

G2 CORBALink

User's Guide

Version 2020



G2 CORBALink User's Guide, Version 2020

June 2020

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

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

Copyright © 1985-2020 Gensym Corporation

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

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

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

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

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

Copyright © 1998-2002 Daniel Veillard. All Rights Reserved.

SCOR® is a registered trademark of PRTM.

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

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

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

Ignite Technologies, Inc.
401 Congress Ave., Suite 2650
Austin, TX 78701 USA
Telephone: +1-800-248-0027
Email: success@ignitetech.com

Part Number: DOC066-1200

Contents

	Preface	vii
	About this Guide	vii
	Version Information	vii
	Audience	vii
	A Note About the API	viii
	Conventions	viii
	Related Documentation	x
	Customer Support Services	xii
Chapter 1	Introduction	1
	Introduction	1
	Features of G2 CORBALink	2
	Common CORBA Terms	3
	Building an Application	4
Chapter 2	Getting Started	5
	Introduction	5
	Installing G2 CORBALink on HP-UX	5
	Using G2 CORBALink	6
	Merging G2 CORBALink into G2	6
	Making the G2 CORBALink Modules Required	7
	Starting the ORB	8
Chapter 3	Compiling the CORBA IDL	9
	Introduction	9
	The grid.idl File	10
	IDL Preprocessor	10
	The G2 CORBALink IDL Compiler Workspace	10

	Creating a g2orb-orb Object	12
	Specifying the Gsi-Connection-Configuration Attribute	13
	Setting Up the CORBA IDL File	14
	Compiling the IDL	15
	Updating IDL	16
Chapter 4	Building a CORBA Server Application	19
	Introduction	19
	Registering Objects	19
	Exceptions	20
	Return Values	20
	Server Classes	20
Chapter 5	Building a CORBA Client Application	23
	Introduction	23
	Specifying an Object Location	24
	Co-Residence of Servers and Clients	24
	Exceptions	24
Chapter 6	CORBA Objects and Data Structures	27
	Introduction	27
	Proxy Class	29
	Server Class	30
	Interface Class	31
	g2orb-iobject	31
	g2orb-interface-parms	31
	g2orb-orb	32
	g2orb-file	36
Chapter 7	Advanced Topics	37
	Introduction	37
	Smart Proxies	37
	Object Loaders and Locators	38

Private Objects 39

Chapter 8 API Reference 41

Introduction 42

Initialization 43

g2idl-compile 44

g2orb-download-ir 45

g2orb-init-bridge 46

g2orb-init-orb 47

Object Registration 48

g2orb-deregister 49

g2orb-register 50

g2orb-register-objects 51

g2orb-register-private 52

g2orb-release 53

Exception Management 54

g2orb-throw-exception 55

Object Reference Manipulation 56

g2orb-decode-user-ref 57

g2orb-object-to-string 58

g2orb-objref-to-obj 59

g2orb-obj-to-objref 61

g2orb-ping 62

g2orb-string-to-object 63

Chapter 9 Language Mapping 65

Introduction 65

IDL Primitive Types 66

Representing Long Long Data Types 67

Interfaces 70

Operations 72

Attributes 73

Object References 73

IDL Type to G2 Language Mapping 74

Enumerated Types 74

Structures 74

Sequences 74

Arrays 74

Unions 74

Any 75

Chapter 10 Sample G2 CORBALink Application 77

Introduction 77

Starting G2 CORBALink 78

Merging the g2idl KB into the Application 78

Setting Up and Compiling the IDL 79

Creating a Server Object 80

Creating a Proxy Object 82

Creating a Client Procedure 82

Starting Your G2 CORBALink Application 83

Appendix A Interfacing G2 CORBALink with Other CORBA Servers 85

Introduction 85

Iona's Orbix 87

Index 89

Preface

Describes this document and the conventions that it uses.

About this Guide	vii
Version Information	vii
Audience	vii
A Note About the API	viii
Conventions	viii
Related Documentation	x
Customer Support Services	xi



About this Guide

This guide describes the basic features and functions of G2 CORBALink and explains how to build CORBA-based applications, using G2 and G2 CORBALink.

Version Information

You must run G2 Gateway with the same version of G2 or with a higher version. You cannot run G2 Gateway with an older version of G2.

Audience

This guide is intended for developers of G2 CORBALink bridge applications. It provides the needed information to interface G2 to a CORBA-based system, using G2 CORBALink. It assumes that the reader is familiar with G2 and CORBA technology.

A Note About the API

The G2 CORBALink 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 G2 CORBALink release that includes them.

Therefore, it is essential that you use G2 CORBALink 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 CORBALink may change, or in the present, because the code may not correctly manage the internal operations of G2 CORBALink.

If G2 CORBALink does not seem to provide the capabilities that you need, contact Gensym Customer Support as described under [Customer Support Services](#) for further information.

Conventions

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

Typographic

Convention Examples	Description
g2-window, g2-window-1, ws-top-level, sys-mod	User-defined and system-defined G2 class names, instance names, workspace names, and module names
history-keeping-spec, temperature	User-defined and system-defined G2 attribute names
true, 1.234, ok, "Burlington, MA"	G2 attribute values and values specified or viewed through dialogs
Main Menu > Start KB Workspace > New Object create subworkspace Start Procedure	G2 menu choices and button labels
conclude that the x of y ...	Text of G2 procedures, methods, functions, formulas, and expressions

Convention Examples	Description
<i>new-argument</i>	User-specified values in syntax descriptions
<u><i>text-string</i></u>	Return values of G2 procedures and methods in syntax descriptions
File Name, OK, Apply, Cancel, General, Edit Scroll Area	GUIDE and native dialog fields, button labels, tabs, and titles
File > Save	GMS and native menu choices
Properties	
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/UIIL 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 and by email.

To obtain customer support online:

➔ Access Ignite Support Portal at <https://support.ignitetechnology.com>.

You will be asked to log in to an existing account or create a new account if necessary. Ignite Support Portal 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 or email:

➔ Use the following numbers and addresses:

United States Toll-Free +1-855-453-8174

United States Toll +1-512-861-2859

Email support@ignitetechnology.com

Introduction

Describes the features and functions of G2 CORBALink.

Introduction	1
Features of G2 CORBALink	2
Common CORBA Terms	3
Building an Application	4



Introduction

CORBA, an acronym for Common Object Request Broker Architecture, is an industry standard for a distributed object architecture. It is designed to be language, platform, and location independent.

Using CORBA, an application developer may access objects, including attributes and methods that do not reside in the same address space. Different objects of an object-oriented application may reside in different physical locations or may be implemented, using different languages. CORBA extends the client/server paradigm by making each CORBA object a server.

CORBA allows object references to be freely passed as arguments in CORBA operation requests. Object references are like pointers in a language, such as C++, but are language independent and may cross address spaces from one system to another.

To use CORBA, you need an Object Request Broker (ORB) for the language in which you are working. Object Request Brokers are available for all of the major object-oriented languages, including C++, Smalltalk, Java, and G2. G2 CORBALink is the ORB for G2.

CORBA separates the concept of an object interface from the implementation. You can specify the ways that a remote client may access an object, using a language independent representation called Interface Description Language (IDL). Once the IDL is specified, it may be compiled into each target language, using an IDL compiler. G2 CORBALink provides an IDL compiler for G2. The G2 Language mapping is described later in this document.

For a full description of IDL and CORBA, see this Web site:

<http://www.omg.org>

Features of G2 CORBALink

G2 CORBALink has the following features:

- Compatible with other CORBA 2.0 Object Request Brokers using the Internet Inter-ORB Protocol (IIOP).
- Compatible with G2 Version 5.0 Rev. 3 and later.
- Any number of requests may be outstanding either as a Client or as a Server and the requests may return out of order.
- Objects can reside anywhere. Clients and servers can be co-resident within the same process.
- Object Loader support. Object state information may be kept in a persistent storage such as a database and brought into G2 as needed.
- Object Locator support. During an object's life, it may migrate from location to location.

Common CORBA Terms

The following terms are used in this document but may not be familiar to a G2 developer. To prevent confusion, they are defined here:

Term	Description
operation	An operation is similar to a method. In fact, in G2 and most other object-oriented languages, a CORBA operation is a method.
interface	An object is a CORBA object if it implements a CORBA interface. This means that it provides a way to execute each operation specified by the interface. The interface is just a set of operations that may be invoked. An object may implement more than one interface at a time, and interfaces themselves may be combined through inheritance, including multiple-inheritance.
operation request	A client can request that a server execute an operation. This is known as an operation request. An operation request is invoked just by calling a method on a CORBA proxy object.
ORB	This stands for Object Request Broker. This is the piece of software that redirects the operation request from a client to a server. ORBs can talk to each other using IIOP
IIOP	Internet Inter Orb Protocol. This is a simple protocol based on TCP/IP that specifies how requests are made from one ORB to another. G2 CORBALink generates and interprets IIOP messages.

Term	Description
implementation object	A G2 object representing a remote CORBA object, implemented at some location, using some tool. An instance of an implementation object in G2 is a normal G2 object, with normal G2 methods. The methods control the behavior of the object. Remote CORBA clients can invoke the methods of implementation objects, which thus act as servers for the CORBA clients.
proxy object	A G2 object representing a remote CORBA object. Calling a method on a CORBA proxy object results in the operation request being redirected across the network to the implementation object where it will be executed.

Building an Application

To build a CORBA application with G2:

- 1 Create one or more CORBA IDL files to describe the CORBA interfaces needed by the application.
For a description of the default IDL file provided by G2 CORBALink, see [The grid.idl File](#).
- 2 Compile the CORBA IDL files into G2 object definitions and methods, using the G2 CORBALink compiler.
For instructions on setting up and compiling the CORBA IDL, see [Compiling the CORBA IDL](#).
- 3 Write the client and server application logic.
For information related to client applications, see [Building a CORBA Client Application](#). For information related to server applications, see [Building a CORBA Server Application](#).

Getting Started

Describes how to start the ORB executable and create and configure the g2orb-orb object.

Introduction 5

Installing G2 CORBALink on HP-UX 5

Using G2 CORBALink 6

Starting the ORB 8



Introduction

G2 CORBALink provides a setup program that copies the G2 CORBALink file from the distribution media to your local files system. The installed files include examples in C++, Java, and G2, as well as a number of example KBs.

Installing G2 CORBALink on HP-UX

The `libstdc++.sl` shared library, which is required to run G2 CORBALink, is not included as part of HP-UX 11 or AIX 5L V5.2. It is installed with the GNU C++ compiler. The GNU C++ compiler is in the public domain. You can find sites where it available for downloading by searching for "GNU C++ HPUX" or "GNU C++ AIX" with a search engine such as Google. At the time of this writing, it is available for HP-UX at: <http://hpux.cs.utah.edu/hppd/hpux/Gnu/>.

Using G2 CORBALink

G2 CORBALink includes two G2 modules. They are:

Module	Description
g2orb	Provides runtime support for G2 CORBALink.
g2idl	Contains the IDL compiler. This module is only needed during development.

You integrate G2 CORBALink by:

- Merging the developer's knowledge base (KB) module, named `g2idl`, into your G2 application. This module includes `g2orb`.
- Making the `g2idl` and `g2orb` modules required modules of your application.

Merging G2 CORBALink into G2

A G2 CORBALink application must be modularized. The filename of the G2 CORBALink development module is `g2idl.kb`. This file should be merged with your user application.

To merge G2 CORBALink into your application:

- 1 Pause or reset your KB.
- 2 Choose Merge KB from the Main Menu to display the Load KB workspace.
The merge in this KB option is enabled on the workspace.
- 3 Specify the location of the `g2idl.kb` file and click End.

This KB is located in the `kbs` subdirectory in the `corbalink`, which is located in your G2 product directory.

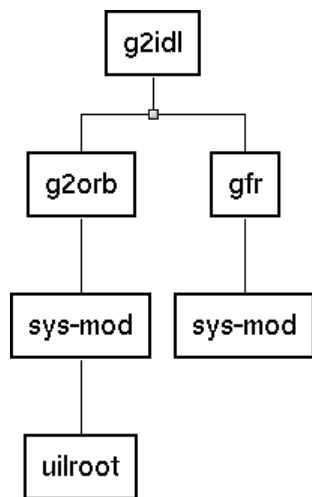
Tip When merging G2 CORBALink, let G2 resolve conflicts by enabling the `automatically resolve conflicts` option.

G2 CORBALink Modules

When you merge the `g2idl.kb`, its required modules are automatically loaded into G2. The following table describes these modules:

Module	File Name	Contents
<code>g2idl</code>	<code>g2idl.kb</code>	Definitions, API support and compiler for G2 CORBALink. This module may be deleted from the deployed application.
<code>g2orb</code>	<code>g2orb.kb</code>	Definitions and API support for the G2 CORBALink bridge.
<code>gfr</code>	<code>gfr.kb</code>	Definitions and API support for the G2 Foundation Resources (GFR) utility. This module is required only by <code>g2idl</code> .
<code>sys-mod</code>	<code>sys-mod.kb</code>	The library of G2 system procedures.
<code>uilroot</code>	<code>uilroot.kb</code>	Definitions and API support for navigation buttons.

This is the module hierarchy of `g2idl`:



Making the G2 CORBALink Modules Required

When you merge G2 CORBALink into your KB, it is not a required module unless it is specified in the Module Information table of your KB. Gensym recommends

that you make `g2idl` and `g2orb` required modules of the top-level module of your user application.

To make G2 CORBALink a required module:

- 1 Choose Main Menu > System Tables > Module Information.
- 2 Specify `g2orb` and `g2idl` as a directly required modules of the top-level module of your KB.

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

Starting the ORB

The `g2-corba` executable implements the CORBA IIOP protocol and allows G2 to communicate with third-party Object Request Brokers.

You specify two port numbers: one used by G2 to communicate with the ORB and the other used by the ORB to communicate with IIOP clients.

To start the ORB:

➔ Execute the following command:

```
g2-corba gsi-port iiop-port
```

where:

gsi-port is the TCP/IP port through which G2 communicates with the ORB.

iiop-port is the port through which the ORB communicates with IIOP clients.

For example:

```
g2-corba 22041 5998
```

Note The *gsi-port* number must match the `tcp-ip` port number specified in the `Gsi-connection-configuration` attribute of the `g2orb-orb` object in your application.

Once the ORB is running, you are ready to set up and compile the IDL for your application. For instructions, see [Compiling the CORBA IDL](#).

Compiling the CORBA IDL

Describes how to compile CORBA IDL, using the G2 CORBALink compiler.

Introduction	9
The grid.idl File	10
IDL Preprocessor	10
The G2 CORBALink IDL Compiler Workspace	10
Creating a g2orb-orb Object	12
Setting Up the CORBA IDL File	14
Compiling the IDL	15
Updating IDL	16



Introduction

Before you can write a CORBA client or server application, you must compile the CORBA Interface Description Language (IDL) into G2 object definitions and methods. CORBA IDL files describe the CORBA interfaces needed by your application. The G2 CORBALink compiler needs to know the location of these files.

The grid.idl File

G2 CORBALink provides an example IDL file, `grid.idl`, which defines an interface named `grid` that has two read-only attributes, `width` and `height`, and an operation named `fetch()`.

```
interface grid {
    readonly attribute short height; // height of the grid
    readonly attribute short width; // width of the grid

    // IDL operation

    // return element [n,m] of the grid;
    long fetch(in short n, in short m);
};
```

The interfaces specified in IDL depend on the requirements of the application, which can be much more complex than the example shown here.

IDL Preprocessor

Preprocessors are mostly used with IDL to refer to definitions found in other IDL files. The preprocessor directives found in most CORBA IDL are `#include` and “Include guards” to ensure that files are only included once.

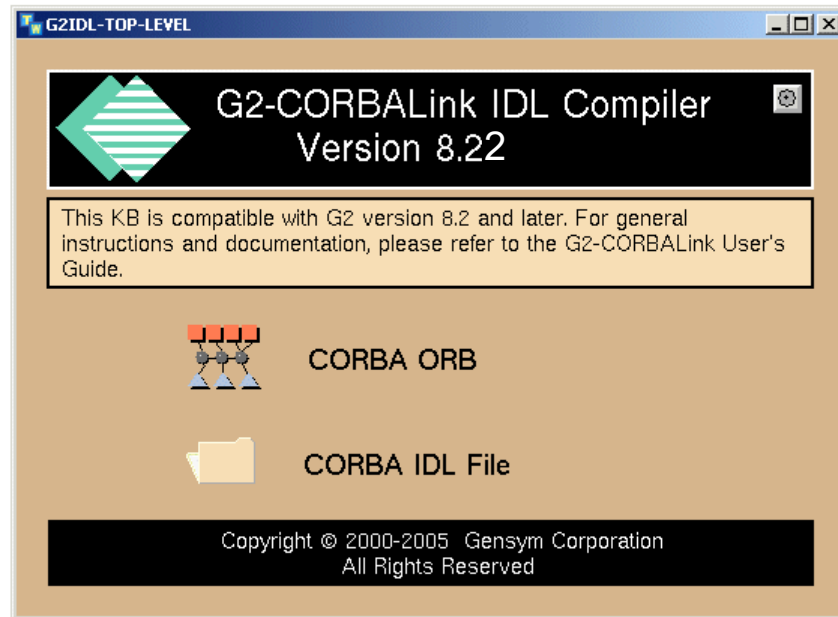
The G2 CORBALink compiler provides limited preprocessor support. G2 CORBALink honors CORBA `#include` directives and automatically ensures that files are only included once. If other preprocessor directives are needed, you should first pass the IDL file through an external preprocessor and then compile the pre-processed IDL file with G2 CORBALink.

The G2 CORBALink IDL Compiler Workspace



The `g2idl` module contains the G2 CORBALink IDL Compiler Workspace. This workspace includes a palette of objects for setting up and compiling CORBA Interface Description Language (IDL) into G2 object definitions and methods for your application.

To display the IDL compiler workspace:

→ Choose Main Menu > Get Workspace > g2-idl-top-level.



The objects on this workspace are:

Object	Description
	<p>The CORBA ORB icon represents a <code>g2orb-orb</code> object. An Instance of this object creates a link to the CORBA interface. The attributes of <code>g2orb-orb</code> contain values that control how the IDL is compiled, as well as the run time characteristics of the ORB, such as time-out values.</p>
	<p>The IDL FILE icon represents a <code>g2orb-file</code> object. An instance of this object specifies the location of a CORBA IDL file that contains IDL definitions to be compiled by the G2 CORBALink compiler. The IDL file has a connection stub, which is used to connect it to a <code>g2orb-orb</code> object.</p>

The following steps summarize setting up and compiling the CORBA IDL for your application:

Steps	For details, see...
1 Clone and configure a g2orb-orb object.	
2 Clone and configure one or more g2orb-file objects.	
3 Connect IDL file objects to the ORB object and compile the IDL.	

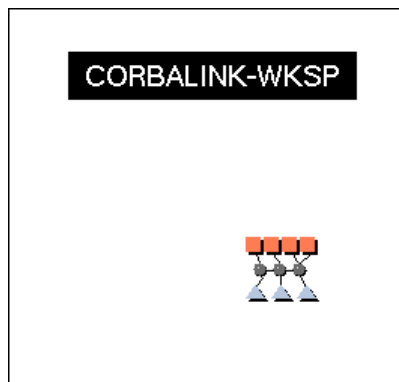
Creating a g2orb-orb Object

Before you can use the ORB, you must create and configure a g2orb-orb object in your KB.

To create a g2orb-orb object:

- 1 Create a new workspace to contain the objects needed to support your application.
- 2 Choose Main Menu > Get Workspace > g2idl-top-level to display the G2 CORBALink IDL Compiler workspace.
- 3 Clone a CORBA ORB object from the palette onto the workspace.

For example:



- 4 Display the attribute table of the Orb and enter values for the following attributes:

Attribute	Value
names	<p>A unique symbolic name for this Orb.</p> <p>Note: The Orb must be named or G2 will not activate a connection.</p>
gsi-connection-configuration	<p>The network protocol that G2 uses to communicate with the G2 CORBALink bridge, specified as:</p> <p><i>tcp-ip host "hostname" port-number tcp-ip-port-number</i></p> <p>The <i>tcp-ip-port-number</i> must match the first port number that was specified to start the g2-corba executable.</p> <p>For more information, see Specifying the Gsi-Connection-Configuration Attribute.</p>

Specifying the Gsi-Connection-Configuration Attribute

In the `gsi-connection-configuration` attribute of the `g2orb-orb` object, you must specify the communications protocol and location of the G2 CORBALink process. This attribute contains an expression that identifies a running G2 CORBALink. You can specify this attribute in the following format:

```
tcp-ip host "hostname" port-number tcp-ip-port-number
```

where:

hostname is the name of the machine on which the g2-corba executable is running.

tcp-ip-port-number is the port number on which the g2-corba executable is running.

For example, the following expression specifies a G2 CORBALink process running on port number 22041 of the local computer (the computer on which G2 is running) and to which this G2 connects through a network connection over the TCP/IP protocol:

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

In order for G2 to establish a connection to the specified G2 CORBALink, the `gsi-connection-configuration` attribute must match:

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

Note Be sure to use the ICP TCP/IP port number of the ORB, which is the first argument in the command line.

Setting Up the CORBA IDL File

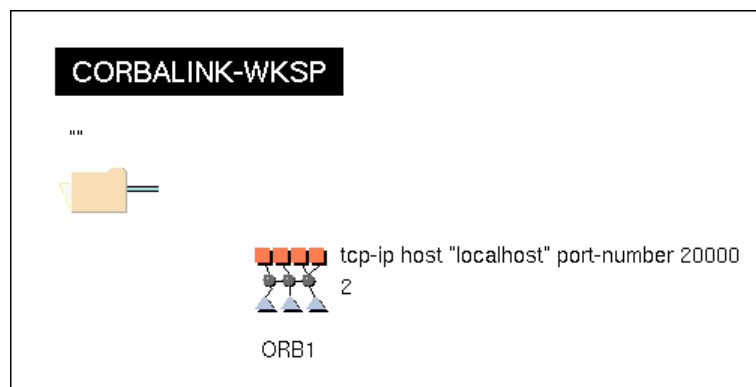
The IDL file defines the interfaces and operations for your application that are used by G2 CORBALink. You create the IDL file of definitions specific to your application.

G2 CORBALink provides a default IDL file, named `grid.idl`. For a description of this file, see [The `grid.idl` File](#).

To set up a CORBA IDL file:

- 1 Clone an IDL file object from the palette onto the workspace.

For example:



- 2 Display the attribute table of the IDL file object and specify the value of the `G2orb-file-path` attribute as a string containing the full pathname of the IDL file you want to compile.

For example:

a g2orb-file	
Notes	OK
Names	none
G2orb file path	"c:\apps\corba new\grid.idl"
G2orb time of last change	0.0

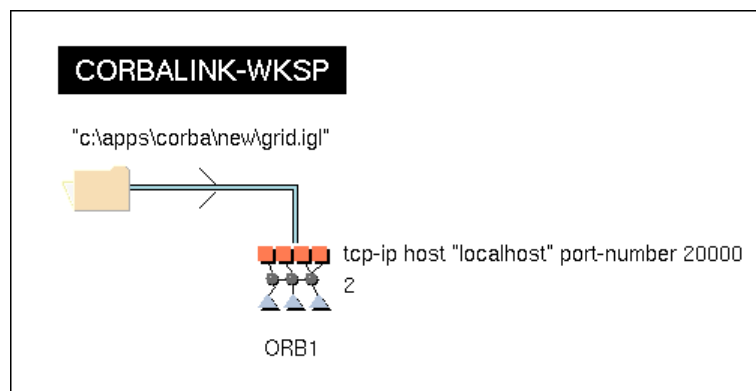
Compiling the IDL

Before you compile the IDL, you must connect the IDL file object to the ORB object on your workspace.

To connect the IDL file to the ORB:

- 1 Click the connection stub attached to the IDL file.
When you move the mouse, the connection stub follows the cursor.
- 2 Position the stub directly over the ORB object and click to complete the connection between the two objects:

For example:



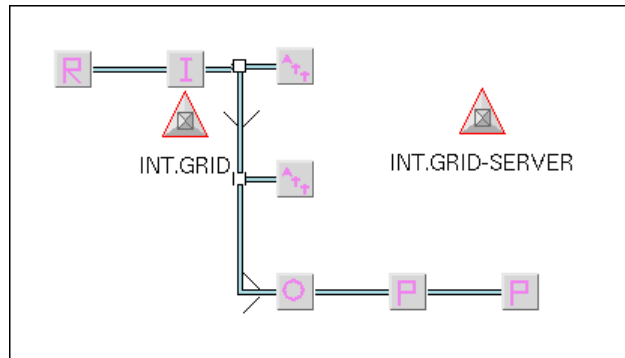
To compile the IDL:

- ➔ Choose compile idl from the menu of the Orb object.

The G2 CORBALink compiler begins processing the IDL and creates sample implementation objects and methods.

Log book messages may appear during the compilation process, because the compiler sometimes creates references to definitions before the actual definitions are created. You can ignore these messages.

When the compilation is complete, a workspace appears, containing the results. For example:



This workspace contains:

- The objects that describe the interfaces from the IDL source file, which are downloaded into the CORBA ORB, enabling it to send and receive requests.
- G2 objects that map the interfaces defined in the CORBA IDL into the G2 object system. Instances of these objects are created to represent CORBA objects and proxies.

When compilation is complete, a working CORBA server will have been created. However, it will not do much until you provide the application. For information see [Building a CORBA Server Application](#).

After the IDL is compiled, you can remove the `g2idl` module from the list of required modules of your KB.

Updating IDL

In an ideal world, all interfaces would be specified at the beginning of the project, carved in stone and never change. Alas, this is not the world that we live in. Interfaces do sometimes change.

The CORBA IDL may be recompiled. Existing object definitions will be updated and the changes downloaded into the CORBA interface. G2 does not need to be reset. When the IDL is recompiled, neither client or server object instances are lost, nor is the application server or client logic removed.

Since you may modify the implementation class and methods, the compiler will not modify them once they are created. If these definitions are deleted, the compiler will re-create default versions.

Interfaces that are no longer defined by the IDL are automatically removed, if there are no instances of these class definitions. If instances exist, then they are placed in an area by themselves, away from the other class definitions. When this occurs, a message appears on the Message Board.

Building a CORBA Server Application

Describes how to build a CORBA server application.

Introduction	19
Registering Objects	19
Exceptions	20
Return Values	20
Server Classes	20



Introduction

Most of the work involved with creating a CORBA server is done for you by G2 CORBALink. The G2 CORBALink compiler automatically generates default implementation classes and methods for the CORBA interface and operation.

The default methods are empty, but contain an `inform the operator` statement that will show up on the G2 Message Board. Clicking on the message sends you directly to the operation method.

You must provide the application logic for the operation method and modify the method as needed. Be careful not to modify the signature of the method.

Registering Objects

Before requests may be invoked on an object, the object must be registered with the CORBA ORB. This may be done automatically at startup, or done programmatically with the [g2orb-register](#) method.

Before registering the object, the `g2orb-object-name` attribute of the object must be assigned a unique value. The value must be unique over all objects that share the same G2 CORBALink interface process.

One G2 CORBALink interface process may be used simultaneously with more than one G2. G2 CORBALink tracks the location of each registered object and ensures that no two objects have the same object name.

Exceptions

The server application may throw a user exception by creating an instance of the exception object and calling the [g2orb-throw-exception](#) procedure. The exception object definition is automatically created by the G2 CORBALink compiler.

Return Values

To prevent memory leaks in CORBA servers, all transient items that are returned to clients are automatically deleted. Permanent items are not deleted.

Server Classes

A Server or Implementation class implements the behavior of a object that may be accessed either locally or from a remote system. Methods on a server object are executed when a remote client calls the corresponding method on a object proxy. Methods may have parameters that are input, output, or both in and outputs. Attributes defined in the IDL are also included in server objects.

The compiler generates a default server class. The name of the default server class is the name of the interface with the suffix `-server`. This class includes each IDL attribute and method. The methods are “stubs.” Each method includes:

- The correct signature.
- An “inform the operator” statement.
- A comment that reads “TODO: Put your logic here”.

The developer may modify these methods as needed to implement the logic of the application.

If the developer needs to add additional attributes to the server object or inherit from other classes, then a new server class should be created that subclasses from the class generated by the compiler. This prevents the class from being overwritten the next time the IDL compiler is run.

It is safe to modify the server methods and re-run the IDL compiler. Delete a server method to allow the IDL compiler to re-create the default method.

Hint An easy way to find the default server method is to click on the logbook message generated by the default server method and choose **go to message origin** from the menu.

Building a CORBA Client Application

Describes how to build a CORBA client application.

Introduction	23
Specifying an Object Location	24
Co-Residence of Servers and Clients	24
Exceptions	24



Introduction

When you develop a G2 CORBALink application, you use the standard G2 procedure language. Proxy objects are created when needed to represent remote objects. You may treat these proxy objects as normal objects.

Proxy objects are automatically created when a remote object reference enters G2 as an operation parameter. If an object reference for a local object enters G2, G2 automatically returns a normal G2 object reference for the actual object rather than create an object proxy.

If a proxy object is created, then a proxy of the proper type will be created. For example, an application has two interfaces, one called `Account` and one called `currentAccount`, and `currentAccount` is a subclass of `Account`. If an operation returns an object reference of type `Account` and a `currentAccount` is actually returned then, a `currentAccount` proxy will be created and returned as the result of the operation. Because a `currentAccount` is an `Account`, G2 considers this to be a valid return type.

When a client application is finished with a proxy, the [g2orb-release](#) method should be called to discard the proxy.

Specifying an Object Location

Object locations can be specified by either a Uniform Resource Locator (URL) format or an Inter-operable Object Reference (IOR) generated by a CORBA ORB. URLs are a simple way to specify the location of an object. IORs may always be used, but are not as convenient.

Either a URL or IOR may be entered into the `g2-orb-user-ref` attribute of a proxy object. For example, a URL format object location can be specified in the `g2-orb-user-ref` attribute of a proxy object as:

```
"\ftp.gensym.com:1999\my object"
```

This refers to an object managed by an Object Request Broker with a IIOP port at TCP/IP port number 1999 on the system named `ftp.gensym.com`. The object has an object key represented by the string "my object".

Not all CORBA ORBs use a string for object keys. However, G2 CORBALink and several other popular ORBs do.

After updating the `g2-orb-user-ref` attribute, either the [g2orb-decode-user-ref](#) method or the `parse user ref` menu choice must be used before the change is recognized by G2 CORBALink.

In addition, Object References can be passed from system to system or stored in a CORBA Name Service.

Co-Residence of Servers and Clients

CORBA hides the location of an object from the application developer. In addition, G2 CORBALink supports co-residence. If the implementation of a server object happens to reside in the same G2 as a client, then they are considered to be co-resident. Normal G2 method calls are used for co-resident operation requests and the overhead of a remote request over the network is avoided.

This is accomplished by using the actual implementation object rather than an object proxy for object references that refer to local objects.

Exceptions

If a method call on a remote CORBA object returns or throws an exception, G2 CORBALink converts this exception to a G2-error object. The G2 application may catch the error object with a "on error" statement, or it may allow the default error handler to display a message on the operator log book.

There are two types of exceptions:

- System exceptions
- User exceptions

System Exceptions are defined in the CORBA standard and are generated by G2 CORBALink or the remote ORB.

User Exceptions are defined in IDL. The G2 CORBALink compiler will generate a G2 error class definition for each User Exception defined in the IDL.

CORBA servers written in G2 may return a User Exception to a caller by creating an instance of a User Exception object and calling the [g2orb-throw-exception](#) method.

CORBA Objects and Data Structures

Describes G2 CORBALink objects and data structures.

Introduction	27
Proxy Class	29
Server Class	30
Interface Class	31
g2orb-iobject	31
g2orb-interface-parms	31
g2orb-orb	32
g2orb-file	36



Introduction

G2 CORBALink provides predefined classes that G2 can use to represent remote objects and encapsulate information that G2 needs to communicate with remote clients and servers. The sections in this chapter describe each of the classes and their attributes.

The following table summarizes these classes:

Class	Description
proxy	The IDL Compiler creates a proxy class for each interface defined in IDL. Methods are defined for each IDL operation. Instances of these objects are proxies to remote server objects.
server	The IDL Compiler creates a server class for each interface defined in IDL. The compiler generates default methods for each IDL operation, which you may override.
interface	The IDL compiler creates a interface class for each interface defined in IDL. This class is abstract and no instances of this class may be created. However, both the proxy class and the server class are derived from it.
g2orb-iobject	The G2 CORBALink compiler creates a set of g2orb-iobject object instances to represent application interfaces.
g2orb-interface-parms	An instance of a g2orb-interface-parms object is created by the G2 CORBALink compiler for each interface defined by the IDL. This object holds information about the interface.
g2orb-orb	The g2orb-orb object represents a CORBA ORB. It holds configuration parameters for the Object Request Broker.
g2orb-file	The g2orb-file object represents a CORBA IDL file. It hold the location of the IDL file that is compiled for your CORBA application.

Proxy Class



G2 CORBALink client applications use instances of the `g2orb-proxy` class to represent remote objects. The proxy class definition is automatically created by the CORBA compiler. Methods on this class are created for each IDL operation. These methods redirect the operation over the network to the CORBA object.

You may create a subclass of the proxy class created by the compiler and override the logic used to redirect operation calls.

You may specify the location of the object that will be the target of operation requests as either a CORBA IOR or a URL.

The name of the proxy class is the interface name with `-proxy` appended.

This is the attribute of `g2orb-proxy`:

Attribute	Description
g2orb-user-ref	Specifies an object location, using either an IOR or a URL. For more information, see Specifying an Object Location .

Allowable values: Text string

These are the user menu choices for instances of `g2orb-proxy`:

Menu Choice	Description
deregister	Calls <code>g2orb-deregister</code> to unregister the proxy object with the ORB.
register	Calls <code>g2orb-register</code> to register the proxy object with the ORB.
ping	Calls <code>g2orb-ping</code> to verify the existence of the remote object.
show-ior	Generates an Interoperable Object Reference (IOR) for the object. This may be used in a remote proxy or by another CORBA ORB.

Server Class



The IDL compiler creates a server class for each interface defined in IDL. Instances of this class are “servers” that may be accessed by remote object proxies.

The IDL compiler creates default methods for each IDL operation. These methods should be modified as needed. You can also create subclasses to add additional non-IDL attributes.

The name of the Server Class is the name of the interface class with **-server** appended.

This is the attribute of a server object:

Attribute	Description
g2orb-object-name	Name of the server object, which must be unique.
<i>Allowable values:</i>	Text string
<i>Notes</i>	This text value is referred to by the g2orb-user-ref attribute of the proxy object.

These are the user menu choices of a Server object:

Menu Choice	Description
deregister	Calls g2orb-deregister to unregister the object with the ORB. This is not normally needed.
register	Calls g2orb-register to register the object with the ORB.
show-ior	Generates an Interoperable Object Reference (IOR) for the object. This may be used in a remote proxy or by another CORBA ORB.

Interface Class

The Interface class is an abstract class that is generated by the IDL compiler for each interface defined in IDL. All objects, either clients or servers that implement the given IDL interface are subclasses of this class. User applications may use the interface class in method signatures to refer to an object that provides the needed interface without caring if the object is local or remote.

The name of the interface class is derived from the name of the `g2orb-orb` object and the interface name.

g2orb-iobject

The G2 CORBALink compiler creates a set of `g2orb-iobject` object instances to represent application interfaces. This data structure is then compiled into G2 language definitions and is also downloaded into the Object Request Broker. This data structure is based on the Interface Repository data structure defined in the CORBA specification. However, G2 CORBALink 1.0 does not expose the interface repository to user applications.

g2orb-interface-parms



The G2 CORBALink compiler creates an instance of a `g2orb-interface-parms` object for each interface defined by the IDL. The attributes of this object contain information about the compiler-generated interface.

These are the attributes of a `g2orb-interface-parms` object:

Attribute	Description
g2orb-interface	Name of the interface definition class.
<i>Allowable values:</i>	Symbolic class name
g2orb-proxy-class	Name of the proxy class associated with this interface.
	You may define a subclass of this class and enter that class name here.
<i>Allowable values:</i>	Symbolic class name

Attribute	Description
g2orb-abs-name	The Interface Repository ID as specified by CORBA. It includes the interface name and version.
<i>Allowable values:</i>	Text string
g2orb-repository-id	Identification information used internally by G2CORBA-Link.
<i>Notes</i>	Do not modify this value.

g2orb-orb



The g2orb-orb object represents a CORBA ORB. It holds configuration parameters for the Object Request Broker.

These are the attributes of a g2orb-orb object:

Attribute	Description
names	Unique name of the g2orb-orb object.
<i>Allowable values:</i>	Symbolic object name
<i>Notes:</i>	The Orb must be named or G2 will not activate a connection.
gsi-connection-configuration	Specifies the network protocol that G2 uses to communicate with the bridge. The CORBA interface can reside on a different system. For more information, see Specifying the Gsi-Connection-Configuration Attribute .
<i>Allowable values:</i>	tcp-ip host "hostname" port-number tcp-ip-port-number
<i>Default value:</i>	none

Attribute	Description
gsi-interface-status	A value displayed by the bridge to indicate the current status of the connection between the bridge and the G2 application.
<i>Allowable values:</i>	<p>2 (OK) - The connection has been made and is active.</p> <p>1 (Initializing) - The external system is being initialized. When G2 receives this code, it refrains from sending messages to the bridge until it receives the OK code (2).</p> <p>0 (Inactive) - The connection is disabled or inactive.</p> <p>-1 (Timeout) - The connection has been timed out because G2 has not heard from the bridge within the time specified by the <code>Interface-timeout-period</code> attribute.</p> <p>-2 (Error) - An error has occurred, and the connection is broken.</p>
<i>Notes:</i>	This read-only attribute is automatically updated by G2 after each transmission between G2 and G2 CORBALink.
g2orb-include-search-path	A list of the directories that the pre-processor should search when it encounters a <code>#include file</code> command.
<i>Allowable values:</i>	A text string containing any valid directory or directories, or a null list (""). Multiple directories must be separated by commas.

Attribute	Description
g2orb-idl-connection-timeout	<p>A number of seconds after which the ORB drops IIOP connections on which there has been no activity.</p> <p>The ORB automatically reestablishes a dropped connection when it is needed for an operation request.</p>
<i>Allowable values:</i>	0 (prevents the ORB from dropping idle connections), or any other positive value
<i>Default value:</i>	300 seconds
g2orb-connect-timeout	The number of seconds the ORB waits before aborting an attempt to establish a connection.
<i>Allowable values:</i>	Any positive value
<i>Default value:</i>	15 seconds
g2orb-client-request-timeout	The number of seconds that the ORB waits before aborting a remote request before returning a system exception.
<i>Allowable values:</i>	0 (causes the ORB to wait indefinitely), or any other positive value
<i>Default value:</i>	120 seconds
g2orb-server-request-timeout	The number of seconds that the ORB waits for G2 to finish a request from a remote system before returning a system exception.
<i>Allowable values:</i>	0 (Causes the ORB to wait indefinitely) Any other positive value
<i>Default value:</i>	60 seconds

Attribute	Description
g2orb-register-objects-on-startup	If set to true, g2orb-register-objects is called by g2orb-init-orb when a connection is established with the CORBA interface. This automatically registers all CORBA objects in the KB with the interface.
<i>Allowable values:</i>	true or false
<i>Default value:</i>	true
g2orb-prefix	Specifies a prefix that the CORBA compiler prefixes to all global objects generated by the CORBA compiler to prevent name collisions.
<i>Allowable values:</i>	Any text string
<i>Default value:</i>	"INT"
g2orb-init-procedure	Specifies the procedure called when G2 establishes a connection with the CORBA interface.
<i>Allowable values:</i>	The name of a G2 procedure (symbol)
<i>Default value:</i>	g2orb-init-orb
g2orb-object-loader	The name of a user-provided procedure that handles requests on unknown objects.
<i>Allowable values:</i>	The symbolic name of any user-provided G2 procedure
<i>Default value:</i>	none
g2orb-principal	Specifies a user name when an external ORB is communicating with a secured application.
<i>Allowable values:</i>	Any text string

The following table lists the user menu choice of g2orb-orb objects:

Menu Choice	Description
compile idl	Compiles the user IDL code in the CORBA IDL file object (g2orb-file) connected to the g2orb-orb object.

g2orb-file



A g2orb-file object represents the IDL file that you compile to create your CORBA application. You create a g2orb-file object by cloning the CORBA IDL file object, which is on the G2 CORBALink IDL Compiler workspace.

This is attribute of g2orb-proxy whose value you must provide:

Attribute Name	Description
g2orb-file-path	<p>The location of the IDL file as a text string.</p> <p>The default is an empty string (“”).</p> <p>You can specify either:</p> <ul style="list-style-type: none">• The full, absolute pathname of the file, or• The filename. <p>If you specify only the filename, G2 CORBALink looks for a file with this name in the directories that you specify in the G2orb-include-search-path attribute of the g2orb-orb object.</p>

You connect this object to the g2orb-orb object, using the connection stub.

For instructions on connecting and compiling the IDL file, see [Compiling the CORBA IDL](#).

Advanced Topics

Describes advanced techniques for using G2 CORBALink.

Introduction 37

Smart Proxies 37

Object Loaders and Locators 38

Private Objects 39



Introduction

This chapter describes the following advanced topics:

- [Smart Proxies](#)
- [Object Loaders and Locators](#)
- [Private Objects](#)

Smart Proxies

For some applications, it is desirable to satisfy some operations locally rather than remotely. Improved performance can result from performing some operations locally, rather than redirecting all operation requests over the network. To do this, you can use a **smart proxy**. A smart proxy overrides the behavior of the default proxy object generated by the compiler.

To create a smart proxy:

- 1 Create a subclass of the default proxy class and override the methods needed for the application.
- 2 Include a call next method statement in the method to allow a request to continue across the network.
- 3 Indicate to G2 CORBALink that this proxy class should be used instead of the default proxy class for object references that enter G2 for this interface by updating the `g2orb-proxy-class` attribute of the `g2orb-interface-parms` object for this interface.

Object Loaders and Locators

If an operation request arrives at a CORBA ORB for an object that is unknown to it, the ORB may do one of three things:

- 1 Forward the request to a different location.
- 2 Create an object and allow the request to continue.
- 3 Refuse the request and return a system exception to the client

A CORBA server that uses the first option is known as a Locator, or a server that knows the location of other objects. It is also used to allow objects to migrate from one location to another.

The second approach calls a user defined function known as an Object Loader. The Object Loader may retrieve the state of an object from a persistent storage device, such as a database, and allow the request to continue. As a result, not every object needs to reside in memory at the same time and very large and scalable applications may be built.

You may control what is done with attempts to access unknown or unregistered objects by writing an object loader procedure and specifying the name of the object loader in the `G2orb-object-loader` attribute of the `g2orb-orb` object. If this value other than the symbol `none`, then the procedure by this name will be called for each request on an unknown object.

If more than one G2 shares a G2 CORBALink interface, then the loader for each G2 is called on a round-robin basis. This allows several G2 servers to cooperate for building large, scalable applications. With this approach, the processing load is evenly spread across all available G2 systems and the total throughput of the system may be increased by adding additional G2 application servers. In addition, if some G2 servers go off-line, the total capacity of the overall system is decreased, but the functionality of the system is not necessarily impaired.

The signature of the object loader is as follows:

```
my_object_loader_name
(object-key: text, connection-id: integer)
-> (symbol, ref: text)
```

The **object-key** is a text identifier for the unknown object that was included in the operation request. Since not all object keys are text, binary object keys are converted to Hex, if they contain any non-printable characters.

The procedure may do one of the following:

- Return the symbol **reject**. In this case G2 CORBALink rejects the request.
- Return the symbol **forward** and a new object reference. In this case G2 CORBALink forwards the request to the new object reference. The client uses the new object reference for future requests.
- Create, register a new implementation object, register it with `g2orb-register()`, and return the symbol **new-object**. In this case G2 CORBALink continues processing the operation.
- Create a new implementation object, register it with the `g2orb-register-private()` procedure, and return the symbol **private-object**. In this case, G2 CORBALink continues processing the operation, but the object is only visible over the connection that generated the request.

Private Objects

Consider the case of an application that has a large number of objects that reside in a persistent storage, such as a database. As requests come from remote systems, the objects are pulled from the database. The application may provide commit and rollback operations that will end a transaction and restore the state of the object back to the database. However, if the client program aborts without ending the transaction, the server has no way to know, other than a time-out, that the client has aborted.

If the object is registered as a private object, then it is only visible to requests that come over a given connection. If the connection is broken for any reason, the ORB will reregister all private objects for that connection and call the `g2orb-finalize()` method on the object. The default behavior of the `g2orb-finalize()` method is to delete the implementation object.

A given implementation object may be registered as private on more than one connection. In this case, the `g2orb-finalize()` method will not be called until every connection that the object is registered on ends. The application may call `g2orb-deregister` on a private object to unregister the object on all connections.

G2 CORBALink automatically closes idle connections. The length of time that a connection remains idle before a connection is closed is controlled by the `G2orb-`

idle-connection-timeout attribute of the g2orb-orb object. Setting this value to 0 will disable idle connection time-outs.

API Reference

Describes the API of G2 functions provided with G2 CORBALink.

Introduction 42

Initialization 43

- g2idl-compile 44
- g2orb-download-ir 45
- g2orb-init-bridge 46
- g2orb-init-orb 47

Object Registration 48

- g2orb-deregister 49
- g2orb-register 50
- g2orb-register-objects 51
- g2orb-register-private 52
- g2orb-release 53

Exception Management 54

- g2orb-throw-exception 55

Object Reference Manipulation 56

- g2orb-decode-user-ref 57
- g2orb-object-to-string 58
- g2orb-objref-to-obj 59
- g2orb-obj-to-objref 61
- g2orb-ping 62
- g2orb-string-to-object 63



Introduction

This chapter divides the Application Programmer's Interface (API) to G2 CORBALink into the following functional categories:

- [Initialization](#)
- [Object Registration](#)
- [Exception Management](#)
- [Object Reference Manipulation](#)

Initialization

This section describes the API methods and procedures used to programmatically compile and initialize G2 CORBALink:

- [g2idl-compile](#)
- [g2orb-download-ir](#)
- [g2orb-init-bridge](#)
- [g2orb-init-orb](#)

g2idl-compile

Compiles a user IDL file.

Synopsis

g2idl-compile

(*orb* class:g2orb-orb, *window*: class g2-window)

Argument	Description
<i>orb</i>	The g2orb-orb object that is connected to the g2orb-file object containing the IDL that you want to compile.
<i>window</i>	The G2 window used to display messages.

Description

This procedure compiles the user IDL code in the g2orb-file object connected to the specified *orb*.

The compile idl user menu choice of a g2orb-orb object calls this procedure. This procedure is not called by user code.

g2orb-download-ir

Downloads the interface definitions into the ORB.

Synopsis

g2orb-download-ir
(*object*: class g2orb-orb)

Argument	Description
<i>object</i>	The ORB into which interface definitions are downloaded.

Description

Interface definitions must be downloaded into the ORB before objects are registered, or remote operation requests are received or issued. The definitions may be updated at any time.

The default initialization routine of the ORB, `g2orb-init-orb`, calls this procedure. You do not need to call `g2orb-init-orb` explicitly, unless you write a customized initialization routine in place of `g2orb-init-orb`.

g2orb-init-bridge

Verifies the version of the interface executable and sets interface configuration options.

Synopsis

g2orb-init-bridge
(*orb*: class g2orb-orb)

Argument	Description
<i>orb</i>	The ORB being initialized.

Description

g2orb-init-bridge is called by g2orb-init.

The default initialization routine of the ORB, g2orb-init-orb, calls this procedure. It is not normally necessary to call this procedure directly from your application.

g2orb-init-orb

Initializes the CORBA ORB.

Synopsis

g2orb-init-orb
(object: class g2orb-orb)

Argument	Description
<i>object</i>	The ORB to be initialized.

Description

This is the default initialization procedure that is called whenever the g2orb-orb object establishes a connection with the ORB executable.

g2orb-init-orb calls the following procedures:

- g2orb-init-bridge
- g2orb-download-ir
- g2orb-register-objects

Optionally, you can create a custom initialization procedure to use in place of g2orb-init-orb and specify its name in the g2orb-init-procedure attribute of the g2orb-orb object.

Example

The following G2 procedure, my-init-routine, calls g2orb-init-orb and then specifies additional steps to follow after the ORB is initialized:

```
my-init-routine(G: class g2orb-orb)
  begin
    call g2orb-init-orb(G);
    { place additional steps here}
  end
```

Object Registration

This section describes the API methods and procedures used to programmatically register and unregister G2 CORBALink objects:

- [g2orb-deregister](#)
- [g2orb-register](#)
- [g2orb-register-objects](#)
- [g2orb-register-private](#)
- [g2orb-release](#)

g2orb-deregister

Unregisters either an object implementation or proxy object.

Synopsis

g2orb-deregister
(*object*: class g2orb-corba-object)

Argument	Description
<i>object</i>	The object to be unregistered.

Description

Before you delete an implementation object, call this method to unregister that *object*. If a client tries to access a deleted implementation object that has not be unregistered, the client receives a system exception.

You can also use `g2-orb-deregister` to unregister private objects on all connections. For information about private objects, see [Private Objects](#).

g2orb-register

Registers either an object implementation or proxy object.

Synopsis

g2orb-register
(*object*: class g2orb-corba-object)

Argument	Description
<i>object</i>	The object to be registered.

Description

You do not need to invoke this procedure explicitly. A proxy object is automatically registered the first time an operation is performed on the proxy object. Implementation (server) objects are automatically registered at startup time if the g2orb register objects on startup attribute of the g2orb-orb object is set to true.

g2orb-register-objects

Registers all implementation objects for a given ORB.

Synopsis

g2orb-register-objects
(*object*: class g2orb-orb)

Argument	Description
<i>object</i>	The ORB whose implementation objects are to be registered.

Description

This procedure is called by the default orb initialization routine. It is not normally necessary to include other calls to this function in your application.

If the `g2orb-register-object` on startup attribute of the `g2orb-orb` object is set to true, `g2orb-register-objects` looks at every implementation object in the KB and calls `g2orb-register` to register each object for this ORB.

g2orb-register-private

Registers a private implementation object to be visible to a client on a given connection.

Synopsis

g2orb-register-private

(*object*: class g2orb-server, *connection-id*: integer)

Argument	Description
<i>object</i>	The private object that is registered.
<i>connection-id</i>	The ID of the connection on which the private object is registered.

Description

Private objects are available only for the particular connections on which they are registered.

The `g2orb-register-private` procedure is called from an Object Loader. For information about object loaders, see [Object Loaders and Locators](#).

When all connections associated with a private object are closed, the `g2orb-finalize` method is called for the object. The default behavior of this method is to delete the object.

g2orb-release

Called by a client application when it is finished with an object reference.

Synopsis

g2orb-release
(*object*: class g2orb-corba-object)

Argument	Description
<i>object</i>	The object that is released.

Description

This method unregisters and deletes a proxy object. Deleting a proxy object without first unregistering it can cause a memory leak in your ORB. For this reason, always call `g2orb-release` to delete proxy objects.

In the case of a co-resident client and server, the object is an implementation object and `g2orb-release` does nothing.

For information about how to build client applications, see [Building a CORBA Client Application](#).

Exception Management

This section describes the API methods and procedures used to programmatically manage exceptions:

- [g2orb-throw-exception](#)

g2orb-throw-exception

Provides a way for CORBA servers to return a user exception to the client.

Synopsis

g2orb-throw-exception
(*exception*: class g2orb-exception)

Argument	Description
<i>exception</i>	The exception thrown.

Description

Applications can create an instance of a user-defined exception class and call the `g2orb-throw-exception` method to cause a signal to be thrown. `g2orb-throw-exception` does not return to the caller.

An object definition is created by the IDL compiler for each user-defined exception described in the IDL.

Object Reference Manipulation

This section describes the API methods and procedures used to programmatically manipulate G2 CORBALink object references:

- [g2orb-decode-user-ref](#)
- [g2orb-object-to-string](#)
- [g2orb-objref-to-obj](#)
- [g2orb-obj-to-objref](#)
- [g2orb-ping](#)
- [g2orb-string-to-object](#)

g2orb-decode-user-ref

Decodes the g2orb-user-ref attribute of a proxy object.

Synopsis

g2orb-decode-user-ref
 (*object*: class g2orb-proxy)

Argument	Description
<i>object</i>	The proxy object.

Description

This method decodes the g2orb-user-ref attribute of a proxy object so that operations may be invoked on the proxy. You must call g2orb-decode-user-ref whenever the g2orb-user-ref attribute receives a new value.

Note If the G2orb-user-ref attribute of a proxy object is changed manually, g2orb-decode-user-ref will be called automatically by G2. However, if the attribute value is changed programmatically by a procedure, g2orb-decode-user-ref must be called to ensure that the change takes effect.

g2orb-object-to-string

Returns a standard CORBA Inter-operable Object Reference (IOR) for an implementation object.

Synopsis

```
g2orb-object-to-string  
  (object: class g2orb-server)  
  -> text
```

Argument	Description
<i>object</i>	The implementation object.

Return Value	Description
<u><i>text</i></u>	The Inter-operable Object Reference (IOR) for an implementation object.

Description

IORs are part of the CORBA standard. Other CORBA ORBs are able to read the return value of this method to generate a reference for a G2 CORBA object.

An IOR may be cut and pasted into another program, written to a file, or sent in an email message.

g2orb-objref-to-obj

Converts a text object reference into a client or server object.

Note This method has two forms: one is a method of `g2orb-obj`, and the other is a method of `g2orb-interface-parms`.

Synopsis

`g2orb-objref-to-obj`
 (*orb*: class `g2orb-obj`, *id*: text)
 -> *proxy*: class `g2orb-corba-object`

`g2orb-objref-to-obj`
 (*params*: class `g2orb-obj`, *id*: text)
 -> *proxy*: class `g2orb-corba-object`

Argument	Description
<i>orb</i>	A reference to an ORB object.
<i>params</i>	A reference to a <code>g2orb-interface-parms</code> for the interface that the object reference is expected to satisfy. A <code>g2orb-interface-parms</code> object is created by the G2 CORBALink compiler for each defined IDL interface. The <code>g2orb-interface-parms</code> object is given a name in the form: <code>g2orb-interface-interface-class-name</code>
<i>id</i>	The text object reference. This object reference may be imbedded as an attribute of a structure or a member of a sequence, union, or array. <code>g2orb-objref-to-obj</code> is automatically called for object references that are arguments of operations.
Return Value	Description
<i>proxy</i>	The proxy object or implementation object.

Description

The G2 language mapping for object references takes two forms:

- Item mapping for references that appear in the signature of a IDL operation.
- Value mapping for references that are imbedded in sequences, structures or unions.

This was done to simplify memory management, since values automatically reclaim storage when they are no longer needed and items allow methods to be called.

The `g2orb-objref-to-obj` and `g2orb-ref-to-objref` methods provide a way to convert back and forth between these two forms of language mapping.

An `objref` is similar to a IOR, but includes additional information that greatly speeds the conversion between text and item representations of object references.

This method converts a text object reference into a proxy or implementation object. This must be done before operations may be invoked on an object reference. If the object is local, the implementation object is returned. Otherwise, an object proxy is created.

The object reference may implement an interface that is more specific than the one indicated by the `g2orb-interface-parms` object. In that case, if the interface is known to the ORB, then a proxy for that class will be used. If the IDL for the more specific interface is not available, then the interface specified by the `g2orb-interface-parms` object will be used. If the interface specified by the object reference is known by the ORB and is not a subclass of the interface specified by the `g2orb-interface-parms` object, an error is signaled.

Note `g2orb-objref-to-obj` does not verify the existence of remote objects. Use `g2orb-ping` to determine if a remote object is accessible.

This method performs the inverse of the `g2orb-obj-to-objref` method.

g2orb-obj-to-objref

Converts a reference to a client or server object to a text object reference.

Synopsis

G2orb-obj-to-objref
 (*object*: class g2orb-corba-object)
 -> *text*

Argument	Description
<i>object</i>	Reference to either a client or a server CORBA object
Return Value	Description
<u><i>text</i></u>	A text reference to the <i>object</i> .

Description

This is the inverse of the [g2orb-objref-to-obj](#) method. See the description of [g2orb-objref-to-obj](#) on [for more information](#)

g2orb-ping

Verifies the existence of a remote object.

Synopsis

g2orb-ping
(*object*: class g2orb-corba-object)
-> *symbol*

Argument	Description
<i>object</i>	The remote object whose existence is being verified.

Return Value	Description
<u><i>symbol</i></u>	One of the following values: SUCCESS, if the object is found. UNKNOWN OBJECT, if the object is not found. COMM-FAILURE, if the server on which the object resides cannot be found.

Description

This method sends a locate message to the remote object, and returns a value indicating whether it verified the existence of the object.

g2orb-ping follows location-forwarding messages if the object has been moved to a new location or if a locator is used.

g2orb-string-to-object

Converts a standard CORBA Inter-operable Object Reference (IOR) to a proxy or implementation object.

Synopsis

```
g2orb-string-to-object
(orb: class g2orb-orb, ior: text)
-> proxy: class g2orb-corba-object
```

Argument	Description
<i>orb</i>	The <code>g2orb-orb</code> object to which the <i>ior</i> refers. For information about objects of this class, see g2orb-orb .
<i>ior</i>	The Inter-Operable Object Reference.
Return Value	Description
<i>proxy</i>	The resulting converted proxy object or implementation object.

Description

If the object referred to by the IOR is a remote object, then a proxy for the object is created and returned. If the object is a local object, then the implementation object is returned.

Language Mapping

Describes the mapping between G2 and IDL.

Introduction	65
IDL Primitive Types	66
Interfaces	70
Operations	72
Attributes	73
Object References	73
IDL Type to G2 Language Mapping	74



Introduction

The way that IDL compiler translates IDL into a target language is described by the language mapping for that language. G2 CORBALink 1.0 takes advantage of several new language features that are available in G2 version 5.0 and later.

G2 5.0 introduces two new types to the G2 class hierarchy:

- Structures
- Sequences

These language features allow all CORBA IDL types to map into G2 value types and do not require special efforts to reclaim memory passed as arguments in CORBA operations. This language mapping has the following characteristics:

- Any instance of an object that implements a CORBA interface will satisfy a G2 “is-a” test for the interface.
- Any proxy object will satisfy a G2 “is-a” test for it’s interface.
- User procedures may use “class interface-name” to hold a reference to an object without knowing if the object is local or remote.
- User procedures may use the “is a g2orb-proxy” to test if an object reference is remote or “is a g2orb-server” to test if an object reference is local.
- IDL inheritance and multiple-inheritance are supported.
- More than one implementation class may be defined for a given interface.
- Additional implementation attributes that are not defined in the interface may be added to implementation classes.
- Implementation objects may also subclass from non-CORBA classes in addition to the skeleton class.
- The work required to implement a CORBA client or server in G2 is minimized.

The full details of IDL are outside the scope of this document. The full specification may be found at the following Web site:

<http://www.omg.org>

IDL Primitive Types

The following table describes the mapping of IDL primitive types to G2:

IDL Type	G2 Type
short (16 bit)	integer
unsigned short (16 bit)	integer
long (32 bit)	quantity
long long (64-bit)	quantity
unsigned long (32 bit)	quantity
float (IEEE single precision)	float
double (IEEE double precision)	float
char	text
octet (8 bit)	integer

IDL Type	G2 Type
string	text
boolean	truth-value

The G2 type quantity is a super type that includes both G2 integers and G2 floats. G2 integers are 30 bit values. IDL Long values that fall into the valid range of a G2 integer are represented in G2 as integers. IDL Long values that fall outside this range are represented in G2 as a G2 float.

Representing Long Long Data Types

Because the resolution of G2 integers is 30 bits and the long long data type requires 64-bit resolution, the G2 representation of a long long is a sequence of 4 integers with values between 0 and 65,535.

Here are some examples of converting in both directions between long long data types and sequences of 16-bit integers where each integer has a value between 0 and 65,535. The least-significant 16 bits of the 64-bit long long or unsigned long long are in the first position of the sequence, the next 16 bits are in the second position, and so on.

Representing Unsigned Long Long Data Types

To represent unsigned long long and non-negative (signed) long long types:

- 1 Convert the number to hexadecimal.
- 2 Break the results into groups of 4 digits.
- 3 Convert each of the 4-digit hexadecimal numbers to integers.
- 4 Construct a sequence of these integers with the least-significant integer first.

For example, to represent 5,000,000,000,000,000 as a sequence:

- 1 5,000,000,000,000,000 = 0x4563918244F40000
- 2 The 4 hexadecimal numbers are 4563, 9182, 44F4, and 0000
- 3 The decimal values of these numbers are 17763, 37250, 17652 and 0
- 4 The G2 representation of 5,000,000,000,000,000 is:

```
sequence(0, 17652, 37250, 17763)
```

To represent 87,000,000,000 as a sequence:

- 1 87,000,000,000 = 0x14419AA600
- 2 The 4 hexadecimal numbers are 0000, 0014, 419A, A600
- 3 The decimal values of these numbers are 0, 20, 16794, 42496

- The G2 representation of 87,000,000,000 is:
sequence(42496, 16794, 20, 0)

Representing Negative Long Long Data Types

To represent negative long long types:

- Convert the absolute value of the number to hexadecimal.
- Reverse each bit in the result. This is equivalent to subtracting the result from FFFFFFFFFFFFFFFF or each digit from 15 decimal.
- Add one to the result.
- Break the results into groups of 4 digits.
- Convert each of the 4-digit hexadecimal numbers to integers.
- Construct a sequence of these integers with the least-significant integer first.

This example shows how to find the G2 representation of a long long -20:

- The long long hexadecimal representation of positive 20 is
0x0000000000000014
- $0xFFFFFFFFFFFFFFFF - 0x0000000000000014 = 0xFFFFFFFFFFFFFFFEB$
- $0xFFFFFFFFFFFFFFFEB + 1 = 0xFFFFFFFFFFFFFFFEC$
- The four hexadecimal numbers are FFFF, FFFF, FFFF, and FFEC
- The equivalent decimal numbers are 65535, 65535, 65535, and 65516
- The G2 representation of a long long -20 is:
sequence(65516, 65535, 65535, 65535)

This example shows how to find the G2 representation of a long long -5,000,000,000,000,000,000:

- The hexadecimal representation of positive 5,000,000,000,000,000,000 is
0x4563918244F40000.
- ```

FFFFFFFFFFFFFFFF
- 4563918244F40000

BA9C6E7DBB0BFFFF

```
- $BA9C6E7DBB0BFFFF + 1 = BA9C6E7BBB0C0000$
- The four hexadecimal numbers are BA9C, 6E7B, BBB0, and 0000.
- The equivalent decimal numbers are 47772, 28285, 47884, and 0.
- The G2 representation of -5,000,000,000,000,000,000 is:  
sequence( 0, 47884, 28285, 47772)



## Calculating the Value of Unsigned Long Long Data Types

The following algorithm shows, in general terms, how to calculate the value represented by an unsigned long long returned to G2. Suppose you put the elements of an unsigned long long sequence in the integer array `els[4]` and the data type `huge` is capable of holding the maximum value of an unsigned long long.

```

huge g2ToULongLong(int els[4])
{
 huge accumulator = 0 ;
 inti ;
 for (i = 3 ; i >= 0 ; i--)
 {
 // Equivalent to shifting the accumulator
 // up by 16 bits (accumulator << 16)
 // =====
 accumulator = accumulator * 65536 ;
 accumulator = accumulator + els[i] ;
 }
 return accumulator ;
}

```

For signed long longs, if `huge` were replaced by `long long` in the above algorithm, if `long long` held 64-bit values, and if negative numbers were represented using two's complement, then the above algorithm would correctly translate signed long longs.

Suppose the `huge` data type could hold a large value, for example, 128-bit values, and you wanted to translate a signed long long. Then:

- You would know the array represented a negative number if `els[3] >= 32,768`.
- You could find the absolute value of the represented number by following the above algorithm and subtracting the results from `0x10000000000000000` ( $2^{**64}$ ).

For example, suppose `sequence(0, 47884, 28285, 47772)` represents a signed long long. To determine what number it represents:

- 1 Since `47772 >= 32768`, we know it represents a negative number.
- 2 Following the algorithm above gives a result equivalent to:

$$65536 * (65536 * ( (65536 * 47772) + 28285) + 47884) + 0 = 13,446,744,073,709,551,616$$

$$3 \quad 2^{**}64 = 18,446,744,073,709,551,616$$

$$\begin{array}{r}
 18,446,744,073,709,551,616 \\
 - 13,446,744,073,709,551,616 \\
 \hline
 5,000,000,000,000,000,000
 \end{array}$$

In other words, sequence (0, 47884, 28285, 47772) represents -5,000,000,000,000,000,000.

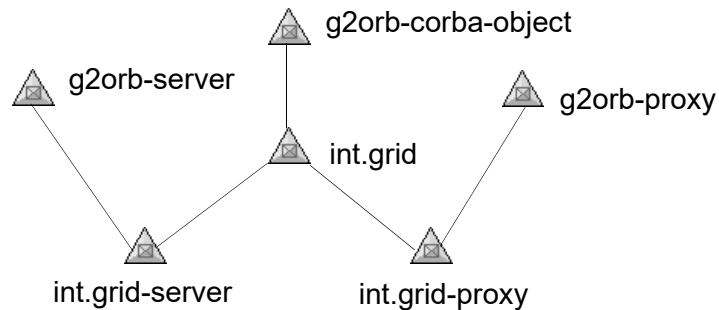
Now suppose sequence(0, 47884, 28285, 47772) represents an *unsigned* long long. You calculate its value, using the same formula used in step 2, above. The value is 13,446,744,073,709,551,616.

## Interfaces

IDL Interfaces are mapped to the following G2 class definitions:

| Class     | Definition                                                                                                                                                                                                                                                                                                                                                                                                  |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| interface | The interface class is created without attributes. It is an abstract class that is used in method signatures to represent an object that implements the IDL interface. This class is a subclass any base classes, if any, or <code>g2orb-corba-object</code> if there are no base classes. The name of the interface class is the absolute name of the interface.                                           |
| proxy     | The proxy object represents a client to an object that implements the interface. The name of the default proxy class for an interface is the absolute name of the interface with <code>-proxy</code> appended. Method calls on an instance of this object are redirected over the network to the object implementation. The proxy class is a subclass of the interface class and <code>g2orb-proxy</code> . |
| server    | The server class is used to represent server objects. The name of the server class is the absolute name of the interface class with <code>-server</code> appended. The server class includes attributes defined in the IDL definition of the interface.                                                                                                                                                     |

For example, the following figure illustrates the class hierarchy of the object definitions created for the “grid” IDL interface:



This means that:

- g2orb-corba-object refers to any CORBA object, either client or server.
- g2orb-server refers to any CORBA server object.
- g2orb-proxy refers to any CORBA client object.
- int.grid refers to any grid object, either client or server.

The IDL compiler generates a total of three object definitions for each IDL interface. For example, the class definitions of the “grid” IDL interface are:

| Class    | Description                                                                                                                                                                                                     |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| int.grid | <p>This is an abstract class that allows an application to refer to any object that implements the “grid” IDL interface, either client or server.</p> <p>Instances of this object class may not be created.</p> |

| Class           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| int.grid-server | <p>This class is a object that implements the “grid” server. The compiler will generate empty stubs for each method defined in the IDL.</p> <p>The user may modify these definitions as needed. Also, the class may be subclassed if:</p> <ul style="list-style-type: none"> <li>• The user application needs more than one object class that implements the given interface.</li> <li>• Additional attributes are needed for the objects that are not in the IDL interface.</li> </ul> |
| int.grid-proxy  | <p>This class is a client or proxy for a remote object. The compiler will generate methods for each method defined in the IDL. These methods will forward the method call to the remote object.</p> <p>This object may be subclassed if there is a need to add more complex behavior, such as caching of recently accessed values. This is known as a CORBA smart proxy.</p>                                                                                                            |

## Operations

CORBA Operations are mapped into G2 as methods. Methods are defined for:

- Proxy class
- Server class

Methods on the Proxy class will redirect the operation over the network. Methods on the Server class will represent the server side functionality of the operation.

## Attributes

Attributes are implemented as get and set methods. The names of the methods are:

- `get-orb-name`
- `set-orb-name`

where:

*name* indicates the name of the attribute. Set methods are not created if the attribute is read-only.

## Object References

Mapping object references into G2 depends on whether the object reference appears in an operation signature. Two mappings are used to minimize the effort required to avoid memory leaks.

If the object reference appears in an operation signature, then the object reference is mapped into a G2 object reference, either to an implementation object or a proxy object. Each time an object reference enters G2 from the Orb, it is checked to see if the object that it represents resides in this G2 memory space.

- If the object does reside in the memory space, then a reference to the object implementation is used. In either case, operations may be invoked on the object by calling the corresponding method.
- If the object implementation resides in the same memory space as the client, then the operation is invoked with no overhead. Otherwise, the operation will be redirected over the network.

If the object reference does not appear as a parameter of an operation, then the object reference is represented as a value. This would occur if the object reference is passed as a member of a structure, a union, a sequence, or an array element. The method `g2orb-objref-to-obj` will convert the value into an implementation or proxy. This must be done before an operation may be invoked on the object reference

To delete an object reference that appears in a signature, the `g2orb-release` method must be called on the object reference. Object references that do appear in the operation signature are represented as values and do not require special memory management. When the object reference goes out of scope, it will be removed without special action.

# IDL Type to G2 Language Mapping

## Enumerated Types

IDL enumerated types are mapped to G2 symbols.

## Structures

Each IDL structure is mapped to a G2 structure. The name of the G2 structure member is the name of the IDL structure member prefaced with “orb-“. The prefix ensures that attribute names will not collide with G2 reserved words.

An additional structure member is provided for each G2 structure. It has the name, `g2orb-structure-name`, and contains a symbolic name of the structure.

## Sequences

Each IDL sequence is mapped to a G2 sequence.

## Arrays

Each IDL array is mapped to a G2 sequence.

## Unions

Each IDL union is mapped to a structure, whose attributes are:

| Attribute                        | Type  | Description                           |
|----------------------------------|-------|---------------------------------------|
| <code>g2orb-discriminator</code> | value | Indicates which case of union holds.  |
| <code>g2orb-value</code>         | value | Holds the current value of the union. |

## Any

Each IDL any is mapped to a structure, whose attributes are:

| Attribute   | Type  | Description                                                                                            |
|-------------|-------|--------------------------------------------------------------------------------------------------------|
| g2orb-type  | text  | Indicates the IDL type held by the any object. This text field is used internally and is not readable. |
| g2orb-value | value | Holds the value of the any object.                                                                     |

---

**Note** G2 CORBALink does not support the Any type.

---





# Sample G2 CORBALink Application

*Illustrates the steps that you follow to build a G2 CORBALink application.*



## Introduction

In this example, both the client and the server happen to be implemented in G2 and reside in the same G2. However, the client and sever can also reside in different G2s or be implemented in a different language.

The following table summarizes the steps for building a simple G2-CORBALink application:

| Steps                                                                         | For details, see... |
|-------------------------------------------------------------------------------|---------------------|
| <b>1</b> Start the G2-CORBALink bridge process.                               |                     |
| <b>2</b> Merge the <code>g2idl.kb</code> into your application.               |                     |
| <b>3</b> Set up G2-CORBALink to support your application and compile the IDL. |                     |
| <b>4</b> Create an Server object.                                             |                     |
| <b>5</b> Create a Proxy object.                                               |                     |
| <b>6</b> Create a client procedure.                                           |                     |
| <b>7</b> Start the application.                                               |                     |

These steps summarize how to building a simple G2 CORBALink application:

- 1 [Start the G2 CORBALink bridge process.](#)
- 2 [Merge the g2idl.kb into your application.](#)
- 3 [Set up G2 CORBALink](#) to support your application and [compile the IDL.](#)
- 4 [Create a Server object.](#)
- 5 [Create a Proxy object.](#)
- 6 [Create a client procedure.](#)
- 7 [Start the application.](#)

## Starting G2 CORBALink

**To start the G2 CORBALink process:**

➔ Execute the following command:

```
g2-corba gsi-port iiop-port
```

where:

*gsi-port* is the port through which G2 communicates with the ORB.

*iiop-port* is the port through which the ORB communicates with IIOP clients.

For example:

```
g2-corba 20000 1594
```

## Merging the g2idl KB into the Application

**To merge g2idl into your application:**

- 1 Choose Main Menu > Merge KB to display the Load KB workspace.  
The merge in this KB option is enabled on the workspace.
- 2 Specify the location of the `g2idl.kb` file and click End.

---

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

---

## Setting Up and Compiling the IDL

### To set up G2 CORBALink:

- 1 Start G2.
- 2 Chose Main Menu > New Workspace to create a workspace on which to place the G2 CORBALink objects that support the application
- 3 Choose Main Menu > Get Workspace > g2idl-top-level to display the G2 CORBALink IDL Compiler workspace
- 4 Clone a CORBA ORB object and place it on your workspace, and name it ORB1.
- 5 Name and configure the Orb by specifying values for these attributes:

| Attribute                    | Value                                                                                                                                                                             |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| names                        | orb1<br><br><b>Note:</b> The Orb must be named or G2 will not activate a connection.                                                                                              |
| gsi-connection-configuration | tcp-ip host "localhost" port-number 20000<br><br><b>Note:</b> The <i>tcp-ip-port-number</i> must match the first port number that was specified to start the g2-corba executable. |

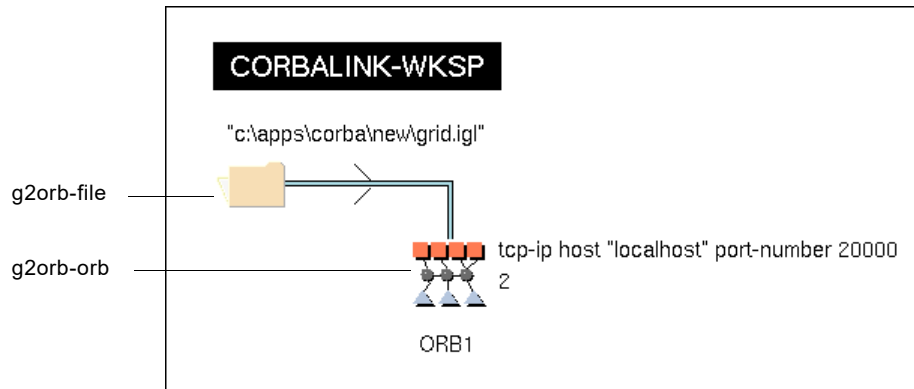
- 6 Clone a CORBA IDL File object and place it on your workspace, and specify the pathname of the `grid.idl` file provided with G2 CORBALink for the `g2orb-file-path` attribute.

For example:

| a g2orb-file              |                              |
|---------------------------|------------------------------|
| Notes                     | OK                           |
| Names                     | none                         |
| G2orb file path           | "c:\apps\corba new\grid.idl" |
| G2orb time of last change | 0.0                          |

- 7 Connect the corba-orb and corba-idl-file objects.

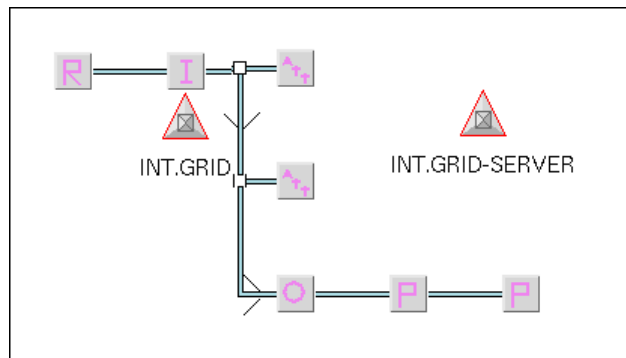
For example:



### To compile the IDL:

→ Chose compile idl from the orb1 object menu.

The compilation process begins. When complete, a workspace appears containing the results of the compilation, for example:



This workspace contains:

- Objects that describe the interfaces from the IDL source file. These objects are downloaded into the CORBA ORB to enable it to send and receive requests.
- G2 objects that map the interfaces defined in the CORBA IDL into the G2 object system. Instances of these objects are created to represent CORBA objects and proxies.

You have successfully compiled the CORBA IDL.

## Creating a Server Object

Server objects are concrete objects in CORBA. You create instances of server objects on your workspace.

**To create an Server object:**

- 1 Choose New Object > int.grid > int.grid-server from your KB Workspace menu.
- 2 Specify values for the these attributes:

| Attribute         | Value     |
|-------------------|-----------|
| g2orb-object-name | "my grid" |
| orb-width         | 10        |
| orb-height        | 20        |

- 3 Choose register from the Server menu to register this object with the ORB.

After creating the Server object, you need to edit its method, which was generated by the IDL compiler.

**To edit a method of the Server object:**

- 1 Choose Inspect > go to > int.grid-server::orb-fetch.  
This method was generated by the IDL compiler.
- 2 Choose edit from the method menu to open the editor.
- 3 Replace the line {TODO: Put your code here} with the following lines:  

```
ret-result =random(5000);
inform the operator that "Server: returning value [ret-result]";
```

This is the edited version of the example method:

```
orb-fetch(O: class INT.GRID-server, orb-n: integer, orb-m: integer) =
(quantity)

ret-result: quantity;

begin
inform the operator that "Got @"fetch@" request on class [the class of
O]";
ret-result = random(5000);
inform the operator that "Server: returning value [ret-result]";
return ret-result;
end
```

You now have a working CORBA server.

## Creating a Proxy Object

The proxy object is a client representation of a remote object. You create instances of proxy objects on your workspace

### To create a proxy object:

- 1 Choose New Object > int.grid > int.grid-proxy from your KB Workspace menu.
- 2 Specify the location of the object implementation (URL or IOR) that this proxy uses in the G2orb-user-ref attribute.

For example, the URL location for this example is:

```
"//localhost:1594/my grid"
```

This assumes that the ORB is running on the local system with a IIOP port at 1594. The object-name of the object is `my grid`. For more information, see [Specifying an Object Location](#).

- 3 Name this object `grid-proxy`.
- 4 Choose ping from the `grid-proxy` object menu to check the connection.

The Message Board should display the following message, indicating that G2 CORBALink was able to connect the `grid-proxy` object to the Server object:

```
Ping: SUCCESS
```

## Creating a Client Procedure

Create a G2 procedure to be a client of G2 CORBALink. The following example is a simple client procedure:

```
client()
Width, Height, Val: integer;

begin
 Width = call get-orb-width(grid-proxy);
 Height = call get-orb-height(grid-proxy);
 Val = call orb-fetch(grid-proxy, 1,2);
 inform the operator that "Got: Width = [Width], Height = [Height],
 Value = [Val]";
end
```

For each attribute, the IDL compiler generates a `get` method of the form `get-orb-attribute name()`.

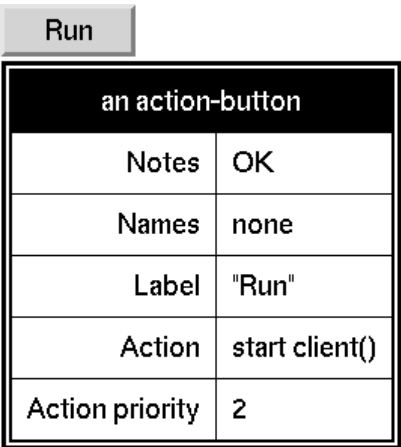
In this example, the procedure retrieves the values of the width and height attributes of the remote object. The client calls the `fetch()` method with two arguments and displays the results on the Message Board.

## Starting Your G2 CORBALink Application

To start your G2 CORBALink application:

- 1 Pause your KB.
- 2 Create a G2 action button to start the G2 procedure.

For example:



| an action-button |                |
|------------------|----------------|
| Notes            | OK             |
| Names            | none           |
| Label            | "Run"          |
| Action           | start client() |
| Action priority  | 2              |

- 3 Resume your KB and click the Run action button to see the results of running your application.

The `client()` procedure posts messages on the Message Board. The first two messages in the Message Board are posted by `inform` statements in the `orb-fetch` method of the Server object. The third message is posted by the `inform` statement in the G2 procedure `client()`.

For example:

```
MESSAGE-BOARD
#42 11:58:22 a.m. Got "fetch" request on
 class INT.GRID-SERVER
#43 11:58:22 a.m. Server: returning value
 1348
#44 11:58:22 a.m. Got: Width = 10, Height =
 20,
 Value = 1348
```

The first two messages in the Message Board are posted by `inform` statements in the `orb-fetch` method of the Server object. The third message is posted by the `inform` statement in the `G2` procedure `client()`.

When you complete the preceding steps, you have created a working CORBA client and server in `G2`. This simple example includes only a single server object co-resident with a single client, and has only two attributes and one operation. This CORBA server is accessible from CORBA clients written in different languages.

Although real working application will include more objects and have more complex interfaces than this example, you must follow the same steps to create them that you followed to create this example.

You should now save your work.

The IDL compiler is no longer needed by the application. The module information system table can be modified to make the top level module no longer dependent on the `G2` IDL module. The `G2ORB` module is still needed to run the application. The modules that are no longer needed can be deleted, along with their workspaces.

To recompile or modify the CORBA interfaces, the IDL compiler can be merged with the application.



# Interfacing G2 CORBALink with Other CORBA Servers

---

*Describes G2 CORBALink interfaces with other CORBA Servers, the problems and solutions.*

Introduction 85

Iona's Orbix 87



## Introduction

G2 CORBALink is typically used to interface G2 to CORBA systems that are built with other Object Request Brokers such as Orbix or Visi-Broker. This is possible since CORBA Object Request Brokers use the same network protocol, Internet Inter-ORB Protocol (IIOP). This allows ORBs built by different vendors to exchange object references, call methods, and access remote object attributes.

However, CORBA is an evolving standard, and there are still a few rough edges. One problem area is difficulty obtaining initial object references. Let's say that you are building a distributed object system with lots of objects. In general, these systems need a way to find object references based on an object's identity, relationships, or characteristics.

Standard, “Common Object Services” are available from several vendors that obtain object references. These services are part of the CORBA specification, in which:

- The CORBA Name Service will return an object reference based on a name.
- The CORBA Relationship Service will return object references that participate in a specific relationship with other objects.
- The CORBA Trading Service will find a reference for an object that has a specific characteristic.

These Common Object Services are implemented as simple CORBA servers and are available from several vendors. G2 G2 CORBALink may be used with any of them. Other services may be created on an ad-hoc basis to meet specific project requirements and implemented in G2 or some other language such as C++ or Java. To use one, simply:

- 1 Compile the service's IDL into G2 objects.
- 2 Create a proxy object for the remote Service.
- 3 Call the methods of the proxy object. This proxy object may then obtain proxies for other objects.

However, how does one create the initial object reference? This is known as the “bootstrap” problem, since a reference to the Name Service or some similar server may be used to “bootstrap” other object references.

Basically, this problem is solved by either:

- Constructing the object reference, using the URL format, or
- Obtaining an Interoperable Object Reference (IOR) from the remote server.

Using a URL format requires knowledge of the remote object's “object key” or “marker.” The CORBA standard does not specify how this value is generated. Some ORBs do not provide a way to discover or specify this value. Some ORBs use a different value each time the server is started or use a binary value, preventing it from being hard-coded into the G2 application.

The other approach is to use the remote server's `object_to_string()` API call to get an IOR from the server. The IOR is a long string of hexadecimal numbers that starts with “IOR:...” This string may be cut and pasted into the `G2orb-user-ref` attribute of a G2 proxy object. If the IOR changes each time the server starts, then it may be written to a file and read by G2 procedure. Users have reported that this approach works well with the Visigenic ORB and DIAS.

## Iona's Orbix

The Object Request Broker with the largest market share is Orbix. Interfacing to Orbix is a little tricky due to some bugs that exist in Orbix 2.2. The method that we describe here works with Orbix 2.2 and later.

Orbix objects may be addressed with G2 proxies using the URL format. The URL for a Orbix object has the following general format, which is not likely to change for compatibility reasons:

```
\\[server-host]:[iiop-port]\\:[server-host]:[serverName]:[marker]::IR:[type]
```

where:

The marker is provided by the programmer. If this field is blank, it will match any object on the server.

- *server-host* is the network name of the computer where the server is running.
- *iiop-port* is the IIOP port of the server.
- *serverName* and *type* are usually just the IDL name of the interface.

For example, if the “iiop\_grid” example that ships with Orbix is started on port 4592 on system “bill”, the URL for the grid object would be:

```
\\bill:4592\\:bill:grid::IR:
```

The server should be started and left running. Orbix calls this a “persistent” server. However, there does not seem to be a reliable way to specify the IIOP port number. Surprisingly, this is NOT the IIOP port number that is specified in the Orbix Server Manager. A bug? Orbix will assign a port number at run time, and there is no obvious way to predict what this port number will be (Version 2.2).

There is a way around this problem. Orbix provides something called a Locator Daemon. The daemon has an IIOP interface that is defined in IDL. It is possible to configure the IIOP port of the daemon and thus construct a URL for it.

The URL for the daemon is:

```
\\[daemon-host]:[daemon-port]\\:[host]:IT_daemon::IR:IT_daemon
```

The daemon may also be started with the “-i” option. This will write the daemon's IOR to a file at startup. The daemon's `getIIOPDetails()` method will start the remote server if needed and provide the server's IIOP port number. This port number may be used to construct a URL for the remote object as described above.

The `grid_iiop` server that is included as one of the Orbix examples provides a good way to test these methods.

---

**Note** The `g2orb-ping` operation does not work with Orbix 2.2. It will always respond with an “object here” reply, regardless of the actual status of the object.

---

To summarize, the steps to use G2 with Orbix are:

- 1** Compile the Orbix daemon IDL along with your application IDL. Just add another "IDL file" object to your Orb object in G2 to include this IDL.
- 2** Generate a proxy for the Orbix daemon using either a URL or an IOR generated by the daemon with the "-i" option.
- 3** Call `getIOPDetails()` on the locator for the object of interest.
- 4** Using the provided port number, construct a URL for the needed object. Place this in the `G2orb-user-ref` attribute of a proxy object. Remember to call `g2orb-decode-user-ref` on the object if you do this from a procedure.
- 5** Other object references may be freely passed between G2 and Orbix as method parameters or return values.

---

**Note** The `g2orb-util.kb` file provides an example of how this is done.

---

|   |          |          |          |          |          |          |          |          |          |          |          |          |          |
|---|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| @ | <b>A</b> | <b>B</b> | <b>C</b> | <b>D</b> | <b>E</b> | <b>F</b> | <b>G</b> | <b>H</b> | <b>I</b> | <b>J</b> | <b>K</b> | <b>L</b> | <b>M</b> |
| # | <b>N</b> | <b>O</b> | <b>P</b> | <b>Q</b> | <b>R</b> | <b>S</b> | <b>T</b> | <b>U</b> | <b>V</b> | <b>W</b> | <b>X</b> | <b>Y</b> | <b>Z</b> |

---

## Numerics

64-bit data types, representing in G2

## A

any IDL type  
 Application Programmer? Interface (API)  
   exception management  
   initialization  
   object reference manipulation  
   object registration  
 array IDL type

## C

client applications  
   building  
   exceptions thrown in  
   object location specifying for  
 CORBA  
   client applications  
     building  
   exceptions  
     system  
     user  
   server applications  
     building  
 CORBA operations  
   mapping to G2  
 CORBA servers  
   interfacing with G2 CORBALink  
 co-resident server and client  
 customer support services

## D

data types  
   64-bit  
   long long

## E

enumerated IDL type  
 exception management API  
   g2orb-throw-exception  
 exceptions in CORBA

## F

files  
   grid.idl

## G

G2 CORBALink  
   interfacing with other CORBA servers  
 G2 CORBALink classes  
   g2orb-file  
   g2orb-interface-parms  
   g2orb-iobject  
   g2orb-orb  
   Interface  
   proxy  
   server  
 G2 CORBALink Compiler  
   preprocessor support provided by  
   g2idl.kb compiler module  
   g2idl-compile  
   g2orb-decode-user-ref  
   g2orb-deregister  
   g2orb-download-ir  
   g2orb-file object  
     attributes  
       g2orb-file-path  
   g2orb-init-bridge  
   g2orb-init-orb  
   g2orb-interface-parms object  
     attributes  
       g2orb-abs-name  
       g2orb-interface  
       g2orb-proxy-class  
       g2orb-repository-id  
   g2orb-iobject object

- g2orb-object-to-string
- g2orb-objref-to-obj
- g2orb-obj-to-objref
- g2orb-orb object
  - attributes
    - g2orb-connect-timeout
    - g2orb-idl-connection-timeout
    - g2orb-include-search-path
    - g2orb-init-procedure
    - g2orb-object-loader
    - g2orb-prefix
    - g2orb-principal
    - g2orb-register-objects-on-startup
    - g2orb-server-request-timeout
    - g2orb-client-request-timeout
    - gsi-connection-configuration
    - gsi-interface-status
  - names
  - user menu choice
- g2orb-orb objects
  - creating
- g2orb-ping
- g2orb-proxy object
  - attributes
    - g2orb-user-ref
  - user menu choices
- g2orb-register
- g2orb-register-objects
- g2orb-register-private
- g2orb-release
- g2orb-server object
- g2orb-string-to-object
- g2orb-throw-exception
- grid.idl file
- GSI Interfaces
  - gsi-connection-configuration attribute of
  - gsi-connection-configuration attribute of GSI Interfaces

## I

- IDL
  - default file
    - grid.idl
  - preprocessor
  - updating
- IDL type
  - mapping to G2
    - any
    - arrays
    - enumerated types

- sequences
- structures
- unions
- initialization API
  - g2idl-compile
  - g2orb-download-ir
  - g2orb-init-bridge
  - g2orb-init-orb
- Interface object
  - attribute
    - g2orb-object-name
  - user menu choices
- Inter-operable Object References (IOR)
- IOR *See* Inter-operable-Object References

## L

- language mapping
  - attributes
  - CORBA operations
  - IDL interfaces
  - IDL primitive types
  - IDL type
  - object references
- long long data types, representing in G2

## O

- object locations
  - specifying
- object reference manipulation API
  - g2orb-decode-user-ref
  - g2orb-object-to-string
  - g2orb-objref-to-obj
  - g2orb-obj-to-objref
  - g2orb-ping
  - g2orb-string-to-object
- object references
  - mapping into G2
- object registration API
  - g2orb-deregister
  - g2orb-register
  - g2orb-register-objects
  - g2orb-register-private
  - g2orb-release
- objects loaders and locators
- ORB
  - starting
- Orbix
  - interfacing to

## **P**

- preprocessor support for IDL
- private objects
- procedures
  - API
    - exception management
    - initialization
    - object reference manipulation
    - object registration

## **S**

- sequence IDL type
- server applications
  - building
  - exceptions thrown by
  - implementation classes for
  - registering objects for
- server classes
  - compiler-generated default
- smart proxies
- starting the ORB
- structure IDL type

## **U**

- Uniform Resource Locator (URL)
- unions IDL type
- updating IDL

