateway bridge processes, Telewindows, and other applications. ICP is a layer built on top of the TCP/IP networking protocol, depending on which platforms you are using in your network. Note that ICP is not visible to the G2 Gateway user. See also *communications link*.

item handle: An integer identifier that G2 generates to identify a G2 item that it registers with the bridge, or that it passes to the G2 Gateway bridge through a remote procedure call declared with the as handle grammar. G2 Gateway assigns the item handle to the *gsi_registration*, *gsi_registered_item*, and *gsi_item* data structures this it uses to perform data service for a GSI variable. G2 Gateway assigns an item handle to the *gsi_registration* and *gsi_item* that it generates when it receives a G2 object that G2 passes to it through a remote procedure call declared using the as handle grammar.

J

One-Cycle Mode: The operational mode of a G2 Gateway bridge process. When the bridge is running in one-cycle mode, the function *gsi_run_loop()* is called, receives control, performs any outstanding tasks while it has control, and returns control to the user-written event loop in the bridge. This loop must make periodic calls to *gsi_run_loop()* to process network events.

One-cycle mode is the better mode for bridges designed to respond to network activity on connections to external systems, rather than to poll the external systems actively. You can use the API function *gsi_watch_fd()* to designate the connections to external systems that the bridge watches for network activity. One-cycle mode can be selected in the *gsi_set_up()* callback using the macro *gsi_set_option(GSI_ONE_CYCLE)*.

See also *Continuous Mode* and *G2 Gateway options*.

L

listener: A G2 process that listens for a network connection to a bridge process. The G2 Gateway API function `gsi_start()` takes as input the *argc* and *argv* arguments that were passed to the `main()` function of your G2 Gateway program and uses these arguments to set up listeners as specified in the command line.

P

preprocessor macro: A macro directs the C preprocessor to substitute one string of characters for another string or replace a list with another list. For example, G2 Gateway uses the __GENSYMKR__ to enable you to force the use of Kernighan and Ritchie style function declarations, when using a compiler that supports ANSI C prototypes.

programmable logic controller (PLC): A part of the external system that sends and receives signals to factory equipment. The equipment is directly controlled or measured by the PLC. The G2 Gateway bridge process can communicate with the PLC using a data path, such as a serial line, data bus, or Ethernet connection.

R

real time: Refers to the default value for the scheduler-mode attribute in the Timing Parameters system table of G2. The scheduler-mode attribute determines how G2 schedules tasks. In real-time mode, a G2 clock-tick corresponds to one second of real time. For more information, see the *G2 Reference Manual*.

receiver function: A user-written function in your G2 Gateway user code that receives return values from a G2 procedure that your G2 Gateway bridge invokes as a remote procedure.

remote procedure call (RPC): A programming language procedure contained in the bridge is considered a remote procedure when called from G2; likewise, a G2 procedure is considered a remote procedure to G2 Gateway when called from within the bridge. Both kinds of RPC can have more than one calling argument, and can return values if called rather than started. See also *call* and *start*.

runtime option: Most of the G2 Gateway C preprocessor macros have corresponding runtime options that, at run time, you can use to select or deselect the option defined by the macro. The functions `gsi_set_option()` and `gsi_reset_option()` set and reset G2 Gateway options, using the name of an option as their single argument.

S

set: An action in G2 that sends a data value through a GSI variable and its GSI interface to the bridge process. G2 Gateway receives the request and calls the callback function `gsi_set_data()`. You code `gsi_set_data()` to set the value of

the external data point in the external system. You can optionally echo the value set in the external system back to the corresponding variable in G2 using the API function `gsi_return_values()`. See also *echo*.

skeleton file: A file named `skeleton.c`, provided with GSI. It contains an empty stub version of each G2 Gateway callback function. You complete the code of the stub functions as needed for your G2 Gateway bridge.

In order for your user code to link successfully, it must include *all* the callback functions in `skeleton.c`. If you do not intend to use a particular callback function, you must nevertheless include the stub version of that callback in your user code.

See also *callback functions* and *stub functions*.

solicited data: Data returned to GSI variables in G2 only by GSI data seeking (G2 solicits the bridge process for data). Contrast with *unsolicited data*. See also *data seeking*.

start: An action in G2 that starts a remote procedure in the bridge process, which cannot return values. It is also used to start G2 procedures. See also *call* and *remote procedure call*.

string conversion style: A character set that contains all the characters in the Unicode character set. G2 5.0 uses the Unicode character set for all strings. You can cause G2 Gateway to use the string conversion style by compiling your G2 Gateway application with the `GSI_WIDE_STRING_API` preprocessor macro set, by setting the `GSI_STRING_CHECK` runtime option, or by calling the API function `gsi_set_string_conversion_style()`.

stub functions: Empty versions of callback functions provided in the source file called `skeleton.c`. You complete the code of stub functions as needed for your G2 Gateway bridge. Copy the `skeleton.c` file to form the basis of your own source file, and then complete the code of the stub functions, as required by your application.

In order for your user code to link successfully, it must include *all* the callback functions in `skeleton.c`. If you do not intend to use a particular callback function, you must nevertheless include the stub version of that callback in your user code. See also *callback functions* and *skeleton file*.

T

TCP/IP: A transportation level protocol used in the communications link between a G2 process and a bridge process. This protocol is available on each of the supported platforms and is the default protocol for all platforms. See also *communications link*.

timestamp: The collection time of a data value. The G2 Gateway data structure `gsi_item` contains a component `history times`, which is used to store timestamps. The history-keeping-spec attribute of a GSI variable indicates

whether to keep a history of values for the variable. If history keeping is enabled for a variable, timestamp information is returned to the variable along with each new data value, and G2 stores the timestamp information in the history for that variable.

U

unsolicited data: Data returned to a variable in G2 by the bridge process without G2 making a prior request for the data. Unsolicited data is always returned to G2 from within *gsi_g2_poll()*. The poll-external-system-for-data attribute of the GSI interface used by the variable must be set to yes for *gsi_g2_poll()* to be called once every G2 cycle. See also *solicited data*.

user code: The portion of the bridge process that is coded by you the G2 Gateway user, and is specific to your application. See also *callback functions* and *stub functions*.

user data: To store application-specific information on the objects that your application registers with G2 Gateway, you can have your G2 Gateway user code associate data with the following data structures:

- *gsi_registration*
- *gsi_registered_item*
- *gsi_item*
- *gsi_attr*
- *gsi_symbol*

Index

A

header file *gsi_main.h required by* remote procedures and that allocate a *gsi_symbol* data structure

B

C

D

E

F

G

H

I

P

R

S

T

U

V

W