Orbix Wonderwall Administrator's Guide

IONA Technologies PLC
February 1999

M 2 2 5 8

Contents

Preface

The Internet Inter-ORB Protocol (IIOP) was introduced as part of the Common Object Request Broker Architecture (CORBA) 2.0 General Inter-ORB Protocol (GIOP). IIOP facilitates the use of distributed CORBA objects over the Internet.

Typical Internet security involves the use of a *firewall* to restrict access to hosts on a local network. Wonderwall is IONA Technologies' implementation of the firewall model and was developed to address security issues arising from using CORBA over the Internet.

This guide presents details of Wonderwall's implementation of the firewall model and addresses security issues arising from using CORBA over the Internet.

Audience

This guide is aimed at system administrators who wish to set up a Wonderwall environment and programmers who wish to develop OrbixWeb/Java applications that communicate over the Internet via Wonderwall.

This guide does not assume that the reader has any knowledge of firewall security issues. This guide assumes that programmers have significant knowledge of OrbixWeb programming.

Organisation of this Guide

This guide is divided into eight chapters:

Chapter 1, "An Introduction to Wonderwall"

This chapter provides an overview of Internet security and describes how Wonderwall was developed according to the firewall model.

Chapter 2, "Getting Started with Wonderwall"

This chapter explains how to get started with Wonderwall. It details how to set up and configure Wonderwall.

Chapter 3, "The Wonderwall GUI Configuration Tool"

This chapter explains how to use the Wonderwall GUI Configuration Tool to modify default security configuration settings for Wonderwall. It does this by editing the `iiopproxy.cf` file which stores the configuration settings for your Wonderwall installation.

Chapter 4, "The Wonderwall Log Analysis Viewer"

This chapter describes the Wonderwall Log Analysis Viewer which allows you to view, edit, and modify log files via a graphical user interface.

Chapter 5, "IORs and IIOP"

This chapter discusses the Interoperable Object Reference (IOR), the mechanism used to establish communication between clients and servers, and the Internet Inter-ORB Protocol (IIOP) in detail, describing IOR formats and IIOP message formats respectively.

Chapter 6, "Interoperability and Details"

This chapter discusses issues associated with interoperability. For example, object references, proxification, connection establishment, factory objects, and IORs. Wonderwall is fully interoperable.

Chapter 7, "Transformers"

Transformers allow encryption of messages prior to transmission via the TCP/IP protocol. This chapter introduces transformers by describing how they can be used with Wonderwall.

Chapter 8, "Using Wonderwall with OrbixWeb"

OrbixWeb 3 contains built-in support for Wonderwall. This chapter describes how Wonderwall can be used with OrbixWeb as either an intranet request-router or as a firewall proxy. It also details how to configure OrbixWeb to use Wonderwall.

Appendix A, "iiopproxy and iortool"

This appendix describes the `iiopproxy` process which is responsible for implementing the firewall, and the `iortool` utility which is responsible for manipulating object references.

Appendix B, "Configuration"

This appendix describes the Wonderwall configuration file under the following headings: basic settings, list of IORs, and access control list.

Document Conventions

This document uses the following typographical and keying conventions:

`Constant width`	Constant width words or characters represent source code or system values you must use literally, such as commands, options, and path names.
Italic	Italic words in normal text represent emphasis and new terms. Italic words or characters in code and commands represent variable values you must supply, such as arguments or commands or path names for your particular system.

This guide uses the following keying conventions:

< >	Some command examples use angle brackets to represent variable values you must supply. For example, `<Wonderwall location>`
... . . .	Horizontal or vertical ellipses in format and syntax descriptions indicate that material has been eliminated to simplify a discussion.
[ ]	Brackets enclose optional items in format and syntax descriptions.
{ }	Braces enclose a list from which you must choose an item in format and syntax descriptions.
|	A vertical bar separates items in a list of choices enclosed in { } (braces) in format and syntax descriptions.

1

An Introduction to Wonderwall

Wonderwall, developed according to the firewall model, addresses security issues arising from using CORBA over the Internet. This chapter introduces Wonderwall—an object-oriented and flexible approach to security.

Internet Security Overview

The Internet Inter-ORB Protocol (IIOP), a specialization of the CORBA General Inter-ORB Protocol (GIOP), paves the way for using distributed CORBA objects over the Internet. This opening up of the Internet, however, brings its own problems and risks. There will always be a few users willing to exploit any security loopholes to cause damage to your system. Some level of security is necessary to keep these intrusions at bay. A typical approach to Internet security is to use a *firewall* to restrict access to hosts on your local network. The basic model is to direct all traffic to and from the internal network through a single point which can monitor and control every transmitted message.

There are firewalls currently available which restrict access to a local network in a variety of ways—for example, access to certain hosts and certain commands can be limited. However, in a distributed object environment such as CORBA, it is important that security be object-oriented. Experience has shown that it is bad practice to implement security which is too coarse-grained. Users presented with a choice between two levels of security, one which is too restrictive and another which is too permissive, will inevitably choose the permissive level of security on occasion—and consequently a breach appears in the network defences.

Wonderwall is developed according to the firewall model and addresses the security issues arising from using CORBA over the Internet. It provides a flexible, object-oriented approach to security allowing control of access to individual objects even down to the level of individual operations on objects.

Wonderwall and the Firewall

Wonderwall is a firewall proxy server that is specifically designed to filter, control, and log IIOP traffic between Orbix clients on the exterior (the Internet), and Orbix servers on the interior (the intranet).

The usual approach to building a typical firewall involves restricting Internet access to a single IP port on a single host for each service. This host will be the only host which is physically connected to the Internet and the restriction to using a single well known IP port provides an additional safeguard.

A certain refinement of this model involves making the firewall host a dedicated, secure host, known as a *bastion host*. The bastion host is dedicated exclusively to the role of gateway to the Internet and its configuration can be toughened to make it extra secure against unwanted incursions. This approach has the clear advantage that much of the security effort can be focused on this one machine. For example, many directories on the bastion host can be made read only to `root` without inconveniencing anyone.

The firewall is usually implemented as a proxy server process which runs on the bastion host (as is done, for example, with HTTP and SMTP firewalls). Wonderwall follows this pattern and implements an IIOP proxy server. The role of the server process is to listen to incoming messages on the well-known IP port and to pass on these messages to the internal network, after subjecting them to close scrutiny. When any potentially hostile or forbidden messages are encountered, these are blocked and not passed on to the internal network.

Wonderwall Features

Typically, the potential for security loopholes increases considerably once a complex application connects to the Internet. In contrast, the Wonderwall server is a simple stand-alone process, which requires no special privileges, forks or processes, and interacts with the bastion host in a simple manner.

Wonderwall's IIOP firewall proxy server has the following features which contribute to strengthening the security of an internal network:

- The use of a bastion host is facilitated.

  The model on which Wonderwall is built supports the use of a bastion host as the basis of your firewall. You have only to install the Wonderwall server on the bastion host and it will act as a liaison between the outside world and your internal network. Alternatively, you can install Wonderwall on a regular host if you prefer.

- Messages are filtered.

  All messages arriving on the server's well known port are filtered. Wonderwall is not just a facility to monitor initial connections to CORBA objects. For example, it will continue to monitor (and potentially block) all messages which pass between an external client and the internal CORBA object.

- Message filtering based on Request header.

  A number of message types are defined for the IIOP protocol and any or all of these can be blocked if necessary. The most important group of incoming messages are the Request messages which are used to invoke methods on CORBA objects. Wonderwall provides comprehensive filtering of these messages based on the content of the Request message header. This header provides all information needed to provide effective filtering. For example, the identity of the target object and the intended operation name. Request messages can be checked rapidly and passed to the internal network with little performance overhead.

- Fine-grained control of security.

  Wonderwall provides the kind of fine-grained control of security which is needed for a distributed object environment. It allows you to control access to individual objects and, moreover, to allow or deny access to specific methods defined on that object. There are a number of other criteria which can be checked as will be seen later.

- Message encryption.

  Message encryption is an essential feature needed for the exchange of private messages over the Internet. Wonderwall supports the exchange of encrypted IIOP messages using the Orbix *transformer* mechanism. This allows the programmer to encrypt a complete IIOP message using a custom encryption algorithm.

- Message logging.

  The logging facility of Wonderwall (which can be configured to focus on particular kinds of events) is a powerful facility for tracing the history of suspicious message exchanges. It is also broadly useful as a debugging and monitoring facility.

- Blocks messages unless specifically allowed.

  Wonderwall observes and promotes good security practice. For example, its approach to filtering is that everything is forbidden unless it is expressly allowed.

- Promotes simplicity of proxy server.

  According to Wonderwall, a proxy server ought to behave simply and predictably.

Chapter 2 "Getting Started with Wonderwall" explains how to set up and configure Wonderwall which is also fully interoperable. For further information, refer to Chapter 6, "Interoperability and Details".

Before learning how to use Wonderwall, however, it is necessary to have an elementary understanding of the IIOP protocol itself.

Wonderwall and the IIOP Protocol

The IIOP protocol specifies the way in which CORBA messages are encoded for transmission. In particular, it specifies a universal format for the transmission of operation invocations across the Internet. This makes it possible for clients of one ORB to send operation invocations to any ORB across the Internet, and also to correctly interpret any return values received.

When an IIOP client sends a message to a remote object, it requires an Interoperable Object Reference (IOR) which stores the addressing information for that object. For the IIOP protocol, an IOR will include the following information:

- The name of the host on which the object resides.
- The port it listens to.
- Its object key (a string of bytes identifying the object).

When an IIOP client has the remote object's IOR, it opens a TCP connection to the host and port named therein, and can send and receive messages along this connection. If multiple objects use the same host and port, the client can use the same connection to communicate with the other objects.

The IIOP model is based around two main message types: a *Request* and a *Reply*. Clients send Requests, and servers send Replies. There is also a set of message types used to handle unexpected error conditions or timeouts. Refer to "Reply Message" on page 50 for further information.

In the same way that a filtering router can filter packets based on the packet header, Wonderwall filters incoming Requests based on the following information gleaned from the message header:

- The object key of the object being invoked on, which is used to identify the server it is destined for.
- The name of the operation being invoked.
- The IP address of the client.
- The message type.
- Any IOP Service Contexts.
- The principal of the client's invoker.

Refer to "HTTP Server" on page 19 and Appendix A, "iiopproxy and iortool" (page 85) for further details on the filtering mechanism and how it is specified. The body of Request messages cannot be filtered without knowledge of the Interface Definition Language (IDL) used to define the operations and parameters for each object, so only the message header parameters can be used in a filter.

In the present version of Wonderwall, any Reply messages which pass from the internal server out to the client are not filtered.

The basic component of Wonderwall is the executable `iiopproxy`. This process is intended to run on the bastion host listening for IIOP requests on a specified TCP port. Any requests which arrive on this port from external hosts are filtered so that access can be restricted to certain CORBA objects or operations behind the firewall.

You can control the filtering of packets by editing the configuration file `iiopproxy.cf`. This file allows you to specify a flexible set of rules for either allowing or denying access to certain objects or operations.

Once a given request has been allowed through the firewall, the process `iiopproxy` will forward it to the proper location on the internal network. The `iiopproxy` does this by looking up its own database of IORs which include all the externally accessible CORBA objects.

In the following chapter, "Getting Started with Wonderwall", an example of a configuration file is given and the database of IORs set up so that the firewall can pass on requests to a couple of objects on the internal network.

2

Getting Started with Wonderwall

This chapter introduces basic Wonderwall security by describing how to set up and configure Wonderwall. To achieve this, we define an IDL interface implement a server using Orbix, and develop an OrbixWeb client.

The sample OrbixWeb client developed talks to an OrbixWeb or C++ server. An example configuration file is given and a database of IORs set up so that Wonderwall can pass on requests to objects on the internal network.

Versions of the client application described in this chapter are located in the `grid` demonstration directory of your Wonderwall installation. The server application is implemented in OrbixWeb and its code is located in the same directory as versions of the client application.

Wonderwall is also fully interoperable. Issues associated with interoperability are discussed in Chapter 6 “Interoperability and Details”.

The Grid Application

To illustrate Wonderwall in operation, a simple grid example is considered. This example introduces a `grid` interface whereby a `grid` server and an OrbixWeb client communicate with each other via Wonderwall. It comprises the following components:

- An OrbixWeb Client.

  This client invokes IDL operations in the server via CORBA IIOP protocol.
- A `grid` server—that is, an OrbixWeb/Orbix C++ server.

  This server processes client requests.
- Wonderwall.

  The Wonderwall server acts as a liaison between the outside world and the internal network ensuring that all communications using CORBA over the Internet are secure.

Note: It is assumed that the client, Wonderwall, and server all run on the same host. In a realistic situation these processes would run on three separate hosts.

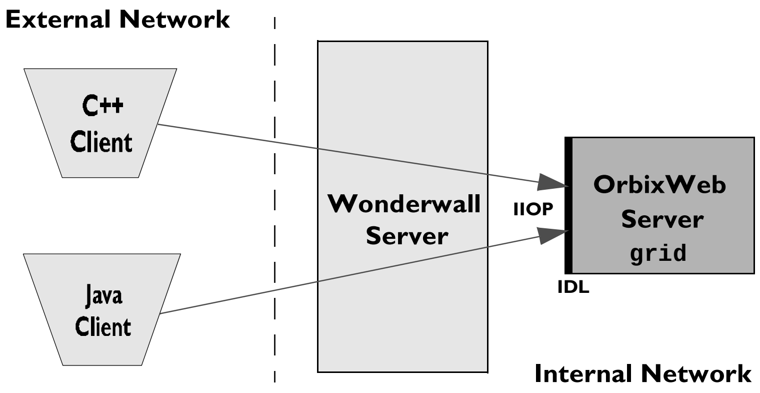


Figure 2.1: *The Grid Application*

The IDL Specification

The first step in writing the grid application is to define the interface to the application objects using the standard CORBA Interface Definition Language (IDL). IDL is a specification that developers use to ensure clients and servers can communicate with each other. The interface (`grid`) to our grid example is defined in IDL as follows:

```
//Grid.idl
// Definition of a 2-D grid.

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

   // width of the grid
   readonly attribute short width;

   // set the element [n,m] of the grid, to value:
   void set(in short n, in short m, in long value);

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

This defines the interface for a two-dimensional grid of long integers whose size is given by the `height` and `width` attributes. Two operations `set()` and `get()` can be invoked to respectively modify or read a single element of the grid.

The details of implementing a grid object need not be considered here. It is assumed that there is a `grid` server which implements at least one grid object. Likewise, it is assumed there is a client that makes use of the object. Both server and client use the IIOP protocol.[1] In this example, the `grid` client represents an external, possibly hostile, process which wishes to use objects in the server. The `grid` server itself is to be protected by Wonderwall.

1. Orbix is shipped with a demo `grid_iiop` which provides an example implementation of the grid using the IIOP protocol.

The OrbixWeb Client

Once the `grid` interface has been implemented, an OrbixWeb client application can be written to access grid objects. A typical client binds to a `grid` server using the Orbix `_bind()` mechanism to connect to an object behind Wonderwall, and subsequently make invocations on that object. It is assumed that the client has been compiled with the relevant options to ensure that it uses the IIOP protocol.[2]

These concepts are illustrated in the following code sample. The Orbix `_bind()` call is used to connect to a grid object behind Wonderwall—invocations are then made on that object.

```
// Java
package gridtest;

import IE.Iona.OrbixWeb._CORBA;
import IE.Iona.OrbixWeb.CORBA.SystemException;

public class Client {
   ...
   public static void main(String args[]) {
      _grid gRef = null;

      try {
1        gRef = gridHelper._bind(“grid1:GridSrv”,
                                 “Host”);
      }
      catch (SystemException se) {
         System.out.println(se.toString());
      }
      ...
   }
};
```

2. This is the default behaviour in OrbixWeb. The `_bind()` operation involves establishing a CORBA IIOP connection between the client and server, and a proxy object is returned. Remote invocations in OrbixWeb occur when normal Java function calls are made on proxies. Refer to the *OrbixWeb Programmer's Guide* for further details.

The code is explained as follows:

1. The `_bind()` call contacts the Wonderwall proxy and establishes an IIOP connection. The first argument to `_bind()` is of the form “`marker:server`”, where *marker* is a string identifying the object within a particular *server*.[3] The second argument “`host`” specifies the *host* where the Wonderwall proxy is running.

   The advantage of using `_bind()` is that the client does not need to have an Interoperable Object Reference (IOR) for `grid1` before making the connection.

For non-OrbixWeb clients or if, for some reason `_bind()` is not used, it is necessary to understand the concepts underlying IORs and the process of *proxification* of IORs. Refer to “Proxification” on page 55 for further information.

OrbixWeb also supports a transparent Wonderwall connection mechanism using the `IT_IIOP_PROXY` and `IT_HTTP_TUNNEL` configuration parameters. Refer to “Using Wonderwall with OrbixWeb” on page 79 for further information.

The Configuration File

Each installation of Wonderwall includes a configuration file that allows you to specify how applications use Wonderwall security. At the heart of Wonderwall’s operation is the Wonderwall security configuration file, `iiopproxy.cf`, which specifies the security policy for your system.

Creating the Wonderwall configuration file `iiopproxy.cf` is the first stage in setting up the firewall. During startup, the file `iiopproxy.cf` is read in by the firewall server `iiopproxy`. Subsequent changes made to `iiopproxy.cf` will affect new clients—any existing client sessions will not be affected by the changes.

Note: The Wonderwall GUI Configuration Tool can also be used to create the Wonderwall security configuration file. Refer to “The Wonderwall GUI Configuration Tool” on page 23 for further information.

3. The identifiers *marker* and *server* are Orbix specific concepts.

For the grid example, a sample Wonderwall configuration file is detailed—refer to “Example iiopproxy.cf File” on page 16. This sample Wonderwall configuration file comprises the following sections:

- Basic Configuration and Ports.
- Object Specifiers.
- Access Control List.

A brief explanation for each line in each section is given. Full explanations of fields, however, can be found in Appendix A on page 85.

Basic Configuration and Ports

In this section of the “Example iiopproxy.cf File” on page 16, lines beginning with a ‘#’ character are comments. Trailing comments on a line are also allowed. Further details include the following:

Line	Explanation
`port 1570`	This port specifies that Wonderwall listens for requests on TCP port 1570.
`orbixd-iiop-port 1571`	This port refers to the port where the Orbix daemon listens for IIOP messages on the internal network. It is essential to specify this port number if you are going to be using the Orbix daemon. Wonderwall needs to know which port the Orbix daemon is listening on, in order to interact with it.
`domain your.domain.com`	This entry gives the DNS domain name of the host where Wonderwall is running.
`log requests replies`	This entry tells Wonderwall to log all IIOP request and reply messages.
`http-port` and `http-files`	These entries are used to configure the optional HTTP server capability of Wonderwall. Refer to “HTTP Server” on page 19 for further information.

Object Specifiers

The next section of the "Example iiopproxy.cf File" on page 16 lists all of the objects which might be made available through Wonderwall. The Wonderwall proxy uses this list to construct an internal table of known objects. The general form of these entries is as follows:

`object` *tag* `[wild` *wildcardflags*`]` *object-specifier*

This entry declares a `tag` which is used to refer to the specified object throughout the configuration file. The optional `wild` field is used to refer to *categories* of objects, rather than a single object, and is discussed in "List of IORs" on page 98. The `object-specifier` can be specified in a number of ways (refer to Appendix A on page 85).

At present, Wonderwall supports four different forms of object-specifier as follows:

Object-specifier	**Definition**
`bind`	An object-specifier beginning with the keyword "`bind`" is used to specify the object using a pseudo-bind syntax (which closely resembles the syntax of `_bind()` as used by a regular OrbixWeb client).
`IOR:`	An object-specifier that begins with the characters "`IOR:`" introduces an IOR coded as a standard CORBA stringified object reference.
`RXR:`	An object-specifier that begins with the characters "`RXR:`" introduces an IOR encoded using the readable-hex-representation.
`/`	An object-specifier that begins with a "`/`" or "`\`" is assumed to be the absolute pathname of a file where the IOR is stored (either in "`IOR:`" or "`RXR:`" format).

All of these forms of object-specifier are explained in detail in "Representations of an IOR" on page 44 and "List of IORs" on page 98.

The `bind` format is the simplest specifier to use. This format requires that Wonderwall is able to contact an Orbix or OrbixWeb daemon in order to locate the server. If using a non-Orbix server, read Chapter 6 "Interoperability and Details" and use one of the three alternative object-specifiers. If using an Orbix or OrbixWeb server, it is possible to use the bind format as given in the "Example iiopproxy.cf File" on page 16. For example:

```
object grid_1 bind("grid1:GridSrv","gridHost") interface grid
```

The pseudo `bind` function has a similar format to `_bind` in the OrbixWeb client. This example specifies an object with marker `grid1`, held by the server named `GridSrv`, found on host `gridHost`. The trailing fields `interface grid` (which must be present) specify that the object is of type `grid`.[4]

The last entry of this section, `allow-unlisted-objects on`, gives you a powerful mechanism for extending the list of known objects. When set to `on` (the default setting), any time a client attempts to access an unlisted object, Wonderwall will automatically update and add the object reference to its table of known objects. This considerably relieves the burden of administration required for a minimal configuration of Wonderwall.

Note: Because an object is automatically listed in this way, this does not mean that the client has permission to connect to the object. That is determined by the Access Control List.

In some high security networks, the administrator can switch this option to `off`.

Access Control List

The last section of the "Example iiopproxy.cf File" on page 16 is the Access Control List. This consists of a list of rules which begin with either one of the keywords `allow` or `deny`. Whenever a request arrives at the Wonderwall server, these rules are checked in sequence until a rule is found which definitely denies access or definitely allows access to the target object.

The first rule given here is `deny servicecontexts *`. A service context is a mechanism which allows extra information to be added to an IIOP request (or

4. It is assumed that the server `GridSrv` has been registered with the Orbix daemon in the usual way.

reply) for use by the CORBA services. In keeping with the firewall philosophy of "anything not expressly permitted is denied," it is considered safer to forbid all requests with a service context attached.

Note: The core specification of CORBA does not make use of service contexts.

The next few rules have a form similar to the following:

```
allow object grid_1 op _get_height
```

This states that the request is allowed if it is to be invoked on object `grid_1` *and* the operation name is `_get_height`. The operation name `_get_height` derives from the attribute name `height`. For every attribute, such as `height`, there are two operation identifiers associated with it: `_get_height` and `_set_height`. If the attribute is declared `readonly`, there will be only one operation, `_get_height`.

The rules applying to the object `grid_2` are specified in a slightly different way, as follows:

```
allow object grid_2 ipaddr 10.23.67.1 op _get_height
```

This stipulates that if the request is to invoke on object `grid_2` *and* the IP address of the invoking host is 10.23.67.1 *and* the operation is `_get_height`, the request is allowed.

The last line of the Access Control List is as follows:

```
allow object grid_2 ipaddr 10.23.67.1 op set log
```

This specifies that the operation `set` is allowed on object `grid_2` when the host has an IP address 10.23.67.1. In addition, the final keyword `log` specifies that all such requests should be logged (in this example, the logging is superfluous since all incoming and outgoing requests and replies will be logged anyway).

It is important to understand how Wonderwall parses the Access Control List. It starts at the beginning of the list, reading each rule in sequence, until it finds a rule which unambiguously allows or denies a request. Wonderwall then stops and does not read any more rules. This approach makes it easy to predict how Wonderwall will interpret the Access Control List.

A non-intuitive side effect of this algorithm is that it is permissible to have contradictory rules. The resolution of any conflict is simple: the first rule takes precedence.

Example iiopproxy.cf File

```
#########################################################
#######
# A sample Wonderwall configuration file.
port 1570
orbixd-iiop-port 1571 # Use the Orbix IIOP port.
domain your.domain.com
log requests replies
http-port 0
http-files /
#######################################################
# Database of Objects.
object grid_1 bind("grid1:GridSrv","gridHost") interface grid
object grid_2 bind("grid2:GridSrv","gridHost") interface grid
allow-unlisted-objects on
#######################################################
# On to the access control list!
# Disallow any IOP Service Contexts, at least until we need
# them... who knows what could be put in here?
#
deny servicecontexts *
# Allow general access to grid_1,
# except for the "set" operation.
#
allow object grid_1 op _get_height
allow object grid_1 op _get_width
allow object grid_1 op get
# Allow access to grid_2 from our link to a semi-trusted
# network, but log any "set" operations.
#
allow object grid_2 ipaddr 10.23.67.1 op _get_height
allow object grid_2 ipaddr 10.23.67.1 op _get_width
allow object grid_2 ipaddr 10.23.67.1 op get
allow object grid_2 ipaddr 10.23.67.1 op set log
# File ends here -- if the message has not matched a rule
# until now, it will be denied automatically.
#######################################################
```

Factory Objects

One of the interesting features of CORBA is that it allows you to pass back and forth object references inside Request or Reply messages, where they might appear either as parameters or return values. This provides a powerful mechanism for clients to obtain references to new objects. The term *Factory Interface* is applied to any interface which can create a new object and return a reference to this object. Individual instances of a Factory Interface are known as *Factory Objects*.

Consider the following example of a Factory Interface:

```
// IDL
typedef string MarkerString;

interface GridFactory {
   // Make an object of type 'grid'
   // and return the object's marker.
   MarkerString makeGrid();
};
```

This particular interface, because it returns an Orbix marker instead of an Interoperable Object Reference, is an Orbix specific example of a Factory. The marker gives an OrbixWeb client enough information to find the object using the pseudo `_bind()` mechanism.

The existence of Factory Objects poses special problems for the Wonderwall administrator. Object level security is based on the idea that a finite number of objects are listed and it is known whether they can safely be accessed from outside. A Wonderwall administrator must consider not only whether the Factory Object is safe, but also whether objects created by the Factory can be considered safe. This also applies to the related idea of *Finder Objects*, which do not actually create new objects, but could return object references not listed in the Wonderwall configuration.

Nevertheless, there are compelling reasons for making use of both Factory and Finder objects. Consider, for example, accessing a database through a firewall that represents its records in the form of CORBA objects. Because the number of objects is likely to be considerable, it would be impractical to list them all in the Wonderwall configuration file. A Finder object is a more practical way of providing access to the records.

Assume that there is a given Factory Object, such as `GridFactory`, which needs to be used through the firewall. This implies that Wonderwall must provide a means of accessing both the Factory Object and objects created by that Factory.

Wonderwall provides the following form of entry in the configuration file for specifying Factories[5]:

```
server tag object-specifier
```

The server keyword is used to define a *tag* which refers to all of the objects on a particular server. The object given by the *object-specifier* refers to an object which can be used to make the initial connection to the server. Usually this will be the factory object. For example, the `GridFactory` object can be listed as follows:

```
server gridFactory \
  bind(":FactorySrv", "gridHost") interface
GridFactory
```

The tag `gridFactory` can now be used to refer to all objects on the `FactorySrv` server, irrespective of marker, or interface name. Therefore a line such as the following in the Access Control List can be used to give away access to all objects on that server:

```
allow object gridFactory
```

Note: Because the object type of the tag `gridFactory` is wildcarded, it is legal to specify a rule such as the following:
`allow object gridFactory operation _get_height`

The operation `_get_height` does not appear in the interface `GridFactory`, only in interface `grid`. The appearance of `interface GridFactory`, in the previous object specifier, is just a placeholder. Any operation at all, from any interface, can be specified in a rule with a server tag.

5. This is equivalent to the following construction:
`object` *tag* `wild marker, ifmarker` *object-specifier*
The `server` keyword is provided as a convenience for defining Factory objects (refer to Appendix B).

Note: Wonderwall's support for factories is dependent on using Orbix or OrbixWeb objects, as it needs to understand the object key format. For more details on the object key format, refer to "Orbix/OrbixWeb Object Key Format" on page 43.

HTTP Server

The Wonderwall proxy normally listens for all IIOP messages on a single dedicated port. It monitors this port and redistributes IIOP Request messages to servers behind the firewall.

However, an IIOP port is not yet a standard feature of most firewalls. Until this port becomes established in client-side firewalls, it will be necessary to use *HTTP tunnelling* to smuggle IIOP messages through the HTTP port.

This approach requires a HTTP server. A HTTP server is required to recognise that some HTTP messages can contain data which is meant to be interpreted as an IIOP message. For this reason, the Wonderwall proxy has had the full functionality of a HTTP server added to it.

This functionality of Wonderwall is illustrated in Figure 2.2 on page 20. The process `iiopproxy` is capable of listening on two ports: one of these is a dedicated IIOP port and the other is a HTTP port (usually port 80).

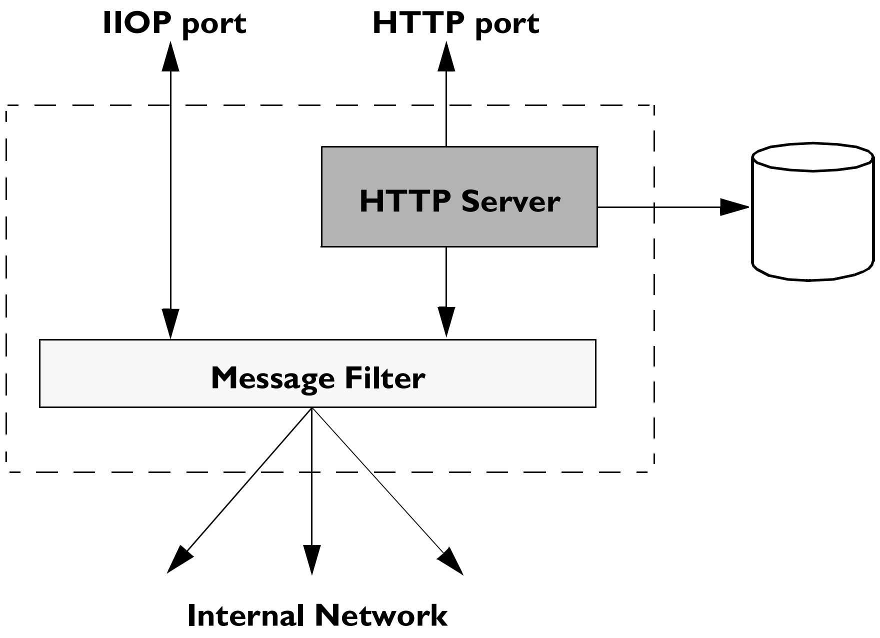


Figure 2.2: *Internal Architecture of the Wonderwall Proxy Server*

When `iiopproxy` listens on the HTTP port, it functions as a full-function HTTP server. Any normal HTTP requests that arrive will cause it to search a designated directory, and return a copy of the requested Web page (if it can be found). However, this HTTP server also has the intelligence to recognise when a tunnelled IIOP message arrives via HTTP. In such a case, it extracts the IIOP message and passes it on to the IIOP gateway.

It does not matter to the gateway whether an IIOP message arrives through the dedicated port or by way of HTTP. The message is still subject to the same filtering mechanism regulated by the configuration file, as described in "The Configuration File" on page 11.

The configuration of the HTTP server only requires two parameters to be set in the configuration file. These are as follows:

```
http-port port
http-files directory
```

The `http-port` is used to set the *port* where the `iioproxy` listens for HTTP requests. The keyword `http-files` is used to specify the *directory* where files can be retrieved to service ordinary HTTP requests.

If you specify `http-port 0`, then HTTP functionality is not enabled and `iioproxy` will listen only on the dedicated IIOP port for ordinary IIOP messages.

Logging Output

The log from the Wonderwall server `iiopproxy` is sent by default to the standard output. Typically, the user will redirect this output to a log file. It is possible to specify what goes into the log file by editing the configuration file and Wonderwall gives you quite an amount of flexibility in this respect. In the “Example iiopproxy.cf File” on page 16, the line `log requests replies` ensures that all IIOP requests and replies passing in or out through Wonderwall will be logged. The log essentially records all the information available in the request or reply headers.

The logged output, when an IIOP message is forwarded, generally takes the following format:

```
forwarded: <client> -> <servername>: [Message v1.x: <size>
            bytes: Request <request id>, op [ObjectKey "<object
            key>"]::[<operation>] from "<principal>", respond?
            <response expected>]

forwarded: <client> <- <servername>: [Message v1.x: <size>
            bytes: Reply <request id>, reply status <reply
            status>]
```

Consider a sample log output generated by a client invoking on the grid via Wonderwall:

```
IIOP connection opened: [ultra:64023]
starting server for activated object “grid”
forwarded: [ultra:64023] -> [grid]: [Message v1.0, 82 bytes:
             Request 0, op [ObjectKey
             “RXR::%5cultra.dublin.iona.ie:grid:0::IR:grid_”]
             ::[_get_height] from “RXR:jmason”, respond? y]
forwarded: [ultra:64023] <- [grid]: [Message v1.0, 14 bytes:
             Reply 0, reply status NO_EXCEPTION]
```

The logging facility also allows the full request and reply bodies to be logged. The rules for the Access Control List also let you dictate that requests or replies be logged only in specific circumstances. For full details of the logging options available, refer to Appendix A on page 85.

3

The Wonderwall GUI Configuration Tool

The Wonderwall GUI Configuration Tool allows you to change the default security configuration settings for Wonderwall using a graphical user interface. The GUI Configuration Tool edits the `iiopproxy.cf` *file which stores the configuration settings for your Wonderwall installation. This chapter describes how to use the Wonderwall GUI Configuration Tool, and provides descriptions of the available parameters.*

Default security configuration settings may need to be changed for a variety or reasons, including:

- Enabling or disabling parts of Wonderwall functionality.
- Altering the use of specific port numbers.

The Wonderwall Configuration Tool can be used to make these configuration changes.

The iiopproxy.cf File

The `iiopproxy.cf` file holds configuration information for Wonderwall. It is located in the Wonderwall installation directory. A default `iiopproxy.cf` file is created by the installation process. For further information on the `iiopproxy.cf` file, refer to “The Configuration File” on page 11.

Starting the Wonderwall Configuration Tool

There are two ways to start the Wonderwall GUI Configuration Tool.

- To start the GUI Configuration Tool from the command line, enter the following:

`wwconfig`	UNIX or Windows NT 4.0

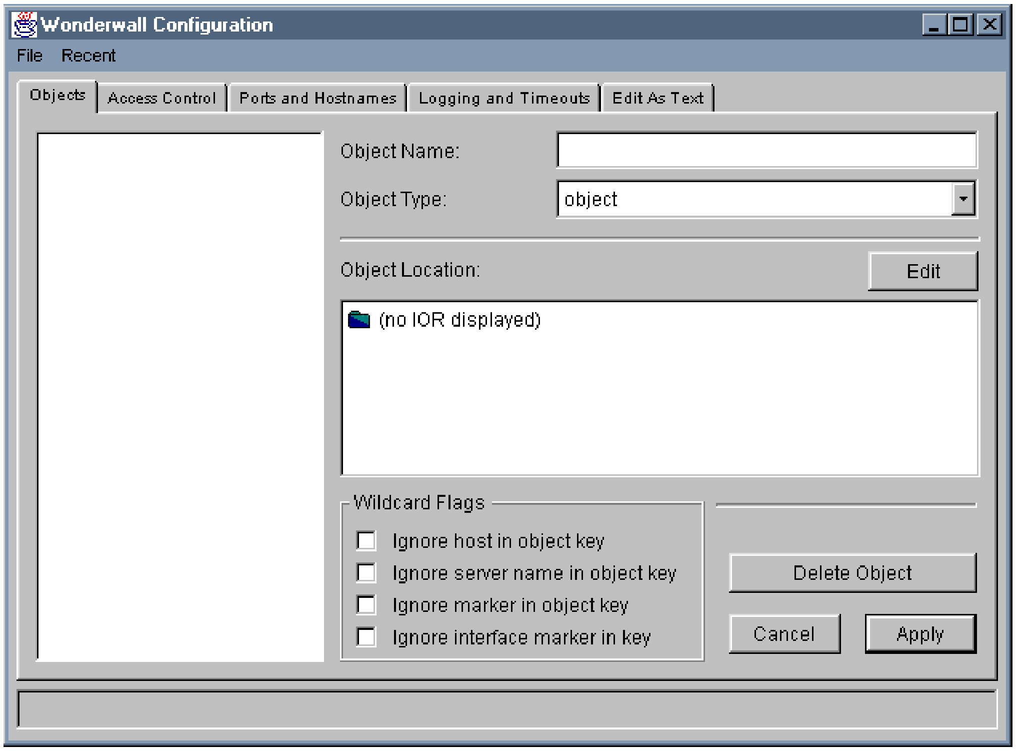
- To start the GUI Configuration Tool from the Windows **Start** menu:
  i. Select the Windows **Programs** menu
  ii. Select the **Wonderwall** sub-menu.
  iii. Select the **Edit Configuration File** from the options displayed.

When the GUI Configuration Tool is invoked, it automatically loads its settings from the default location.

The GUI Configuration Tool startup window is shown in Figure 3.1 on page 25.

GUI Configuration Tool Main Window

The GUI Configuration Tool main startup window consists of a number of tabs, each containing configuration information for different areas of Wonderwall functionality. To load a configuration file, select **File→Open Config File**.



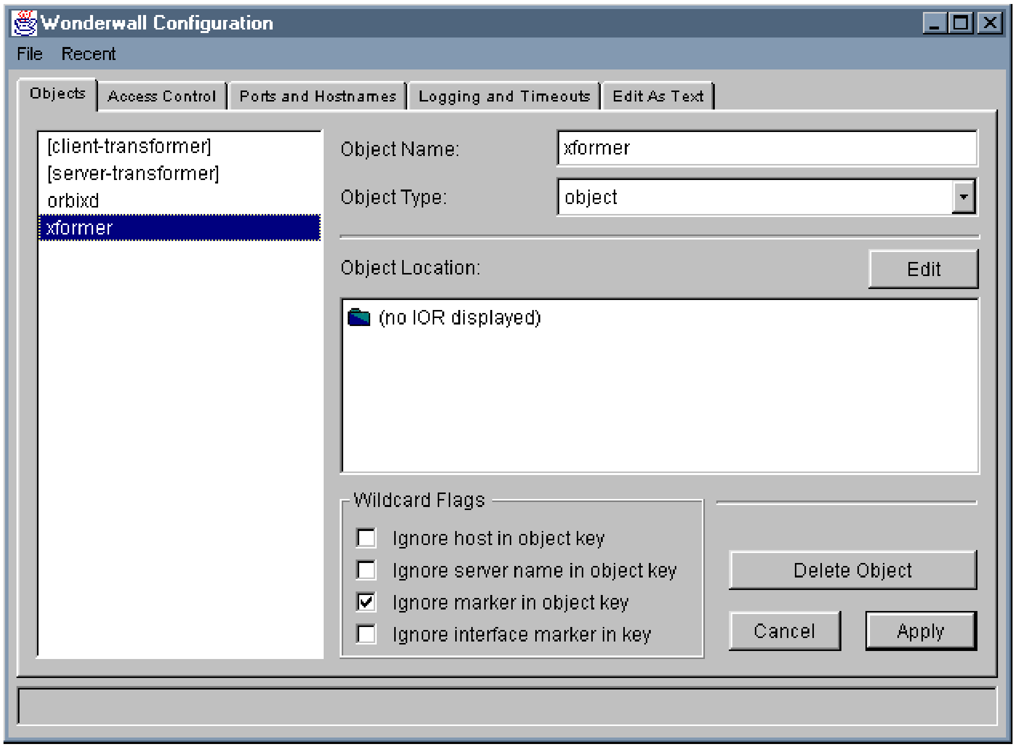
Object Name:
Object Type: object
Object Location:
(no IOR displayed)
Wildcard Flags
- [ ] Ignore host in object key
- [ ] Ignore server name in object key
- [ ] Ignore marker in object key
- [ ] Ignore interface marker in key

Figure 3.1: *Wonderwall GUI Configuration Tool Main Screen*

To access the options in a specific tab, select that tab. The **Objects** tab deals with all the objects which might be made available through Wonderwall. The **Access Control** tab deals with the access control list which either denies access or allows access to the target object. The **Ports and Hostnames** tab deals with the ports and hostnames on which Wonderwall listens and runs, respectively.

Object Specifier Window

The **Objects** window allows you to select or specify an object which can be made available through Wonderwall. To make an object avaialable, select an object from the list of objects displayed or enter the **Object Name** and **Object Type** in the corresponding text fields.



[client-transformer]
[server-transformer]
orbixd
xformer

Object Name: xformer

Object Type: object

Object Location:

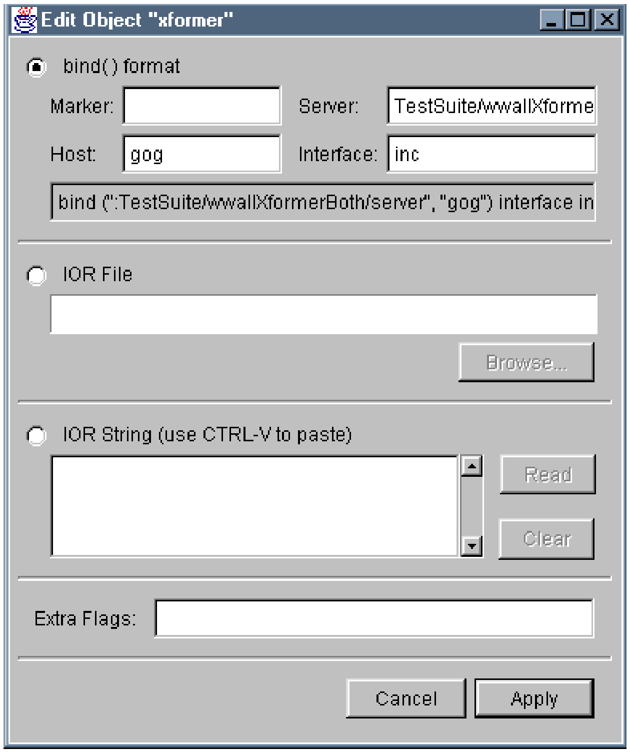
(no IOR displayed)

Wildcard Flags

- [ ] Ignore host in object key
- [ ] Ignore server name in object key
- [x] Ignore marker in object key
- [ ] Ignore interface marker in key

Figure 3.2: *Wonderwall GUI Configuration Tool Object Specifier Window*

After you select the object, select the **Edit** button. This allows you to load the object's IOR using the **Edit** window (Figure 3.3 on page 27). Wonderwall supports four different forms of object-specifier—refer to "Object Specifiers" on page 13 for further information.


Edit Object "xformer"

[✓ bind() format]

Marker:

Server: TestSuite/wwallXforme

Host: gog

Interface: inc

bind (":TestSuite/wwallXformerBoth/server", "gog") interface in

IOR File

IOR String (use CTRL-V to paste)

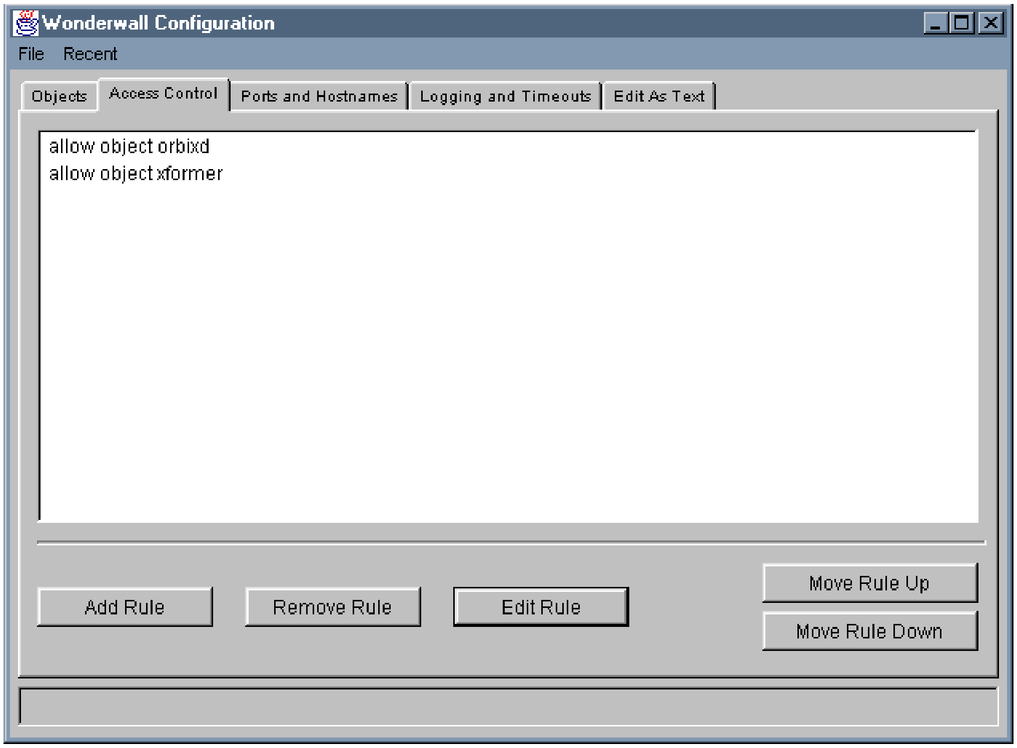
Extra Flags:

Figure 3.3: *Loading IORs*

For further information on object specifiers, refer to “Representations of an IOR” on page 44 and “List of IORs” on page 98.

Access Control List Window

The **Access Control** window lists all the access control list entries which ultimately control access to servers, objects and clients, in order of importance. To change the order of the `allow` and deny rules, select the **Move Rule Up** and **Move Rule Down** buttons.

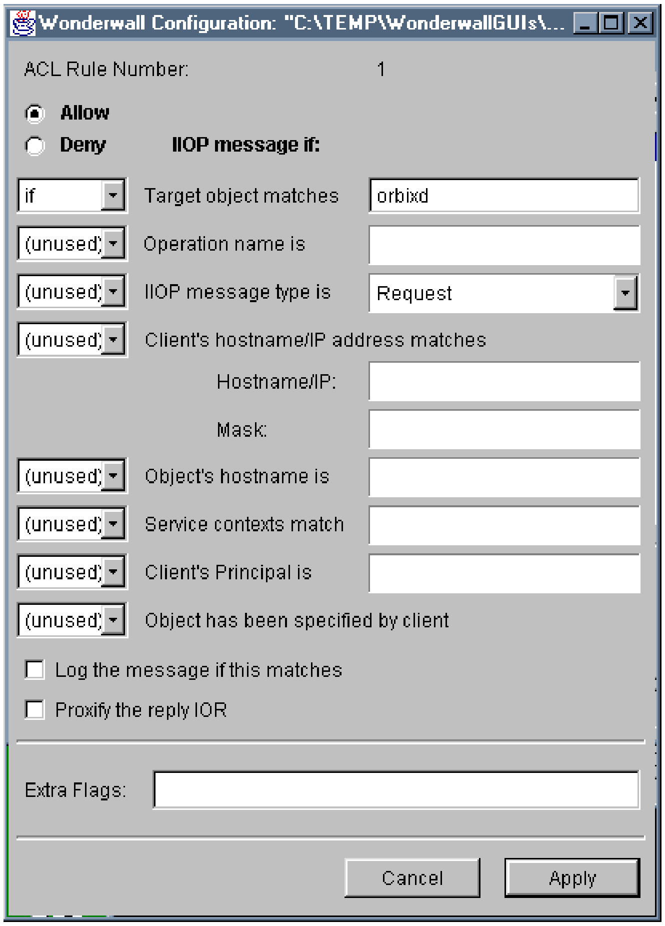


```
allow object orbixd
allow object xformer
```

Figure 3.4: *Wonderwall GUI Configuration Tool Access Control List Window*

From this window it is also possible to add a new rule, remove a rule from the list, and edit an existing rule. For example, Figure 3.5 on page 29 displays ACL rule number one which is as follows:

Allow IIOP message if the target object matches `orbixd`.



ACL Rule Number: 1

(● Allow) ○ Deny — IIOP message if:

if	Target object matches	orbixd
(unused)	Operation name is	
(unused)	IIOP message type is	Request
(unused)	Client's hostname/IP address matches	
	Hostname/IP:	
	Mask:	
(unused)	Object's hostname is	
(unused)	Service contexts match	
(unused)	Client's Principal is	
(unused)	Object has been specified by client	

☐ Log the message if this matches

☐ Proxify the reply IOR

Extra Flags:

Figure 3.5: *The Add Rule/Edit Rule Window*

Note that (unused) here means that the setting is not used to determine whether access is allowed or denied.

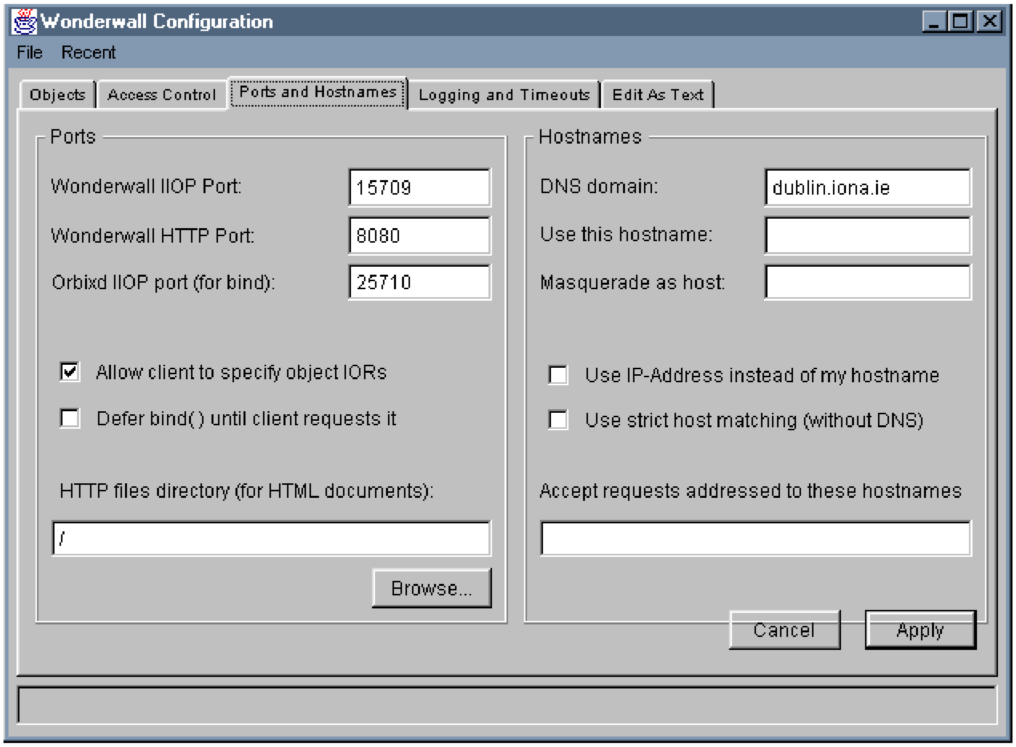
For further information on access control lists, refer to "Access Control List" on page 14 and page 100.

Ports and Hostnames Window

The ports available to launched servers can be modified in the **Ports and Hostnames** window of the GUI Configuration Tool. It is generally only necessary to change default settings if there is a clash with ports used by an existing system, or if a directory needs to be specified to enable HTTP service.

Parameters set in this window must map directly to existing configuration parameters.

Ports must be specified—everything else is optional.



Wonderwall IIOP Port: 15709

Wonderwall HTTP Port: 8080

Orbixd IIOP port (for bind): 25710

[✓] Allow client to specify object IORs

[ ] Defer bind( ) until client requests it

HTTP files directory (for HTML documents): /

DNS domain: dublin.iona.ie

Use this hostname:

Masquerade as host:

[ ] Use IP-Address instead of my hostname

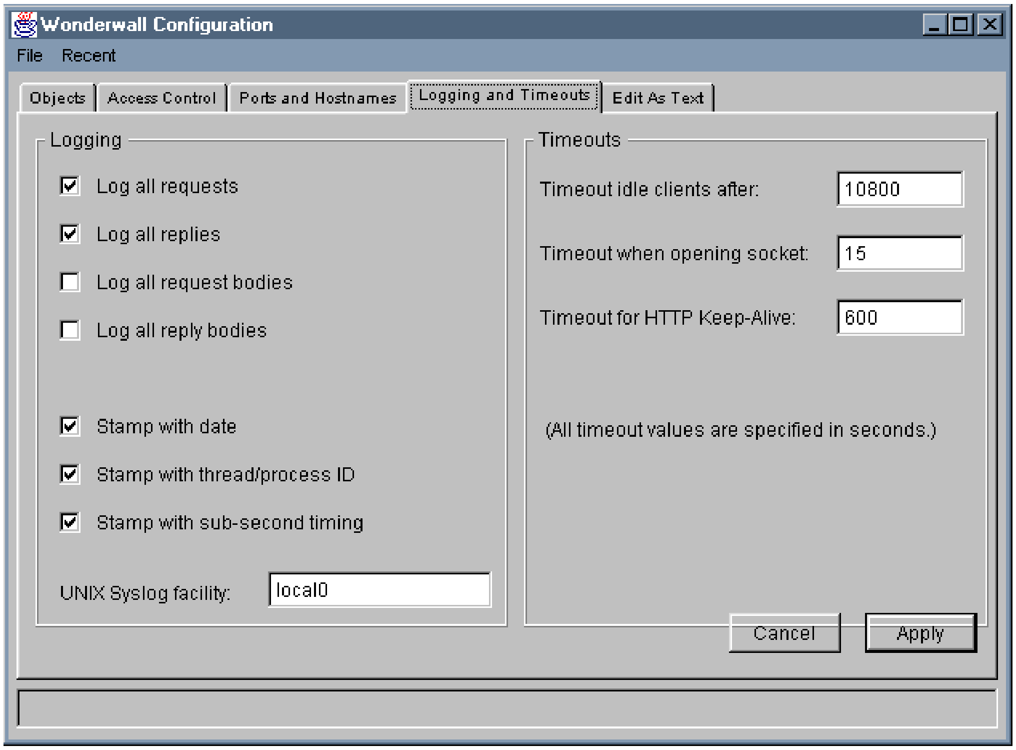
[ ] Use strict host matching (without DNS)

Accept requests addressed to these hostnames

Figure 3.6: *Wonderwall GUI Configuration Tool Ports and Hostnames Window*

Logging and Timeouts Window

The **Logging and Timeouts** window allows you to customise all logging and timeouts for your system.



- [✓] Log all requests
- [✓] Log all replies
- [ ] Log all request bodies
- [ ] Log all reply bodies
- [✓] Stamp with date
- [✓] Stamp with thread/process ID
- [✓] Stamp with sub-second timing

UNIX Syslog facility: local0

Timeout idle clients after: 10800

Timeout when opening socket: 15

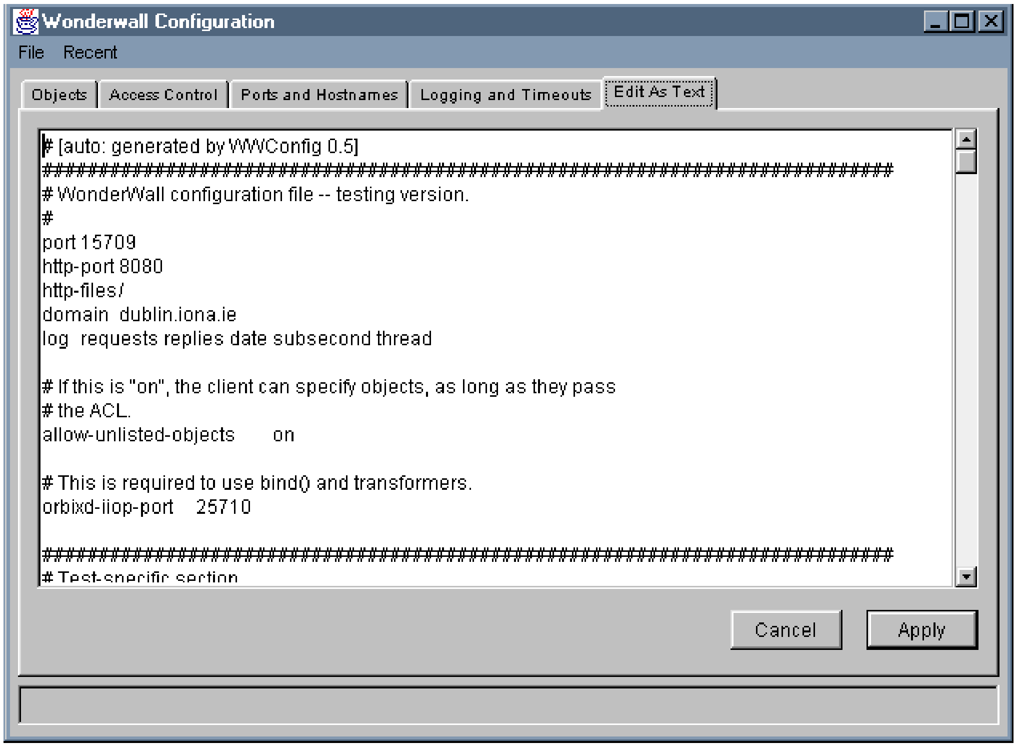
Timeout for HTTP Keep-Alive: 600

(All timeout values are specified in seconds.)

Figure 3.7: *Wonderwall GUI Configuration Tool Logging and Timeouts Window*

Edit As Text Window

The configuration file can be edited by hand from the **Edit As Text** window.



```
# [auto: generated by WWConfig 0.5]
###############################################################################
# WonderWall configuration file -- testing version.
#
port 15709
http-port 8080
http-files /
domain  dublin.iona.ie
log  requests replies date subsecond thread

# If this is "on", the client can specify objects, as long as they pass
# the ACL.
allow-unlisted-objects      on

# This is required to use bind() and transformers.
orbixd-iiop-port    25710

###############################################################################
# Test-specific section
```

Figure 3.8: *Wonderwall GUI Configuration Tool Edit As Text Window*

4

The Wonderwall Log Analysis Viewer

The Wonderwall Log Analysis Viewer allows you to view log files using a graphical user interface. This chapter describes how to use the Wonderwall Log Analysis Viewer and provides descriptions of options available.

The Wonderwall iiopproxy Server

The log from the Wonderwall server `iiopproxy` is sent by default to the standard error file descriptor, `stderr`. Using the `iiopproxy` process which implements Wonderwall, this output is typically redirected to a specified named log file using the `-log logfile` switch. For further information on `iiopproxy`, refer to Appendix A.

It is possible to specify exactly what goes into the log file by editing the configuration file, `iiopproxy.cf`. For further information, refer to "The Configuration File" on page 11.

Starting the Wonderwall Log Analysis Viewer

There are two ways to start the Wonderwall Log Analysis Viewer.

- To start the Wonderwall Log Analysis Viewer from the command line, enter the following command:

`wwlogviewer`	UNIX or Windows NT 4.0

- To start the Wonderwall Log Analysis Viewer from a menu (Windows NT 4.0), do the following:

  iv. Select the Windows **Programs** menu.

  v. Select the **Wonderwall** sub-menu.

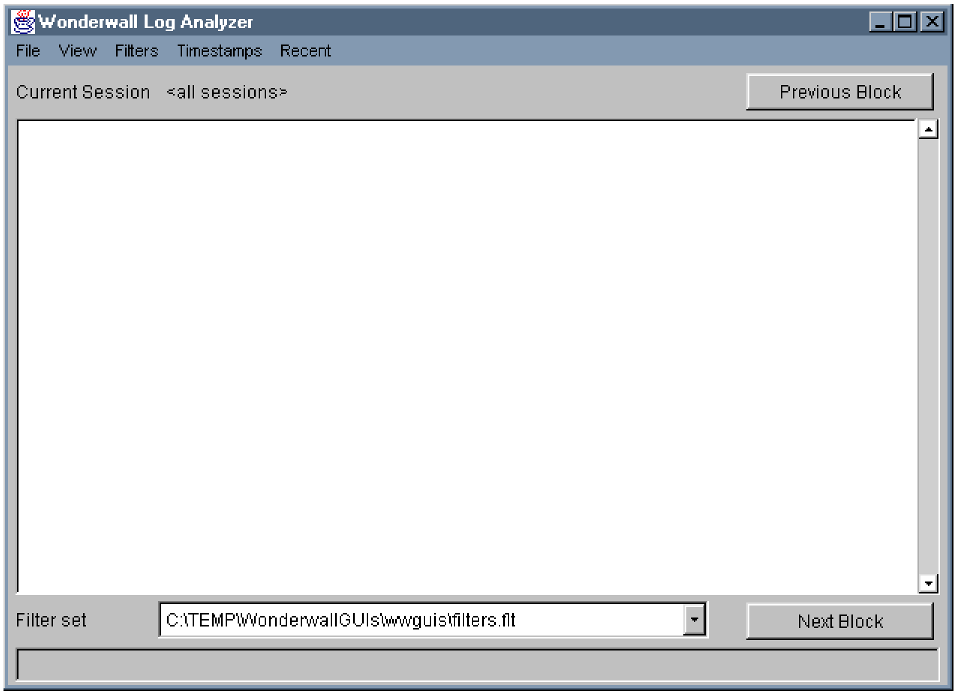
  vi. Select **View Wonderwall Log** from the options displayed.

When the Log Analysis Viewer is invoked, it automatically loads its settings from the default location.

Log Analysis Viewer Main Window

The Log Analysis Viewer main startup window consists of a number of menus each containing information for different areas of log analysis—see Figure 4.1 on page 35.

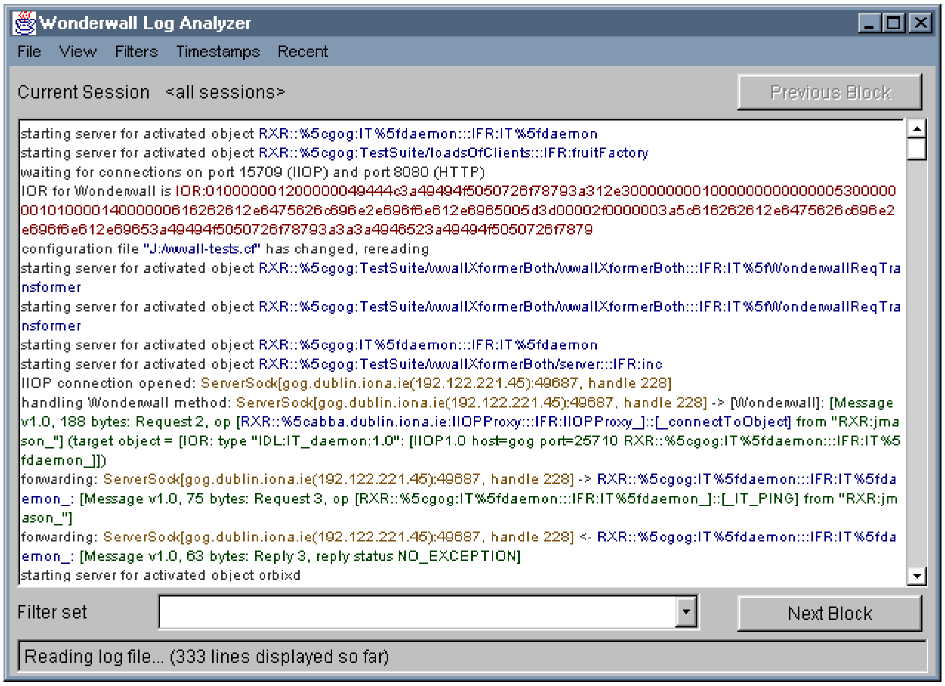
Menu	Description
File	To load a log file, select **File**→**Open Log File**. All sessions in the log file are loaded by default—see Figure 4.2 on page 36.
View	To view a log file in segments, select **View**→**Show Session**— select a session, then select **View Session**. The name of the current session is displayed in the title bar.
Filters	This menu allows you to open, save, and edit filter sets—see Figure 4.3 on page 37.
Timestamps	This menu allows you to switch on/off the following options: timestamps, thread ID, process ID, and log message type—see Figure 4.4 on page 38.
Recent	This menu displays the most recent logs opened.



Current Session <all sessions>

Filter set C:\TEMP\WonderwallGUIs\wwguis\filters.flt

Figure 4.1: *Wonderwall Log Analysis Viewer Startup Window*



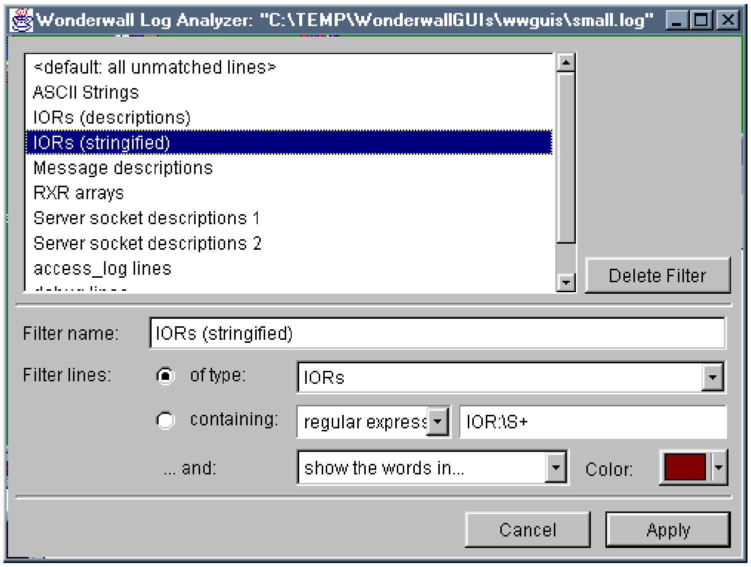
```
starting server for activated object RXR::%5cgog:IT%5fdaemon:::IFR:IT%5fdaemon
starting server for activated object RXR::%5cgog:TestSuite/loadsOfClients:::IFR:fruitFactory
waiting for connections on port 15709 (IIOP) and port 8080 (HTTP)
IOR for Wonderwall is IOR:010000001200000049444c3a49494f5050726f78793a312e3000000001000000000000005300000
00101000014000000616262612e6475626c696e2e696f6e612e6965005d3d00002f0000003a5c616262612e6475626c696e2
e696f6e612e69653a49494f5050726f78793a3a3a4946523a49494f5050726f7879
configuration file "J:/wwall-tests.cf" has changed, rereading
starting server for activated object RXR::%5cgog:TestSuite/wwallXformerBoth/wwallXformerBoth:::IFR:IT%5fWonderwallReqTra
nsformer
starting server for activated object RXR::%5cgog:TestSuite/wwallXformerBoth/wwallXformerBoth:::IFR:IT%5fWonderwallReqTra
nsformer
starting server for activated object RXR::%5cgog:IT%5fdaemon:::IFR:IT%5fdaemon
starting server for activated object RXR::%5cgog:TestSuite/wwallXformerBoth/server:::IFR:inc
IIOP connection opened: ServerSock[gog.dublin.iona.ie(192.122.221.45):49687, handle 228]
handling Wonderwall method: ServerSock[gog.dublin.iona.ie(192.122.221.45):49687, handle 228] -> [Wonderwall]: [Message
v1.0, 188 bytes: Request 2, op [RXR::%5cabba.dublin.iona.ie:IIOPProxy:::IFR:IIOPProxy_]::[_connectToObject] from "RXR:jma
son_"] (target object = [IOR: type "IDL:IT_daemon:1.0": [IIOP1.0 host=gog port=25710 RXR::%5cgog:IT%5fdaemon:::IFR:IT%5
fdaemon_]])
forwarding: ServerSock[gog.dublin.iona.ie(192.122.221.45):49687, handle 228] -> RXR::%5cgog:IT%5fdaemon:::IFR:IT%5fda
emon_: [Message v1.0, 75 bytes: Request 3, op [RXR::%5cgog:IT%5fdaemon:::IFR:IT%5fdaemon_]::[_IT_PING] from "RXR:jm
ason_"]
forwarding: ServerSock[gog.dublin.iona.ie(192.122.221.45):49687, handle 228] <- RXR::%5cgog:IT%5fdaemon:::IFR:IT%5fda
emon_: [Message v1.0, 63 bytes: Reply 3, reply status NO_EXCEPTION]
starting server for activated object orbixd
```

Reading log file... (333 lines displayed so far)

Figure 4.2: *Loaded Log File*

Filters

All the filters defined are listed in the **Edit Filter Set** window. From this window, you can define how you want each filter to appear in the log file. For example, Figure 4.3 defines an `IORs(`*`stringified`*`)` filter containing a regular expression of the string pattern. `IOR:\S+` is the regular expression of the string pattern to search for. If a line matches this filter, it highlights the words (that is, `show the words in...`) in the log file in the specified colour.


Wonderwall Log Analyzer: "C:\TEMP\WonderwallGUIs\wwguis\small.log"
<default: all unmatched lines>
ASCII Strings
IORs (descriptions)
IORs (stringified)
Message descriptions
RXR arrays
Server socket descriptions 1
Server socket descriptions 2
access_log lines
Delete Filter
Cancel
Apply

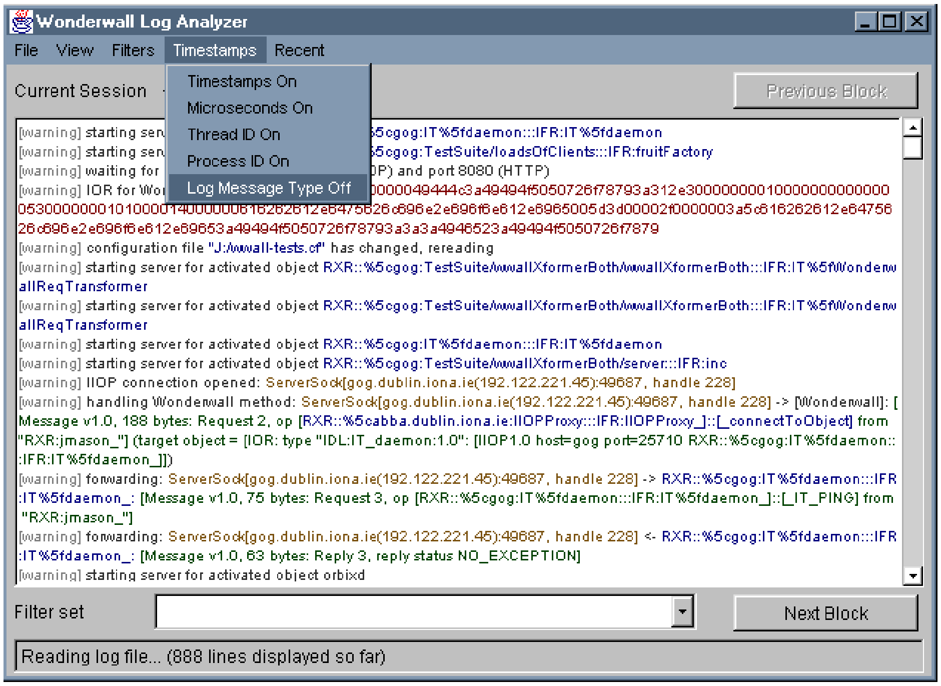
Filter name: IORs (stringified)
Filter lines: (● of type: IORs) ○ containing: regular express IOR:\S+
... and: show the words in... Color: 🟥

Figure 4.3: *The Edit Filter Set Window*

Timestamps

The following options may be switched on/off from the **Timestamps** menu:

Timestamps	Selecting **Timestamps On** will show the date and time of the log. For example: `[1998/03/27 12:40:49]`
Microseconds	Selecting **Microseconds On** will display microseconds. For example: `[.000150]`
Thread ID	Selecting **Thread ID On** will display the thread ID number. For example: `[thread 240]`
Process ID	Selecting **Process ID On** will display, for example: `[pid 1972]`
Log Message Type	Selecting **Log Message Type On** will display the Log message type. For example: `[warning]`



```
[warning] starting serv                              5cgog:IT%5fdaemon:::IFR:IT%5fdaemon
[warning] starting serv                              5cgog:TestSuite/loadsOfClients:::IFR:fruitFactory
[warning] waiting for                                P) and port 8080 (HTTP)
[warning] IOR for Wor                                00000049444c3a49494f5050726f78793a312e300000000100000000000000
053000000001010000140000000616262612e64756... 26c696e2e696f6e612e6965005d3d00002f0000003a5c616262612e64756
26c696e2e696f6e612e69653a49494f5050726f78793a3a3a4946523a49494f5050726f7879
[warning] configuration file "J:/wwall-tests.cf" has changed, rereading
[warning] starting server for activated object RXR::%5cgog:TestSuite/wwallXformerBoth/wwallXformerBoth:::IFR:IT%5fWonderw
allReqTransformer
[warning] starting server for activated object RXR::%5cgog:TestSuite/wwallXformerBoth/wwallXformerBoth:::IFR:IT%5fWonderw
allReqTransformer
[warning] starting server for activated object RXR::%5cgog:IT%5fdaemon:::IFR:IT%5fdaemon
[warning] starting server for activated object RXR::%5cgog:TestSuite/wwallXformerBoth/server:::IFR:inc
[warning] IIOP connection opened: ServerSock[gog.dublin.iona.ie(192.122.221.45):49687, handle 228]
[warning] handling Wonderwall method: ServerSock[gog.dublin.iona.ie(192.122.221.45):49687, handle 228] -> [Wonderwall]: [
Message v1.0, 188 bytes: Request 2, op [RXR::%5cabba.dublin.iona.ie:IIOPProxy:::IFR:IIOPProxy_]::[_connectToObject] from
"RXR:jmason_"] (target object = [IOR: type "IDL:IT_daemon:1.0": [IIOP1.0 host=gog port=25710 RXR::%5cgog:IT%5fdaemon::
:IFR:IT%5fdaemon_]])
[warning] forwarding: ServerSock[gog.dublin.iona.ie(192.122.221.45):49687, handle 228] -> RXR::%5cgog:IT%5fdaemon:::IFR
:IT%5fdaemon_: [Message v1.0, 75 bytes: Request 3, op [RXR::%5cgog:IT%5fdaemon:::IFR:IT%5fdaemon_]::[_IT_PING] from
 "RXR:jmason_"]
[warning] forwarding: ServerSock[gog.dublin.iona.ie(192.122.221.45):49687, handle 228] <- RXR::%5cgog:IT%5fdaemon:::IFR
:IT%5fdaemon_: [Message v1.0, 63 bytes: Reply 3, reply status NO_EXCEPTION]
[warning] starting server for activated object orbixd
```

Reading log file... (888 lines displayed so far)

Figure 4.4: *Timestamps Options*

5

IORs and IIOP

Wonderwall provides firewall security for applications that communicate using the IIOP protocol. An understanding of the IIOP protocol, as detailed in this chapter, is therefore indispensable for the proper use of Wonderwall and will help you to appreciate the issues which affect security. The CORBA interoperability specification defines both the mechanism by which clients establish communication with a server, and the details of message formats and data coding.

The following issues are addressed in this chapter:

- IOR: The key concept which CORBA uses to enable clients to connect to servers is the Interoperable Object Reference (IOR) as discussed in "IOR Format" on page 40.
- IIOP: The message formats and data coding are discussed in "Internet Inter-ORB Protocol (IIOP)" on page 46.

In terms of security implications for the client side, IIOP is not another Java. It does not download executables onto the client machine and it is quite benign. It provides a protocol which enables a client to contact a remote server and call remote functions on this server. Data can pass between client and server, in the form of parameters, however nothing is sent by the server to be executed on the client side.

The server, on the other hand, is in need of some protection because it allows clients to remotely invoke operations which run on the server's host. Wonderwall provides protection for servers which might expose security loopholes and it also restricts access to certain operations which the server does not wish to make available to remote clients.

IOR Format

To identify objects in a distributed object system, CORBA uses the concept of an *object reference*. Once an application has an object reference, it has all the information it needs to connect to the object and make remote invocations on the object's methods.

The notion of an object reference is an abstract one. To the application CORBA programmer it can be represented simply as a C++ pointer. Individual ORB vendors can have their own proprietary representation of an object reference.

However, as part of the infrastructure for an interoperability protocol, CORBA also specifies a universal format for object references known as the Interoperable Object Reference (IOR). This enables the information about an object reference to be either stored or communicated directly to clients in a form which is universally understood. All ORB vendors are required to support this form of object reference.

The information encoded in an IOR (as used in conjunction with the TCP/IP protocol) consists of the following pieces of information:

- The type of the object.

  The type of the object is equivalent to the name of the IDL interface which is used to define the object. For example, in "The IDL Specification" on page 9, an IDL interface is defined for objects of type `grid`.
- The host where the object can be found.
- The port number of the server for that object.

  The host and port together give us the connection information required to contact the server.
- An object key (a string of bytes identifying the object).

  The object key is used by the server itself to locate the object.

Figure 5.1 outlines the format of an IOR in great detail giving a schematic view of the information held in an IOR. The upper part of Figure 5.1 shows the overall format of an IOR as follows:

- It begins with the string `type_id` which gives the type of the object, equivalent to the name of the interface defining the object[1].
- A sequence of profiles preceded by a `profile_count` follows. Two profiles are shown preceded by a `profile_count` of 2.

  A profile contains essentially all the information which is needed to find an object. The facility to specify more than one profile in an IOR is a useful feature which will allow future extensions to the use of IORs. For example, an IOR can specify a number of possible locations for an object. If a client does not succeed in connecting to the location specified in the first profile, the client can try the next profile in the sequence instead. Wonderwall supports the use of IORs with multiple profiles.

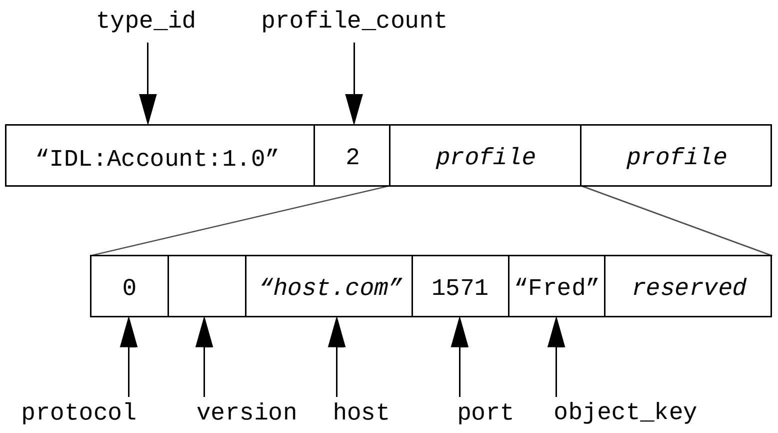


Figure 5.1: *The Format of an Interoperable Object Reference and Profile*

1. To be precise this field holds the `RepositoryId` for the type of object.

The lower part of Figure 5.1 on page 41 shows the details contained in a single profile as follows:

- The connection information stored in a profile is specific to a particular underlying protocol. For this reason the first field is a `protocol_tag`. In this example, the tag value is zero to indicate a TCP/IP transport protocol.
- This is followed by the `Version` field which consists of a `major` and a `minor` version number.
- The next two fields provide the `host` and IP `port` needed to establish communication with the remote server.
- The `object_key` is a field which will be used by the remote server to locate the object being accessed. There can also be additional fields at the end but these are currently not used and are reserved for future expansions to the protocol.

  It may seem surprising that the format of an `object_key` is not specified by CORBA. However, this fact does not affect interoperability nor make the IOR any less portable. The `object_key` is used only by the *server* to identify the object referred to. The client needs to have a copy of the `object_key` but does not need to interpret it in any way. As far as the client is concerned, the key is just an opaque code (in fact, a sequence of bytes) which it passes to a server in order to identify an object. The server, which originally assigned the `object_key`, then makes active use of the key to find the object.

This outline of an IOR is only intended to be schematic although it does include all essential information which is supplied in a typical IOR. The formal specification of an IOR is given in terms of IDL data types. For the complete specification of an IOR, refer to the CORBA interoperability specification.

Orbix/OrbixWeb Object Key Format

Orbix and OrbixWeb object keys in IORs have the same format as Orbix-protocol object references. They take the following the form:

```
:\host:serverName:marker:IR_host:IR_Server:interfaceMarker
```

These fields are defined as follows:

`host`	The host name of the target object.
`serverName`	The name of the target object's server as registered in the Implementation Repository and also as specified to `CORBA::BOA::impl_is_ready()`, `CORBA::BOA::object_is_ready()` or set by `setServerName()`. For a local object in a server, this will be that server's name (if known)—otherwise it will be the process' identifier. The server name will be known if the server is launched by Orbix, if the server is launched manually and the server name is passed to `impl_is_ready()`, or if the server name has been set by `CORBA::ORB::setServerName()`.
`marker`	The object's marker name. This is either chosen by the application or is a string of digits chosen by Orbix.
`IR_host`	The name of a host running an Interface Repository that stores the target object's IDL definition. This field is typically blank.
`IR_server`	The string *IR* or *IFR*, depending on the version of Orbix or OrbixWeb in use.
`interfaceMarker`	The target object's interface. If called on a proxy, this cannot be the object's true (most derived) interface—it may be a base interface.

Representations of an IOR

A portable representation of an IOR is a basic requirement. Typically, an IOR is created by the server which supports the corresponding object. The IOR is then publicised in order to make it available to prospective client processes. Once a client obtains a copy of the IOR it will then be able to connect to the object.

To assist publication of an IOR, it must be possible to convert it to a string format which is not subject to any conversions when communicated from place to place. For this reason, CORBA specifies a standard string format for IORs. The following is an example of such a string:

```
IOR:000000000000000d49444c3a677269643a312e3000000
00000000001000000000000004c0001000000000015756c74
72612e6475626c696e2e696f6e612e696500000963000000002
83a5c756c7472612e6475626c696e2e696f6e612e69653a67
7269643a303a3a49523a67726964003a
```

It consists of the characters `IOR:` followed by a series of hexadecimal numbers. Every byte of the original IOR is translated into a two-digit hexadecimal number. This standard string format is simple and resistant to corruption, however, interpreting the content of the IOR is difficult.

A typical IOR is not really as opaque as this. To make IORs more comprehensible, Wonderwall can use its own format known as the Readable Hex Representation (RXR). The RXR format is a hybrid format which mixes plain ASCII characters with hexadecimal numbers. As an example, consider the RXR representation of the preceding object reference:

```
RXR:_______%0dIDL:grid:1.0_______%01_______L_%01_
____%15ultra.dublin.iona.ie__%09c___(:%5cultra.du
blin.iona.ie:grid:0::IR:grid_:
```

The RXR format is provided in order to provide readable logging messages and a convenient way to specify strings of octets. It incorporates concepts from the URL encoding for HTTP (RFC 1738). RXR format strings are written as follows:

```
RXR:<version><string>
```

The 4-character upper-case string `RXR:` must be present at the start. The `<version>` specifier is optional and can be omitted. If it is present, it takes the form `%vX` where the `X` character encodes a format identification character ranging from 0 to 9, a to z, and A to Z. If this version specifier is not present, version 0 is assumed. This document describes RXR format version 0.

Each octet of the octet string is stored, in order, in the `<string>` specifier. Octets must be encoded if they have no printable representation in the US-ASCII coded character set, if the use of the corresponding character is *unsafe*, or if the corresponding character is reserved for some other interpretation within this representation format.

The octets which must be encoded are as follows (the values are specified in hexadecimal, and ranges are inclusive): any octet from 00 to 20, octets 22, 23, 25, 27, and 3B, octets between 5B and 60, octets from 7B to FF. Here is an annotated list of ostensibly-printable octets deemed unsafe:

Octet value	**Special use**
`%`	Used to signify octet-encoding.
`_`	Used to signify null-encoding.
`# ;`	Can be used as a comment.
`' "(space)`	Can be used as a string delimiter.
`‘ [ ] | { } ~ \ ^`	Can be corrupted by gateways or shells.

The encoding methods are as follows:

- For non-NUL (hex 00) octets, a ‘%’ (percent) character is stored in the string, followed by the high-order nibble of the octet encoded in hexadecimal, followed by the low-order nibble encoded in the same way.
- The character ‘_’(underscore) is used to encode a NUL (hex 00) character. An example RXR encoded IOR from OrbixNames is as follows:

```
RXR:_______%20IDL:CosNaming/NamingContext:1.0
____%01_______W_%01_____%10192.122.221.136_a%eb__
___7:%5cultra.dublin.iona.ie:NS:::IR:CosNaming%5f
NamingContext_
```

Internet Inter-ORB Protocol (IIOP)

The IIOP protocol is a special case of the General Inter-ORB Protocol (GIOP). The GIOP specification provides a general framework for protocols to be built on top of specific transport layers. The IIOP protocol is the specialisation of GIOP which is built on top of TCP/IP.

Many aspects of IIOP discussed in this section apply equally to any GIOP protocol, but no attempt is made to distinguish the different elements of the specification here.

In general, the IIOP specification has three main elements:

- Transport management requirements.

  The transport management requirements give a high level view of the semantics of setting up and ending connections. The roles of client and server and the respective functions of each are outlined at this level. The protocol described is connection oriented with well-defined roles for client and server.

- Definition of CDR coding.

  The second element of the IIOP specification is the Common Data Representation (CDR). This transfer syntax specifies a coding for all IDL types: including basic types, structured types, object references (in the form of IORs), and pseudo-object types such as `TypeCodes`. The CDR coding translates IDL types into a series of bytes to make up an octet stream (the CORBA name for a raw memory buffer).

  A feature of CDR is its ability to deal with the different kinds of byte ordering required by different hardware types: both big-endian and little-endian byte ordering is supported. The convention adopted is that the *sender* of a message sends data using its native byte ordering (and sets a flag in the message header to indicate the ordering used). The receiver of a message is obliged to detect the byte ordering used and carry out any conversion, if it is required. The advantage of this convention is that when both sender and receiver use the same byte ordering, no conversion is required resulting in considerable gain in efficiency.

- IIOP message formats.

  The third element of the IIOP specification is the message format. This is discussed in the following section, "IIOP Message Formats".

IIOP Message Formats

The IIOP protocol defines seven types of message format. The messages allow clients to pass invocations to servers and receive replies which can be either normal or indicate some error status. Some additional messages are available to help manage the connection.

The two most important message formats are the Request and Reply message formats. An operation which has been declared in the IDL interface for an object will be invoked by a client using a Request message. The client will usually wait for a Reply message from the server (unless the operation has been declared to be oneway) which will normally contain a return value, or possibly an error condition.

The other five messages are all concerned with managing some aspect of the connection and their roles are discussed in the following sections.

Typically, IIOP messages fit into one of three formats as follows:

1. A GIOP message header only.
2. A GIOP message header followed by a message header specific to the message type.
3. A GIOP message header followed by a specific message header and message body.

In all cases, a message will begin with a GIOP header. The form of the GIOP message header is illustrated in Figure 5.2 as follows:

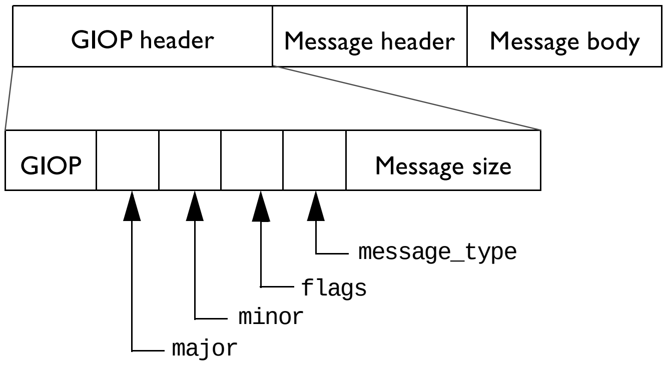


Figure 5.2: *The Format of a GIOP Message and Message Header*

The fields in the header can be described as follows:

- The four characters "GIOP" serve to identify the protocol.
- The GIOP version number (major and minor) is used to create the message.
- A flag byte is currently only used to indicate the byte ordering.
- An integer is used to indicate the message type.
- The message size (excluding the GIOP header itself).

This summarises all information which is sent to the GIOP header. For a formal specification of the exact header format, consult the CORBA specification.

To use Wonderwall effectively, the following sections sufficiently describe the purpose and usage of the different IIOP message formats. For complete details of the message formats, however, consult the CORBA specification.

Request Message

A Request message allows a client application to invoke an operation on a remote server. The message contains all the information which is needed for the invocation including the identity of the object, the operation name, and any parameters associated with the operation. Because a Request message is designed specifically to invoke operations which have been declared in an IDL interface, the message format is designed to support all of the syntax which can appear in an IDL operation definition.

The message consists of a Request header followed by a Request body. An outline of the Request header is shown in Figure 5.3 on page 49. It consists of the following fields:

- The `service_contexts` field allows service specific context information to be passed along with a Request. Intended for use in conjunction with the CORBA services to carry extra information along with the Request[2], the service contexts are not needed in the core specification of CORBA.

2. This field is used by the Transaction Service, for example.

- The request_id field is used to uniquely identify a Request emanating from a client so that the client can later match a received Reply with its corresponding Request (the corresponding Reply will be tagged with the same request_id).
- The response_expected flag is used to indicate whether the Request is oneway or not. A normal Request has response_expected set equal to TRUE.
- The next field is an array of three bytes reserved for future use.
- The object_key field is used at the server end to identify the object which is being invoked.
- The operation field is simply a string giving the name of the operation being invoked.
- The requesting_principal field identifies the user making the request. That is, it is simply the user name of the person running the client.

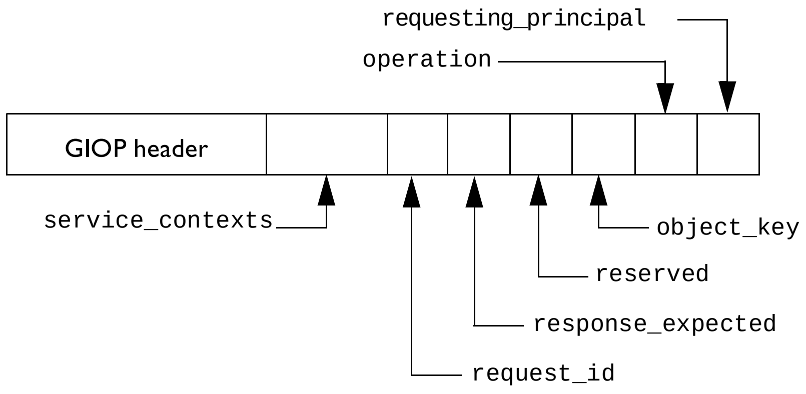


Figure 5.3: *The Format of a Request Message Header*

Figure 5.3 illustrates all of the information available in the Request header. The Request header is of particular importance to the operation of Wonderwall because Wonderwall carries out its filtering based upon the contents of the Request header. Appendix A on page 85 verifies all rules for filtering requests based on the contents of this header.

The Request also has an associated Request body. The body of the Request consists essentially of a list of the operation parameters followed by any `context` strings for the operation.[3] It is possible for the body of the Request to be empty—for example, if the Request was made for an operation which took no parameters and omitted a `context` clause.

Because filtering done by Wonderwall is based entirely on the Request header, there is no need for it to parse, or alter in any way, the Request body. This fact simplifies the filtering process significantly—ensuring the simple and efficient filtering and forwarding of request messages.

Reply Message

A Reply message is normally sent by a server in response to a client Request message. The Reply message consists of a GIOP header followed by a Reply header and a Reply body. The usual intent of a Reply message is to pass back a return value for an operation and to indicate the completion status for the operation.

The Reply header does not pass as much information as a Request header and typically consists of the following three fields:

1. The `service_context` field which is similar to the service context described in connection with a Request message.
2. The `request_id` field which is used to match this Reply to the client Request which gave rise to it. That is, all Replies are paired off with their corresponding Request and the `request_id` is a unique (per client) identifier used to match Request and Reply.
3. The `reply_status` field is used to indicate whether this is a normal Reply or if some error condition occurred in the server.

3. These `context` strings have nothing to do with service contexts. They are effectively middleware environment parameters and they will only be passed if a `context` clause appears at the end of an operation definition in IDL.

The `reply_status` is used to toggle between a number of different Reply types so that a Reply message is almost like four messages rolled into one. The possible values for `reply_status` are as follows:

`NO_EXCEPTION`	This is the normal Reply type. The body of this Reply type contains any return, `out`, or `inout` parameters which have been declared in the IDL for the operation.
`USER_EXCEPTION`	This status indicates that a user exception has been raised in the server. The body of this Reply type contains the details of the user exception.
`SYSTEM_EXCEPTION`	This status indicates that a system exception occurred. The body of the Reply indicates the kind of system exception raised.
`LOCATION_FORWARD`	This is a special kind of Reply which a server can use to let a client know that it does not hold the object to which the Request refers. The body of a `LOCATION_FORWARD` reply contains a new IOR for the object. The client can use the new IOR to resend the Request to the new location (this is done transparently as part of the IIOP protocol).

The Reply is routinely used by the Orbix daemon to dynamically allocate a port to a server process which has been automatically forked by the daemon.

CancelRequest Message

A CancelRequest message is sent by the client to the server to indicate that the client is no longer interested in receiving a Reply to a particular message. However, it is not an error if the server sends the Reply anyway.

LocateRequest Message

A LocateRequest message can be sent from client to server to probe for the location of a remote object. It is advantageous to send this message before sending a large Request on a connection which has just been opened.

LocateReply Message

A LocateReply message is sent from server to client in response to a LocateRequest message. There are three kinds of LocateReply message which the server can send as follows:

1. The `UNKNOWN_OBJECT` response indicates that the server does not hold the object and neither does it know where to find it.
2. The `OBJECT_HERE` response indicates that the server holds the object and communication can proceed as normal.
3. The `OBJECT_FORWARD` response indicates that the server does not hold the object but it does know of a forwarding location for the object. In this case, and in this case only, the LocateReply message has a body. This LocateReply body contains the new IOR.

CloseConnection Message

A CloseConnection Message is sent by the server to the client to tell the client that it intends to close the connection.

MessageError Message

A MessageError message can be sent by either the client or the server. It is used within the IIOP protocol to indicate that the last message received was either corrupted or incorrectly formatted in some way. It consists only of a GIOP header with the message type set to MessageError.

6

Interoperability and Details

The IIOP protocol was introduced to facilitate interoperability between ORBs supplied by different vendors. For the most part, the use of this protocol is transparent to the user—the main difference noticed is that your ORB is able to talk to many different ORBs, as a result of sharing a common protocol.

Object References

One aspect of IIOP which the user should be aware of is that the information required to make an initial connection to an ORB must be passed around by some means other than using the IIOP protocol. A connection is established between client and remote server with the help of an Interoperable Object Reference (IOR), which details the location of the object and the information needed to connect to the server.

There are two main formats of an IOR as follows:

1. An encoded IOR format is used to transmit IORs inside an IIOP message.
2. A stringified IOR format is used to communicate an IOR by any convenient means—refer to "Representations of an IOR" on page 44 for further information.

   This stringified IOR is typically given to a client to allow it to bootstrap the initial connection to a server. Subsequent IORs can be obtained from the server via the IIOP protocol itself.

Normally the server, which holds the object, creates an IOR for the object and makes this public in some way. The four main ways in which an IOR can be made known to a client are as follows:

1. The server can create a stringified IOR and write this IOR to a file in a well-known location which is accessible to the client.
2. The server can register the IOR with the CORBA Naming Service. The Naming Service, as detailed in the CORBA specification, is basically a database that associates names with object references. (A client requires just a single bootstrap reference to the Naming Server in order to access all of the IORs stored there.)
3. An IOR can be sent inside an IIOP message. This applies to any IDL operation which features an interface name as a parameter or return type.
4. The Orbix-specific bind mechanism (used for an Orbix or OrbixWeb client talking to an Orbix or OrbixWeb server) can be used. The client initially makes contact with the Orbix daemon and the daemon helps the client to determine the IOR of the required object. In such a case, the client does not need an IOR to get started—bind provides an alternative bootstrap mechanism.

Of these four methods, the first two provide the most general interoperable way of bootstrapping initial connections.

Wonderwall is based on the use of two IORs for each object: the *real* IOR and the *proxified* IOR. The real IOR is used by servers operating behind the firewall. The proxified version of the IOR is publicised and made generally available outside the firewall. This is detailed in the following sections.

Proxification

As a general convention on the Internet, frequently used servers are assigned to a dedicated port. For example, most HTTP servers operate on port 80, most Internet mail servers operate on port 25, and so on. Whenever contact with a remote host is made, connection to a particular service by opening a socket on its well known port can be established. Wonderwall fits this convention by providing a single dedicated port for IIOP messages.

A port number is embedded directly into every IOR and can have any value. If a large number of CORBA servers are active on a given host, then a large number of ports may be in use for IIOP communications. From a CORBA perspective, this makes sense, as each of these processes is a dedicated server carrying out a specific sort of task.

From a firewall perspective, however, the use of multiple IIOP ports poses difficulties. It is undesirable, from a security point of view, to allow the use of multiple ports on the bastion host. Firewall practice is based on collating all messages of a single protocol type, and passing them through a single port.

Wonderwall uses a single IIOP port on the bastion host. Any IORs which are used remotely should point at the Wonderwall host and port. The IORs generated by servers on the internal network feature a range of hosts and ports, depending on where they were generated. These IORs are suitable for use on the internal network since they allow direct IIOP connections to be established behind the firewall. However, giving them away to users on the Internet is undesirable—because they facilitate direct connections to internal hosts, and a properly constructed firewall (in any case) would make them unusable across the Internet.

It is thus necessary to modify the real IORs before making them available on the Internet—a process referred to as the *proxification* of an IOR. The principle of proxification is illustrated in Figure 6.1 and Figure 6.2 on page 56.

The Proxification Process

Looking at Figure 6.1 on page 56, when a client is communicating with `Obj`, it has the illusion that the object lives on the Wonderwall server. The IOR which is used to contact this object must have the host and port of server **W** (the Wonderwall proxy server) embedded, along with the `object_key` for `Obj`.

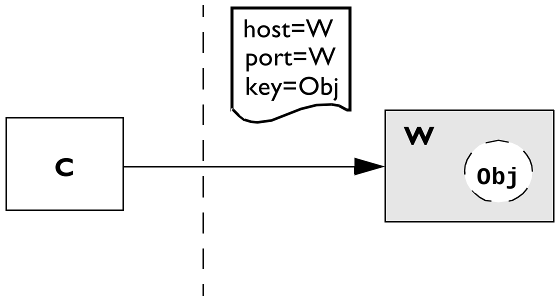


Figure 6.1: *Apparent Location of Object, in Wonderwall Proxy Server*

In reality, the object lives behind the firewall and is located on server **S** in the internal network (see Figure 6.2). The real IOR for this object has the host and port of server **S** embedded in it, along with the `object_key` for `Obj`. Wonderwall acts as a proxy for this server, forwarding any messages it receives from the client (subject to filtering by the Access Control List).

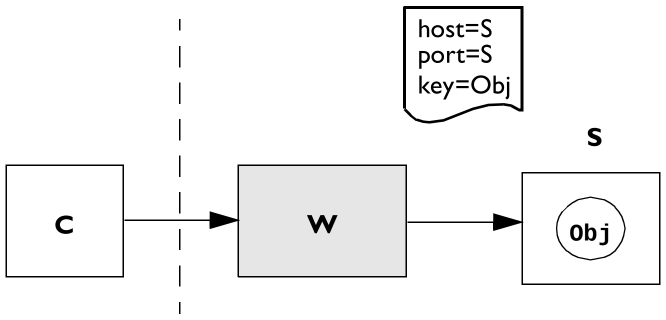


Figure 6.2: *Actual Location of Object, in Server S*

The only difference between the real and the public IOR is the value of the embedded host and port. The host and port embedded in the real IOR must be changed. The resulting IOR is a *proxified* IOR.

Proxification can be carried out using the `iortool` utility which comes with Wonderwall. First the real IOR for `Obj` on server **S** is written to a file, say `real.ref`, in stringified form[1]. Then the real IOR is proxified with the following command:

```
% iortool -ior -proxify \
   -host wonderwall_host -port wonderwall_port \
   real.ref > proxy.ref
```

This takes the IOR stored in file `real.ref` (which can either be in `IOR` or `RXR` format) and replaces the current host and port embedded in the IOR by `wonderwall_host` and `wonderwall_port` instead. The result of this proxification process is written to the file `proxy.ref` in `IOR` format.

Note: The `IOR` format is the portable string representation, as defined by CORBA.

1. Refer to “Representations of an IOR” on page 44 and consult your ORB programming guide for instructions on how to generate a stringified version of the real IOR.

Non-Orbix Client

If using a non-Orbix client to connect to a server via Wonderwall, you will not have the option of using the bind mechanism. The interoperable approach is based on the use of proxified object references, as described in "Proxification" on page 55. A proxified object reference is obtained (see "The Proxification Process" on page 55) and this proxified reference is publicised using one of the methods discussed in "Object References" on page 53. If the client has access to this IOR in string format, a connection can be established to the server using code similar to the following code sample. This sample assumes that the remote object is of type `grid`:

```
// C++
main () {
   ...
   char *proxifiedIOR;
   CORBA::Object_var objVar;
   ...
   // Read the proxified IOR into a string buffer
   // pointed to by 'proxifiedIOR'.
   ...
   // Convert the string to an object reference.
   objVar = CORBA::Orbix.string_to_object(proxifiedIOR);
   ...
   // Assume that this is the proxified IOR for a
   // 'grid' object. Perform a '_narrow()'
   grid_var myGridVar = grid::_narrow(objVar);
   ...
}
```

A reference `myGridVar` is obtained to the desired `grid` object.

Note: Error handling has been omitted from this example for clarity, but in a real situation it is imperative to enclose the calls in a `try/catch` clause.

It is generally more difficult for a client to get a reference to its first object, whether it is via a stringified object reference, as described in the preceding code sample, or via the Name Server. If this first object is either a Finder or Factory object, then subsequent object references can be obtained more easily.

Non-Orbix Server

For non-Orbix servers, the main restriction is that they are not able to respond to the Orbix bind mechanism. This affects the Wonderwall proxy, because it is the Wonderwall proxy which attempts to make direct connections with servers behind the firewall. Wonderwall will not be able to specify objects using the bind syntax in its configuration file. For example, if you want to make an object known to Wonderwall, the following line should not be included in your configuration file:

```
object grid_1 bind("grid1:GridSrv","gridHost") interface grid
```

Instead, the following steps should be carried out:

- Obtain a copy of the real IOR for the object in CORBA string format—consult your ORB programming guide for instructions on how to do this. This IOR is needed if you want to generate a Proxified IOR for a non-Orbix client.
- Copy this string to a file in a convenient location, for example, as follows:

`/etc/iors/grid1.ref`	UNIX
`C:\IONA\Wonderwall\IORS\Grid1.ref`	Windows NT

- Make this object known to Wonderwall by including the appropriate line in the configuration file as follows:

`object grid_1 /etc/iors/grid.ref`	UNIX
`C:\IONA\Wonderwall\IORS\Grid1.ref`	Windows NT

There are a few alternatives you can use for the *object-specifier* field—see "List of IORs" on page 98. However, the approach outlined here is probably the most convenient for the interoperable case.

Connection Establishment

This section explains some of the steps involved in establishing a connection through Wonderwall. By involving the daemon in the process of connection establishment, it is possible to have servers launched automatically. This means that the server is contacted in a sequence of steps, beginning with an initial connection to the daemon. It is for this reason that the configuration keyword `orbixd-iiop-port` must be set equal to the value of the daemon port—knowledge of this port is needed to facilitate communication with the daemon process.

A Normal IIOP Connection

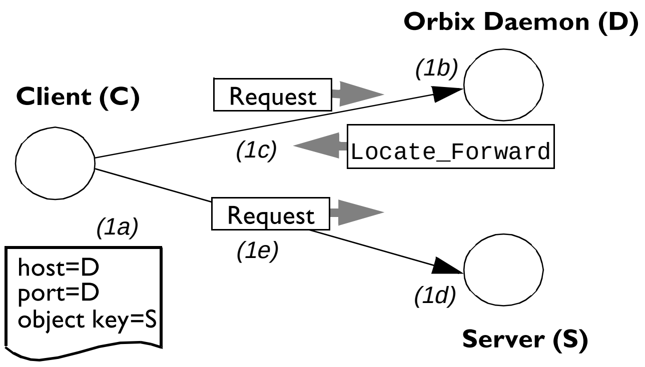


Figure 6.3: *Establishing a Normal Connection*

Figure 6.3 describes a normal IIOP connection as follows:

- Client **C** requests the IOR for server object **S**[2]—a Java applet could get this from an applet tag. This contains the connection details of the activation agent—for example, the Orbix daemon on the server's host.

2. An OrbixWeb client can use the `bind` mechanism as an alternative.

- *(1a, b)*: **C** opens a TCP/IP connection to host **D**, port **D**.
- *(1c)*: **C** sends the request message to **D** (the Orbix Daemon). **D** responds with a `LOCATION_FORWARD` Reply message containing the real location of **S**.
- *(1d)*: **C** opens a TCP/IP connection to host **S**, port **S**.
- *(1e)*: **C** sends the request message to **S**.

Note: This example assumes the use of a non-persistent server. Persistent servers do not require the presence of an activation agent—for Orbix, the Orbix daemon is an activation agent. In such a case, the IOR in the first step would contain the connection details of **S** rather than **D**, and the second and third steps would not be necessary.

An IIOP Connection Through Wonderwall

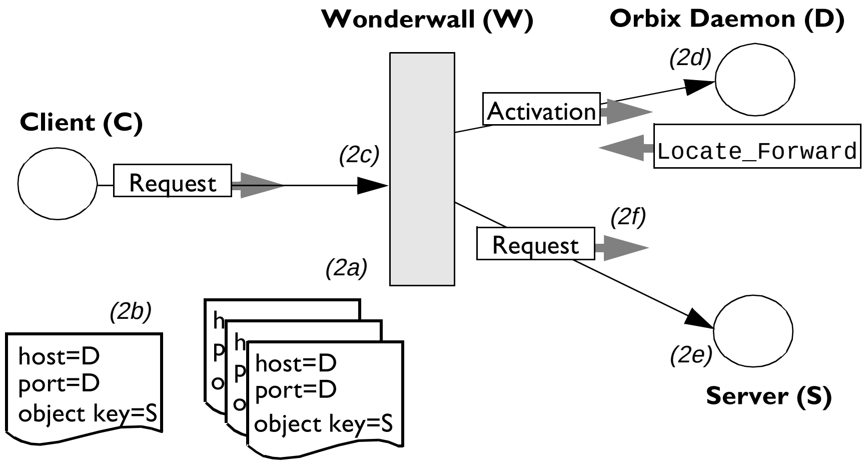


Figure 6.4: *Establishing a Connection Through Wonderwall*

Figure 6.4 on page 61 describes the process of establishing a typical IIOP connection through Wonderwall as follows:

- Wonderwall **W** requests the IOR for server object **S**. This is copied into the configuration file by the system administrator.
- *(2a, b)*: Client **C** requests the proxified IOR for server object **S** using the same method as described in "A Normal IIOP Connection" on page 60.
- *(2c)*: **C** opens a TCP/IP connection to host **W**, port **W**, and sends the request message to **W**.
- *(2d)*: **W** reads the request, finds the IOR in its configuration that matches the object key used in the request, and opens a connection to host **D**, port **D**. An *activation request* is sent to the daemon, causing the server **S** to startup. **D** responds to the activation request with the connection details for **S**.
- *(2e)*: **W** opens a connection to **S** using the details from **D**.
- *(2f)*: **W** forwards the request message to **S**, forwards any replies back to **C**, and so on until the connection closes.

Note: Again, this assumes the use of non-persistent servers. Persistent servers use the connection details of **S** rather than **D**—in such a case, the fourth step is skipped.

A More Complicated Connection: Using Object Factories

Figure 6.5 on page 63 describes a connection using object factories. Factory objects are server objects which create objects to handle requests. Figure 6.5 also applies to servers which return IORs so that clients can bind to objects.

Wildcard flags used in Figure 6.5 indicate that an IOR in the Wonderwall configuration file can be used to match in an approximate manner. Different wildcard flags are required to support other situations. To reduce clutter in Figure 6.5, the server-activation stage has been omitted.

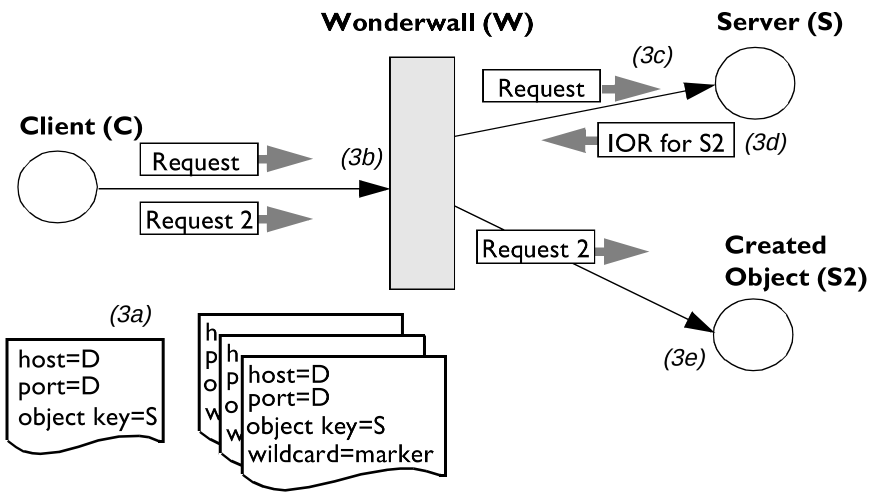


Figure 6.5: *Establishing a Connection to a New Object Through Wonderwall*

Figure 6.5 describes the establishment of a connection to a new object (using object factories) through Wonderwall as follows:

- *(3a)*: Client **C** requests the proxified IOR for server object as described in the previous examples.
- *(3b)*: **C** opens a TCP/IP connection to host **W**, port **W**, and sends a Request message to **W**.
- *(3c)*: **W** reads the Request and examines the object key. Since no IOR in its configuration exactly matches the object key, it runs through its list of *wildcard* IORs. It finds the IOR that approximately matches, opens a connection to host **S**, port **S**, and forwards the Request.
- *(3d)*: **S** creates the object **S2** and sends an IOR for it back to **W** which forwards it on to **C**.
- *(3e)*: **C** makes a second request and this time it invokes on object **S2**. **W** reads the Request and examines the object key. Since **S2** uses the same host, port, interface, and server name, the wildcard IOR used in *(3c)*

matches this Request as well. A connection is opened to host **S**, port **S** (the addressing information in this IOR), and the Request is passed to object **S2**.

This initial version of Wonderwall requires that objects created by the factory are represented by proxified IORs. OrbixWeb has an API call to ensure this.

Note: Objects **S** and **S2** might not share the same connection details. In such a case, a separate wildcard IOR would be necessary listing the known details of **S2**.

Factory Objects and IORs

Factory objects typically employ a method which is used to create a CORBA object on the server to which an interoperable object reference is returned. For example, a general version of a `GridFactory` could be defined as follows:

```
// IDL
interface GridFactory {
   // Make an object of type 'grid'
   // and return an IOR.
   grid makeGrid();
};
```

The `makeGrid()` operation returns an object reference to an object of type `grid`. This type of interface, however, poses problems for the firewall.

Typically, the default behaviour of an operation such as `makeGrid()` is to generate an object reference which points directly at the object on the server itself. But this object reference is not useful on the Internet because it points at a server on the internal network, behind the firewall. The operation of the firewall is designed to prevent *direct* access to such internal servers.

A solution to this is to change the default behaviour of the server, so that any object references it returns will refer to the Wonderwall host and port instead. That is, the Factory object should generate *proxified* object references instead of real object references. Refer to the programming manual for your ORB for further information on this.

Another solution, one which is more interoperable and can be implemented without change to either client or server code, is to use the **proxify** parameter in the Access Control List. This will cause Wonderwall to proxify object references returned by the named operation.

To allow external access to objects generated by the Factory object, in addition to implementing the `GridFactory` interface, you will need to add to the Wonderwall configuration file. A typical approach is the following:

- Generate a real IOR for the `GridFactory` object and store it in a convenient location. For example:

`/etc/iors/GridFactory_real.ref`	UNIX
`C:\IONA\Wonderwall\IORS\GridFactory_real.ref`	Windows NT

- Declare a tag in the Wonderwall configuration file which refers to *all* of the objects on the same server as `GridFactory`. For example:

`server GridFactory /etc/iors/GridFactory_real.ref`	UNIX
`server GridFactory C:\IONA\Wonderwall\IORS\GridFactory_real.ref`	Windows NT

- Use this tag in a rule to allow access to all objects in the `GridFactory` server as follows:

```
allow object GridFactory
```

This configuration allows any proxified object references, generated by `GridFactory`, to be used by the external network irrespective of marker or interface type. Using the `server` keyword to generate a tag allows you to regulate permissions, a server at a time. Currently, this form of wildcarding is supported only for Orbix and OrbixWeb.

Factory Objects and the "Proxify" Parameter

Wonderwall supports automatic proxification of returned IORs as they pass through the proxy. To proxify the IOR returned by the `makeGrid` operation, use the `proxify` parameter in a rule as follows:

```
allow object GridFactory op makeGrid proxify
```

Implications for Developers

From a developer's perspective, the use of Wonderwall has minimal impact. Once the server is ready to be made available to the Internet, the IOR and the list of required operations are passed on to the firewall administrator who assesses the security of the server and updates the Wonderwall configuration file. Alternatively, an Orbix or OrbixWeb server allows you to use the `bind` form of an *object-specifier* in the configuration file (as explained in "List of IORs" on page 98).

Generally, on the client side it is only necessary to ensure that the client receives a copy of the proxified IOR so that it can establish an initial connection. In the special case of an OrbixWeb client, the procedure is simplified so that an OrbixWeb client will transparently connect to the server via Wonderwall—if a direct connection is blocked by a firewall.

Callbacks

A firewall unfriendly feature of the IIOP protocol is its use of dynamic port assignment. Firewalls are based around the idea of ports, protocols and services mapping to one another. For example, the SMTP protocol for Internet mail runs on port 25; thus a connection from a client to a server on port 25 is used for sending mail. Since IIOP dynamically creates and assigns ports, this no longer applies so the usual paradigm of opening up a single port to support a particular protocol cannot be assumed.

This is of particular relevance for callbacks. If the client's site is not protected by a firewall, the callback mechanism in the current IIOP specification is unlikely to work successfully. This is because it relies on the server opening a TCP connection to the client using a dynamically assigned port. If the client's site is protected by a firewall, this connection will be blocked. The typical scenario of opening a well-known port does not apply here.

In such a case, a possibility to consider is to extend the existing IIOP specification to allow the callback to use the same connection as that used for the initial incoming invocation from the client. Until this is standardised, however, the above model will be restricted to the usual Internet client-driven paradigm. In CORBA terminology this means that invocations can only be issued from the client site to the backend service behind the firewall. Objects resident in a client application/applet behind a firewall cannot act as CORBA servers receiving requests from objects resident in the backend service.

Note: Although callbacks are not supported, the Requests can of course be two way.

Orbix and OrbixWeb deals with this problem by extending IIOP to provide bi-directional IIOP. With OrbixWeb 3 or Orbix 2.3c and later, callbacks can be delivered from server to client regardless of any firewalls between the two ORBs.

7 Transformers

Several internal layers of Orbix separate a simple remote invocation—as seen by application level programming—from the final construction and transmission of a message via TCP/IP. In doing so, Orbix offers the user a chance to customise its behaviour by providing hooks at a number of levels. For example:

- When an Orbix operation is called on the client side, it can be intercepted straight away using a *Smart Proxy* to customise its behaviour.
- Orbix provides another hook in the form of `Filter` objects which can inspect (and modify) a Request object.
- Finally, after the full contents of a Request have been marshalled into a raw buffer, Orbix provides access to the buffer via a mechanism known as a *Transformer*.

Transformers are useful for a number of reasons. One of the main uses of a transformer is to allow encryption of the message prior to transmission via the TCP/IP protocol. This provides the user with an added level of security which is desirable in many situations. If your site has a policy of encrypting all messages prior to transmission, you will find that the support provided by Wonderwall for transformers allows you to insert a firewall with no disruption to the encryption process.

Transformer Architecture

Figure 7.1 provides an outline of a typical Transformer setup in the absence of a Wonderwall firewall.

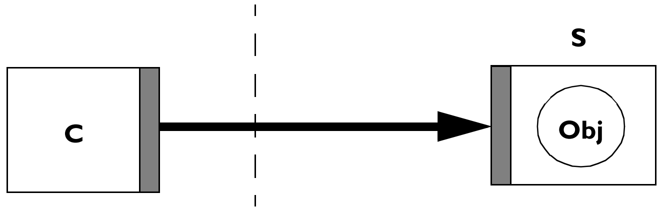


Figure 7.1: *Encrypted Link Using Integral Transformer*

- The shaded blocks shown at the edge of the client process **C** and server process **S** represent the transformers at either end of the connection.
- The heavily shaded line connecting client and server is used to represent the transmission of an encrypted IIOP message.

Note: The transformers in this figure are not implemented as CORBA objects. These transformers are referred to here as *integral transformers*—since they are built into the client or server process.

Consider the problem of interposing a firewall proxy between this client and server. The firewall cannot deal directly with encrypted messages, nor can it properly monitor and filter messages while they are in encrypted form. In Figure 7.1, decryption is carried out by the server's integral transformer. Logically, however, the point at which decryption occurs is just before the packet passes through Wonderwall.

Wonderwall offers two alternative solutions which enable you to insert the firewall even when encryption is being used—Figure 7.2 on page 71 and Figure 7.3 on page 72 both provide an outline of a typical transformer setup in the presence of a Wonderwall firewall. Both of these solutions are based on the definition of *external transformers* which Wonderwall uses to encrypt and decrypt messages. In contrast to integral transformers, these transformers are implemented as CORBA objects.

The first solution is shown schematically in Figure 7.2 where a single *client transformer* **T_C** is implemented to handle encrypted messages arriving from remote clients. This model may be appropriate when the main perceived risk to security is the external network.

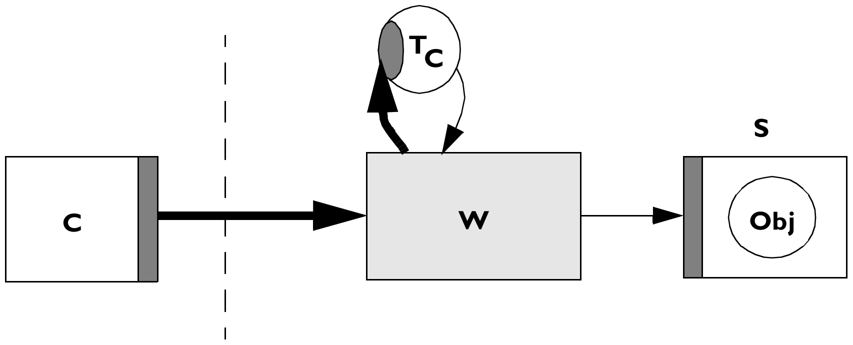


Figure 7.2: *Encrypted Link Using Wonderwall*

- When an encrypted Request message arrives from the client, Wonderwall first sends the message out to the external, client transformer **T_C**.
- The transformer returns a decrypted message (indicated by a thin line) to Wonderwall. The Wonderwall proxy is then able to monitor and filter this Request message as normal and if allowed by the Access Control List, the Request will be forwarded to the Server **S**.
- When the server responds to the client with a Reply message, the reverse procedure is followed.

  The unencrypted message is sent from server to Wonderwall, which might log the message, then passed to the client transformer **T_C** for *encryption*, then relayed by Wonderwall back to the client in encrypted form.

  In this case, the messages which circulate on the internal network are left in unencrypted form. Wonderwall provides a single point of decryption and encryption for all messages entering and leaving the internal network.

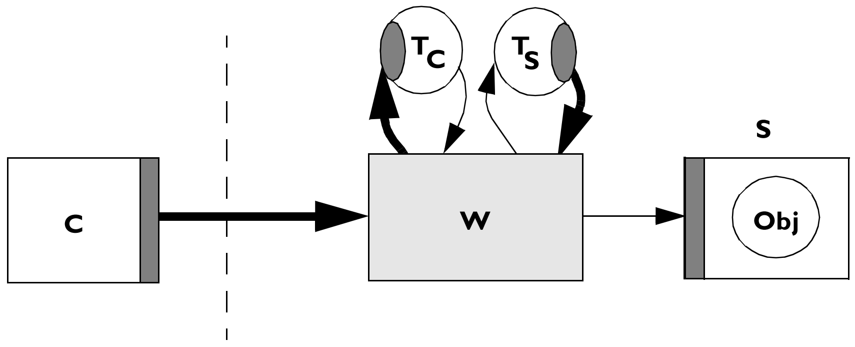


Figure 7.3: *Wonderwall Inserted Into Encrypted Link*

In Figure 7.3, Wonderwall offers an alternative solution enabling you to insert the firewall even when encryption is being used. In this model, encypted messages are circulated on both the internal and external networks. The advantage of deploying Wonderwall in this way is that the firewall can be inserted where an encrypted link already exists. No disruption is caused and the server needs no modification. This requires the use of two external transformers: the first transformer is the client transformer $\mathbf{T_C}$ which exchanges encrypted messages with the client, and the second transformer is the *server transformer* $\mathbf{T_S}$ which exchanges encrypted messages with the server. Sandwiched between these two transformers is the monitoring and filtering portion of Wonderwall performing all its operations on unencrypted messages.

Using Transformers

The external transformers, which Wonderwall uses, are defined as CORBA objects. They are not built into the Wonderwall server process. You are free to implement these transformer objects (both client and server transformer) in whatever way you like. Wonderwall defines an IDL interface for the transformer objects which you must use when writing your implementation. Wonderwall can be configured so that it automatically calls your transformer objects as needed. This is explained in the following sections.

The Transformer IDL

The external transformers used by Wonderwall, both client **T_C** and server **T_S**, are instances of the CORBA interface `IT_WonderwallReqTransformer`. The IDL interface is illustrated as follows:

```
   // IDL
   //
   // Wonderwall client/server Transformer interface:

   typedef sequence<octet> iiopMessage;

   interface IT_WonderwallReqTransformer {

      exception TransformFailedException {
         string reason;
         iiopMessage message;
      };

      void transform (inout iiopMessage data,
                      in string host,
                      in boolean sending)
         raises (TransformFailedException);

      void transform_exception (inout iiopMessage data,
                                in string host,
                                in boolean sending)
         raises (TransformFailedException);
};
```

This interface features a single operation `transform()` which is called whenever Wonderwall needs a message to be transformed. The first argument `iiopMessage` is the message to be transformed. An `iiopMessage` is declared to be of type `sequence<octet>` which is the data type that CORBA typically uses for buffers of bytes. It is perfectly permissible to pass back a transformed buffer which is a different size to the one received from Wonderwall. The next argument is the `host` which sent the Request (or is about to receive the Reply). The argument `sending` is used to indicate whether encryption or decryption is required. When a message is sent out of Wonderwall, this flag is true, and encryption is required. When a message is sent inwards, this flag is false, and decryption is required.

There is a single exception `TransformFailedException` which can be raised by the transformer implementation to abort the message. The user can decide under what circumstances the exception is raised. In a client transformer, Wonderwall will react to this exception by sending the message buffer included in the exception body back to the client. This way the integral transformer in the client application or applet can handle this message as an indication that the Wonderwall client transformer has failed.

Note: A `TransformFailedException` should not be thrown during the `bind()` or `narrow()` negotiation process, as the client ORB will not know how to handle the exception until communication with the server (via Wonderwall) is established.

If a server transformer throws a failure exception, a `MessageError` error message will be sent to the server.

The `transform_exception` operation is used to transform exceptions sent by Wonderwall to the client, if it becomes necessary for one to be raised during the course of operation. Exceptions from server to client are handled using the normal `transform` operation.

Implementing Transformers

The implementation of both client and server transformers is flexible. Apart from keeping to the rule that you should encrypt and decrypt messages when Wonderwall expects, you have complete freedom in implementing the transformers. The transformers can be implemented in any convenient language for which a CORBA mapping exists. They can be implemented in a standalone server, or built into some existing server on the internal network.

The following code sample outlines the skeleton of a client transformer which is implemented in the Java programming language with the help of OrbixWeb 3:

```
// IDL
package wwallXformerBoth;

import IE.Iona.OrbixWeb.CORBA.* ;
import java.util.Random;
Import IE.Iona.OrbixWeb.Features.IT.ReqTransformer;

public class TransformerImpl extends
   _IT_reqTransformerImplBase {

   public boolean transform
      (iiopMessageHolder data, String host, boolean is_send)
   {
      if (is_send) {
         //
         // Implement an algorithm to encrypt ‘data’
         //
         ...
      }
      else {
         //
         // Implement an algorithm to decrypt ‘data’
         //
         ...
      }
      return true ;
   }
```

```
      public boolean transform_exception
         (iiopMessageHolder data, String host, boolean is_send)
      {
         if (is_send) {
            //
            // Implement an algorithm to encrypt 'data'
            //
            ...
         }
         else {
            //
            // Implement an algorithm to decrypt 'data'
            //
            ...
         }
         return true ;
      }
   }
```

Note: The encryption algorithm is allowed to change the size of the sequence buffer and return a transformed sequence of a different length. If you wish to do this, you should reallocate the size of the sequence buffer before completing the transformation. A full example of an OrbixWeb client transformer is available in the `demo` directory of your Wonderwall distribution. The demo is based on the architecture of Figure 7.2 on page 71, where the client transformer is implemented in server **S**.

Configuration

Configuring Wonderwall to use either a client transformer, or a client and server transformer is quite straightforward. Once a client and server transformer have been implemented, insert two lines, for example, as follows into the configuration file (typically called `iiopproxy.cf`):

```
#################################################
# Configure client and server Transformers...
#
client-transformer \
   bind (“:myServer”, “internalHostA”) \
   interface IT_WonderwallReqTransformer
server-transformer \
   bind (“:myServer”, “internalHostA”) \
   interface IT_WonderwallReqTransformer
```

This setup is appropriate when both a client and server transformer object have been implemented in server `myServer`, on host `internalHostA`. If you do not wish to use the Orbix bind mechanism, you can substitute any form of *object-specifier* (as described in “List of IORs” on page 98).

Note: Since the encrypted Requests will be sent from the transformer servers to Wonderwall in unencrypted form, care should be taken that these connections cannot be intercepted. For example, the transformer server could be run on the Wonderwall machine itself. As long as the firewall protects the transformer, this is safe.

8

Using Wonderwall with OrbixWeb

OrbixWeb 3, an implementation of the Object Management Groups's (OMG) CORBA which maps CORBA functionality to the Java programming language, contains built-in support for Wonderwall. This makes Wonderwall much easier to use and administer—IORs no longer need to be proxified to be used with OrbixWeb and in low-security configurations, the need to list server objects in the Wonderwall configuration file is removed.

Because OrbixWeb contains built-in support for Wonderwall, Wonderwall can be used as either:

- A simple intranet request-routing server—which passes IIOP messages from your applet, via the web server, to the target server.
- A full-blown firewall proxy—which can filter, control, and log your IIOP traffic.

The remainder of this chapter discusses using Wonderwall with OrbixWeb.

Note: OrbixWeb 3 has built-in support for Wonderwall. However, if you use OrbixWeb 2, you should contact `support@iona.com` in order to receive an up-to-date patch for it which supports Wonderwall.

Using Wonderwall with OrbixWeb as an Intranet Request-Router

If you simply wish to provide a way for your OrbixWeb applets to contact servers which reside on hosts other than the one your web server is running on, and you do not have any reservations about security issues, then Wonderwall will provide this capability as an intranet request-router for IIOP. The file `intranet.cf` is provided for this configuration. The Wonderwall command line is as follows:

```
        iiopproxy -config intranet.cf
```

This mode of operation requires no configuration, apart from setting your daemon port and domain name. Using Wonderwall, any server can be connected to and any operation can be called.

Using Wonderwall with OrbixWeb as a Firewall Proxy

To run Wonderwall in a traditional secure mode, use the file `secure.cf` as an example of the configuration you should use. This mode of operation requires that the target objects and operations be listed in the configuration file. For further details, refer to Appendix B, "Configuration", which provides a guide to using Wonderwall's access control lists and object specifiers.

OrbixWeb Configuration Parameters Used to Support Wonderwall

OrbixWeb provides automatic support for Wonderwall built-in to its client-side. This allows OrbixWeb to transparently attempt to connect to any IIOP servers via Wonderwall if a connection attempt fails using the default direct socket connection mechanism. It also means that Wonderwall can be used to:

- Provide HTTP tunnelling for OrbixWeb-powered Java applets.
- Provide automatic intranet routing capability for OrbixWeb-powered Java applets, in order to avoid browser security restrictions.
- Use OrbixWeb applications and applets with Wonderwall, with no code changes.

Configuring OrbixWeb to Use Wonderwall

In order for OrbixWeb to use Wonderwall, it must be configured with the Wonderwall location. The following configuration parameters are used for this purpose:

```
OrbixWeb.IT_IIOP_PROXY_HOST
OrbixWeb.IT_IIOP_PROXY_PORT
```

The `IT_IIOP_PROXY_HOST` parameter should contain the name of the host on which Wonderwall is running. The `IIOP_PROXY_PORT` parameter should contain its IIOP port. These parameters can be set using any of the supported configuration mechanisms. For further information, refer to the release notes of the OrbixWeb version you are using. For example, a fragment of a HTML file which uses applet parameters is as follows:

```
<applet code=GridApplet.class height=300 width=400>
   <param name="OrbixWeb.IT_IIOP_PROXY_HOST"
value="wwall.iona.com">
   <param name="OrbixWeb.IT_IIOP_PROXY_PORT" value="1570">
</applet>
```

Configuring OrbixWeb to Use HTTP Tunnelling

HTTP tunnelling is a mechanism for traversing client-side firewalls. Each IIOP Request message is encoded in HTTP base-64 encoding, and a HTTP form query is sent to Wonderwall, containing the IIOP message as query data. The IIOP Reply is then sent as a HTTP response.

Using HTTP tunnelling allows your applets to be used behind a client's firewall, even when a direct connection (or even a DNS lookup of Wonderwall's hostname) is impossible.

In order to use HTTP tunnelling, you must use the new `ORB.init()` API call to initialise OrbixWeb—whether you are using OrbixWeb 2 or 3. The `ORB.init()` function must be called by all clients and servers in a CORBA system before they can make use of the underlying ORB. For further information, refer to the *OrbixWeb Programmer's Reference Guide* and the release notes of the OrbixWeb version you are using.

This is necessary as it allows OrbixWeb to retrieve the codebase from which the applet was loaded. The codebase is then used to find Wonderwall's interface for HTTP tunnelling. This is a pseudo-CGI-script called "`/cgi-bin/tunnel`". For further information on what the codebase is used for in Java, refer to the following URL: `http://www.javasoft.com/`.

Because an untrusted Java applet is only permitted to connect to the server named in the codebase parameter, Wonderwall should be used as the web server which provides the applet's classes. However, it is permissible to provide your main web site's HTML and images from another web server, such as Apache, IIS or Netscape, and simply refer to the Wonderwall web server in the applet tag, as follows:

```
[ in the file http://www.iona.com/demo.html ]

   <applet code=GridApplet.class
      codebase=http://wwall.iona.com/GridApplet/classes
      height=300 width=400>
   </applet>
```

With this setup, your HTML and images are loaded from the main web site (`www.iona.com`) and your applet code is loaded from `wwall.iona.com`—as a result, the applet can open connections to that host. For greater efficiency, it is advisable to make a ZIP, JAR, and/or CAB file containing the classes used by your applet and store them on the Wonderwall site as well. Regardless of whether you are using Wonderwall, this is generally a very good idea.

It is also feasible to provide a Wonderwall setup to support HTTP tunnelling on the same machine as the real HTTP server, by using a different port number from the default port 80. However, bear in mind that some sites can only support HTTP traffic on port 80, the standard port, so this may restrict your applets' potential audience.

You should ensure that the applet's classes are available in the directory you named in the codebase URL. In the previous example, this would be `GridApplet/classes`. This directory path is relative to the directory named in the Wonderwall configuration file's `http-files` parameter.

If you want an application to use HTTP tunnelling, or would prefer to override an applet's HTTP tunnelling setup, three more configuration parameters are provided:

```
OrbixWeb.IT_HTTP_TUNNEL_HOST
OrbixWeb.IT_HTTP_TUNNEL_PORT
OrbixWeb.IT_HTTP_TUNNEL_PROTO
```

The `IT_HTTP_TUNNEL_HOST` parameter should contain the name of the host on which Wonderwall is running. The `IT_HTTP_TUNNEL_PORT` parameter should contain its HTTP port. The `IT_HTTP_TUNNEL_PROTO` parameter should contain the protocol used. Currently, the only protocol value supported for HTTP tunnelling is *http*.

Wonderwall supports HTTP 1.1 and HTTP 1.0's Keep-Alive extension. This means that TCP connections between the client and Wonderwall (or between a HTTP proxy and Wonderwall) will be *kept alive*. That is, more than one HTTP request can be sent across them. This greatly increases the efficiency of HTTP.

Manually Configuring OrbixWeb to Use Tunnelling

In order to test HTTP tunnelling or IIOP via Wonderwall, two more configuration parameters are provided as follows:

```
OrbixWeb.IT_IIOP_PROXY_PREFERRED
OrbixWeb.IT_HTTP_TUNNEL_PREFERRED
```

If either of these are set to `true`, then that connection mechanism will be tried first, before the direct connection is attempted.

Appendix A
iiopproxy and iortool

Orbix Wonderwall is shipped with two binary files: `iiopproxy` and `iortool`. The `iiopproxy` file implements the firewall itself, while `iortool` is a useful utility for manipulating object references. Both of these commands each have a number of options as detailed in this appendix.

The iiopproxy Process

The command `iiopproxy` is usually run as a daemon process to monitor both the dedicated IIOP port and the HTTP port on the bastion host. It is the key component of the Wonderwall and combines the functionality of the IIOP gateway with a full HTTP server.

The `iiopproxy` is generally launched automatically using the `inetd(8)` on UNIX (for further information, refer to Appendix C) and remains permanently active, monitoring the designated ports, until it is explicitly killed.

Syntax of iiopproxy

The syntax of iiopproxy is as follows:

```
iiopproxy [options]
```

The *options* supported can consist of one or more of the following switches (these can be set using the GUI in the GUI version of Wonderwall):

`-config` *file*	Specifies the pathname of the Wonderwall configuration file (refer to Appendix B on page 93). This should be an absolute pathname when `iiopproxy` is run as a daemon process.
`-debug` *n*	The debug level can be set to three values: `0`, `1` or `2`. The value `0` means no diagnostics, `1` means minimal diagnostics, and `2` means high diagnostics. The default is `1`.
`-fork`	Causes the proxy to run in a forking mode, creating a subprocess for each new connection from a client. This was the default behaviour in Wonderwall 1.1—the default behaviour now is to use multiple threads in one process.
`-fg`	This debugging option means do not fork when a new connection arrives. Its usefulness is limited unless you are debugging—and it may cause trouble. This option is only available on UNIX.
`-help`	Give usage information on these command switches.
`-httpport` *port*	Specifies the port to listen on for HTTP requests. This can also be specified in the Wonderwall configuration file—but the value specified by `-httpport` takes precedence.
`-inetd`	Specifies that the `iiopproxy` is running from `inetd(8)` as a daemon process. This causes the `inetd` process to listen to the ports on behalf of Wonderwall. When this flag is not specified, Wonderwall listens on these ports itself. Wonderwall uses the current user ID as an identification when binding to servers. Hence the servers must be registered using `putit`, and `chmodit`'ed so that the user running Wonderwall has invoke and launch rights to the servers. This option is only available on UNIX.
`-log` *logfile*	Sends the log output into the named file, rather than to `stderr`, the standard error file descriptor.

`-port` *port*	Specifies the dedicated IIOP port for Wonderwall. This can also be specified in the Wonderwall configuration file—but the value specified by `-port` takes precedence.
`-user` *username*	Runs as a specified user. If Wonderwall needs to use a privileged port (that is, one under 1024), this switch should be used because it is safer to run as a normal, unprivileged user once the port is acquired. Wonderwall uses the current user ID as an identification when binding to servers. See the `-inetd` switch. On Windows NT, this option only affects the user ID used to bind to servers.
`-v`	Print version information for `iiopproxy`.

Using iiopproxy

When testing Wonderwall, the `iiopproxy` can typically be run from the command line, as follows:

```
iiopproxy -debug 2 -config iiopproxy.cf -log iiop.log
```

where the configuration file is called `iiopproxy.cf` and the output is logged to the file `iiop.log`. When running `iiopproxy` from `inetd(8)`, you would typically use a command line similar to the following on UNIX:

```
iiopproxy -inetd -config /etc/iiopproxy.cf
```

The `inetd` mode is not available on Windows NT.

Wonderwall sends its output to the system log by default.

For recommendations on how to set up the `iiopproxy` to run as a daemon process from `inetd(8)`, refer to Appendix C.

The iortool Utility

Wonderwall requires a certain amount of manipulation and use of IORs. In particular, the administrator of the Wonderwall needs to maintain a database of objects both in their original form (for use behind the firewall) and in their proxified form (for use by remote clients). To make this task easier, Wonderwall is shipped with the `iortool` utility which helps in the reading and editing of IORs.

The `iortool` utility is a general purpose tool which allows you to view, edit, and create IORs. It can be used with IORs generated by any ORB. Some of its features, however, are specific to Orbix.

Syntax of iortool

The syntax of the `iortool` utility is as follows:

```
iortool {-ior | -rxr | -view | -long | -xlong} iorfile
iortool {-ior | -rxr | -view | -long | -xlong} \
   [-proxy [-host host] [-port port]] iorfile
iortool {-ior | -rxr | -view | -long | -xlong} -manual
```

The `iortool` utility is mainly used for the following:

- To view the IOR which is stored in *iorfile*.
- To edit the IOR in `iorfile`.
- To create a new IOR where the user is prompted for input at each stage in the creation of the IOR.

For further information, refer to "Using the iortool Utility" on page 90.

The options supported by the `iortool` utility are as follows:

`-host` *host*	Used, in conjunction with the `-proxy` option, to specify the new proxy *host* which will be embedded in the IOR. If this option is not specified, the host in the IOR is left unchanged.
`-ior`	Specifies that the IOR should be printed in CORBA standard stringified format.
`-long`	Specifies that the IOR should be printed in a long readable format.
`-manual`	Used to create an IOR interactively. The `iortool` proceeds to create a new IOR and the user is prompted along the way to provide all of the information needed.
`-port` *port*	Used, in conjunction with the `-proxy` option, to specify the new proxy *port* number which will be embedded in the IOR. If this option is not specified, the port in the IOR is left unchanged.
`-proxy`	Used to specify that the `iortool` is being used to edit the IOR in *iorfile*. This option is intended to be followed by either `-h` *host* or `-p` *port*, or both. It allows the user to easily create a proxified IOR from the given *iorfile*—by specifying both the proxy host (using `-host` *host*) and the proxy port (using `-port` *port*).
`-rxr`	Specifies that the IOR should be printed in (Wonderwall specific) a readable hex representation RXR format.
`-view`	Specifies that the IOR should be printed in a one-line readable format.
`-xlong`	Specifies that the IOR should be printed in a very long readable format, with the object key and any unrecognised components dumped using the traditional hex dump format instead of RXR format.

Using the iortool Utility

One way of using the `iortool` utility is as a tool for translating IORs between different formats. There are five output formats which can be selected using one of the following flags: `-ior`, `-rxr`, `-long`, `-xlong` or `-view`.

For example, given an IOR stored in the `gridiiop.ref` file of the Wonderwall `tmp` directory, it can be printed out in the standard IIOP stringified format using the `-ior` flag as follows:

```
% iortool -ior /tmp/gridiiop.ref
IOR:000000000000000d49444c3a677269643a312e3000000
0000000000100000000000000004c0001000000000015756c74
72612e6475626c696e2e696f6e612e6965000000096300000002
83a5c756c7472612e6475626c696e2e696f6e612e69653a67
7269643a303a3a49523a67
```

The `-rxr` option writes out the IOR in readable hex format RXR (refer to “Representations of an IOR” on page 44 for full details of this format). The RXR format is specific to the Wonderwall. An example of RXR format is as follows:

```
% iortool -rxr /tmp/gridiiop.ref
RXR:_______%0dIDL:grid:1.0_______%01_______L_%01_
____%15ultra.dublin.iona.ie__%09c___(:%5cultra.du
blin.iona.ie:grid:0::IR:grid_
```

The `-view` option writes the IOR in human readable format. This option can only be used to parse Orbix IORs. For example:

```
% iortool -view /tmp/gridiiop.ref
[IOR: type “IDL:grid:1.0”: [IIOP1.0
host=ultra.dublin.iona.ie port=2403 \ [ObjectKey
“RXR::%5cultra.dublin.iona.ie:grid:0::IR:grid_:”] ]]
```

Another way of using an IOR is to edit an existing IOR. This is done via the `-proxy` option (in addition to the output format specifier which is either `-ior`, `-rxr` or `-view`) which is used in conjunction with the `-host` and `-port` options.

For example:

```
% iortool -x -h host.iona.com -p 1570 -i /tmp/
gridiiop.ref
IOR:000000000000000d49444c3a677269643a312e3000000
00000000010000000000000048000100000000001270726f
7879686f73742e696f6e612e69650006220000000283a5c756
c7472612e6475626c696e2e696f6e612e69653a677269643a
303a3a49523a6772696400
```

The new IOR will have the specified host and port number embedded in it. These become the host and port which the client will attempt to connect to when it uses the new IOR.

Finally the `-manual` option can be used to create an IOR from scratch. For example, if you enter the following command:

```
% iortool -manual -ior
```

you will be prompted to enter each of the requisite fields of an IOR (as specified by the IIOP standard). The resulting IOR is written to the standard output as an IIOP stringified IOR, as specified by the `-ior` option.

Note: This is not the standard way of producing IORs. It is recommended that novice users avoid this option altogether.

Appendix B Configuration

Each installation of Wonderwall includes a configuration file that allows you to specify how applications use Wonderwall security. At the heart of the Wonderwall configuration is its configuration file `iiopproxy.cf` which specifies the security policy for your system.

The first stage in setting up the Wonderwall firewall, is creating the Wonderwall configuration file `iiopproxy.cf`. This file is read in by the firewall server `iioproxy` during startup. Subsequent changes made to `iiopproxy.cf` will affect new clients—any existing client sessions will not be affected by the changes.

The `iiopproxy.cf` file takes the format of a standard UNIX configuration file—it is read line-by-line, anything between a '#' (number sign) and the end of a line is ignored as a comment, and entries can be continued onto the next line using the '\' (backslash) character. The configuration entries are also case-sensitive. A sample configuration file can be found in "Example iiopproxy.cf File" on page 16.

It is convenient to divide the contents of the configuration file into three parts as follows:

- Basic Settings.
- List of IORs.
- Access Control List.

This appendix provides a complete description of all configuration settings.

Basic Settings

`activated-port-range` *lo hi*

The TCP/IP port range (inclusive) that an internal server can use. The default lower range is 1024, and the default upper bound is 65535. If you wish to tighten security, you can restrict the port range to whatever is used in the internal hosts' `Orbix.cfg` files in the `IT_DAEMON_SERVER_BASE` parameter.

`alias-hosts {hostname} [{hostname2}...]`

This configuration parameter allows multiple aliases for Wonderwall's hostname to be listed. If Wonderwall receives an invocation for an object on these hosts, or receives a HTTP message directed for one of these hosts, it will recognise it as referring to itself. The default value is the real hostname (and any aliases it may have in DNS).

`allow-binary-principals` *boolean*

If the value of `boolean` is `on`, then principals (usernames attached to incoming IIOP requests) are sanitised by Wonderwall for server activation purposes, and any non-username characters cause the principal to be replaced with the string `iiopproxy`. Non-username characters are alphanumeric characters, plus the characters _, -, +, =, `.` and `,`.

`domain dns-`*domain*

The DNS domain name used for Wonderwall's hostname should be specified here.

`hostname` *hostname*

Specifies the hostname (or IP address) of the machine that Wonderwall is running on. If this is specified on a machine with multiple IP interfaces, Wonderwall will bind to the interface with that hostname. If it is left unset, the hostname will be determined automatically.

`http-files` *directory*

Sets the directory under which the Wonderwall proxy searches for files when behaving as a HTTP server. The files are fetched in response to HTTP requests incoming on the port specified by `http-port`. If this parameter is not set, no files can be retrieved through the HTTP server (although HTTP tunnelling will still be possible).

`http-idiosyncrasy` *user-agent idiosyncrasy [...]*

Unfortunately, the HTTP support built into the browser's Java runtime is not always bug-free. As a result, Wonderwall may need to be informed of bugs present in certain versions.

The user-agent string indicates the value used by the Java runtime to identify that particular browser, and is usually (but not always) in the format *BrowserName/version*; for example, `JDK/1.1`. Simple glob-style pattern matching can be used here, so `JDK/*` matches all versions of the JDK.

The idiosyncrasy parameters use the following keywords:

Keyword	**Description**
`none`	No idiosyncrasies; full HTTP 1.0 or HTTP 1.1 compliance.
`newline-after-post`	Expect to see a redundant newline after every HTTP post operation. Most JDK 1.0.2-derived Java runtimes do this.
`no-keepalive`	Inhibit the use of HTTP Keep-Alive even if the browser says it supports it.

`http-keepalive-timeout` *timeout*

Specifies how long, in seconds, Wonderwall should wait for a new message to arrive from a HTTP 1.1 connection before closing it down, requiring the client to reconnect. The default value is 600 seconds.

`http-port` *httpport*

Sets the port number which Wonderwall uses to receive HTTP requests and HTTP tunnelled IIOP messages. Typically, this port is set to the standard value 80. If it is set to `0`, the HTTP server capability of Wonderwall is disabled.

`iiop-idle-timeout` *timeout*

Specifies how long, in seconds, Wonderwall should wait for a new message to arrive from a client connection before closing it down, requiring the client to reconnect. The default value is 3 hours.

`log [requests] [replies] [request-bodies] [reply-bodies]`

Specifies what additional messages should be sent to the system log file.

Flag	**Information logged**
`requests`	Headers of messages sent by client.
`replies`	Headers of messages sent by server.
`request-bodies`	Contents of all request messages.
`reply-bodies`	Contents of all reply messages.

`log-to-syslog [on|off]`

Specifies whether the UNIX `syslog(3)` daemon should receive a copy of any log messages produced by the Wonderwall. This is enabled by default. (This is applicable only on UNIX.)

`log-facility` *facility*

Specifies the name of the UNIX `syslog(3)` facility which Wonderwall should log its output to. The facility parameter should be set to the name of the facility without the leading `LOG_` prefix, in lowercase. For example, to cause Wonderwall to log using the `LOG_DAEMON` facility, use `log-facility daemon` (This is applicable only on UNIX.)

`masquerade-as-host {hostname}`

If this entry is present, Wonderwall will masquerade as a different hostname. This can be useful if NAT firewalls are used to protect Wonderwall itself. The default setting involves using the real hostname.

`orbixd-iiop-port` *port*

If you are using Wonderwall with Orbix or OrbixWeb servers on the internal network, and you wish to use the `bind` form of `object-specifier` in the Wonderwall configuration file, then it is necessary to specify this port. The port is set to the port which the Orbix daemon uses, on the internal network, to receive IIOP messages. (In the default configuration, this port is 1571. It can also be set explicitly via the environment variable `IT_IIOP_PORT`, in the environment in which the Orbix daemon process runs.)

`ping-as-user {username}`

This parameter allows a username to be specified so that Wonderwall will `ping` any activated servers as a different user. The default value is the username that Wonderwall is running under.

`ping-during-bind [on|off]`

This configuration parameter determines whether Wonderwall should `ping` the Orbix/OrbixWeb servers listed in its configuration file—when the *bind* object syntax is used. If this value is switched off, Wonderwall reads its configuration file more quickly. The default value is `on`.

`port` *port-number*

This allows you to specify which `port-number` Wonderwall will listen on. This value can also be specified using the `-port` command-line parameter when starting the proxy.

`pseudo-orbixd` *boolean*

This option only needs to be set if Wonderwall receives messages from certain older versions of OrbixWeb clients. It specifies whether Wonderwall should emulate specific aspects of an Orbix daemon (`orbixd`) in order to allow clients to connect to Wonderwall-protected servers using `bind()`. The value of `boolean` can be set to either `on` or `off`. The default setting is `off`.

`server-open-timeout` *timeout*

Specifies how long, in seconds, Wonderwall should retry connecting to a server which has been activated by its `orbixd`. The default value is 15 seconds.

`strict-host-matching` *boolean*

If the value of `boolean` is `on`, hostnames will be matched using string comparisons (this is the default). If it is `off`, hostnames will be matched using DNS name resolution.

List of IORs

`allow-unlisted-objects` *boolean*

If the value of `boolean` is on, it allows Wonderwall to dynamically update, and add to its internal table of known objects. For example, if a client attempts to connect to an unknown IOR (not registered using `object`, `server` or `persistent-object`), Wonderwall will automatically add this IOR to its internal list of known objects, assuming `boolean` is on. The default value of `boolean` is `off`.

Note: Just because an IOR is automatically added to this list, does not mean that the client is necessarily granted access. All messages must still be filtered by the Access Control List, in the usual way.

`object` *tag* `[wild` *wildcardflags*`]` *object-specifier*

This entry is used to define a `tag` which is used throughout the configuration file to refer to an object or group of objects. An entry is made in a runtime table which records all objects known to Wonderwall.

At present Wonderwall supports four different forms of `object-specifier`:

- An object-specifier beginning with the keyword "`bind`" is used to specify the object using a pseudo-bind syntax. This closely resembles the syntax of `_bind()` as used by a regular OrbixWeb client. For further information, refer to "Object Specifiers" on page 13.
- An object-specifier beginning with the characters "`IOR:`" introduces an IOR coded as a standard CORBA stringified object reference. For further information, refer to "IOR Format" on page 40.
- An object-specifier beginning with the characters "`RXR:`" introduces an IOR encoded using the readable-hex-representation. This is explained in detail in "IOR Format" on page 40.
- An object-specifier that begins with a "/" is assumed to be the absolute pathname of a file where the IOR is stored (either in "`IOR:`" or "`RXR:`" format).

If the `wild` parameter is specified, any attempts to match this object with a request from the Internet will ignore that aspect of the object key. Since this requires examination of the object key, it is not interoperable and depends on support in the Wonderwall code for the server ORB vendor's object key format. Currently, Orbix and OrbixWeb are supported. The supported parameters for `wild` are as follows:

Wildcard flag	Effect
`host`	Ignore the hostname used in the object key.
`server`	Ignore the server name used in the object key.
`marker`	Ignore the object marker used in the object key.
`ifmarker`	Ignore the interface marker used in the object key.

The `object` entry is suitable for listing an IOR whether the respective server is activated or persistent. When an object is listed as an `object` entry, Wonderwall will use the facilities provided by the IIOP protocol to check the host and port where the server is currently located. Wonderwall will use this new host and port information to forward messages to the server.

This ensures that Wonderwall functions correctly with activated servers. In Orbix, for example, such servers are started automatically and have host and port assigned by the Orbix daemon process. When Wonderwall contacts the daemon via IIOP, it will be told the current host and port of the particular server.

`persistent-object` *symbol* `[wild` *wildcardflags*`]` *ior*

The `persistent-object` entry can *only* be used when the object is in a persistent server. It is almost identical to the `object` entry. The only difference is that, in this case, Wonderwall assumes the listed `ior` specifies a direct connection to the server. A precautionary message, to determine the actual host and port, is not sent in advance of the real request. The advantage is a gain in efficiency when used in conjunction with persistent servers.

`server` *tag object-specifier*

This is exactly equivalent to the entry:

    object tag wild marker ifmarker object-specifier

It generates a tag which can be used to refer to all of the objects common to a particular server (hence the name). It is provided as a standalone keyword because it is useful for tagging Factory objects (refer to "Factory Objects" on page 17).

`client-transformer` *object-specifier*

Specifies the object which implements an external client transformer for Wonderwall. A client transformer will not be used unless this line is present in the configuration file (refer to "Configuration" on page 77).

`server-transformer` *object-specifier*

Specifies the object which implements an external server transformer for Wonderwall. A server transformer will not be used unless this line is present in the configuration file (refer to "Configuration" on page 77).

`use-ipaddr-in-iors` *boolean*

Specifies whether IORs created by Wonderwall should contain the host's IP address or its hostname. This should be set to true if the hostname Wonderwall is running on is not resolvable using DNS to hosts outside Wonderwall's domain.

Access Control List

The most important part of the configuration is the Access Control List (ACL). This specifies which operations can be accessed on which objects, along with extra conditions and flags.

The ACL is read from the first rule encountered to the last, and is processed by the ACL testing code in that order. This means that you can specify broad filters first, to remove potentially dangerous or unknown features such as Service Contexts, and then go on to allow specific operations on objects after that.

There is no limit to the number of rules in the ACL. If no rules match the message, it is blocked.

Each line is treated as one rule. Longer rules can be continued onto consecutive lines using the '\' (backslash) character.

`allow [`*keyword* `[`*parameter*`] ] [`*keyword* `[`*parameter*`]] ...`

This entry will allow any request which matches the given rule. Each specified `keyword` on the line must correspond to the rule to match (sometimes a `keyword` also has an associated `parameter`). If a `keyword` is not specified on a rule line, that aspect of the message is ignored for purposes of the ACL match.

`deny [`*keyword* `[`*parameter*`] ] [`*keyword* `[`*parameter*`]] ...`

This entry will deny any request which matches the given rule. Each specified `keyword` on the line must correspond to the rule to match (sometimes a `keyword` also has an associated `parameter`). If a `keyword` is not specified on a rule line, that aspect of the message is ignored for purposes of the ACL match.

Keywords Used in Rules

The following keywords only appear as parameters to the `allow` or `deny` rules:

`...ipaddr` *ipaddress*`[/`*mask*`]`

Match if the client IP address equals the given `ipaddress`. The optional `mask` parameter specifies a bit-wise mask. Only these bits will be used to compare the IP addresses.

Some common masks are:

255.255.255.255 all bits (this is the default).

255.255.255.0 class C bits.

255.255.0.0 class B bits.

255.0.0.0 class A bits.

Note: This option should not be relied upon to provide security since the client IP address can be faked.

`...log`

If this keyword appears in a rule which successfully matches, the message header details will be written to the system log. This would be redundant if you had `log requests` set in your configuration file as the header would be logged anyway.

`...msgtype` *type*

Match on message types. This type can be one of the following: `Request`, `Reply`, `CancelRequest`, `LocateRequest`, `LocateReply`, `CloseConnection` or `MessageError`. Refer to "Internet Inter-ORB Protocol (IIOP)" on page 46 for more details on IIOP message types.

`...object` *symbol*

Match if the object being accessed is identified by `symbol` (this rule only applies to incoming Requests). The symbol must have already been declared in an earlier `object` or `persistent-object` entry.

`...object-host` *hostname*

Match if the object being accessed is located on the host identified by `hostname`.

`...op` *operation*

Match if the operation in a Request is the same as the operation string `operation` (this rule only applies to incoming requests).

`...principal` *p*

Match if the `principal` sending a request message is the same as the readable-hex-representation byte string `p` (refer to "IIOP Message Formats" on page 47 for further information on this format).

Note: This parameter does *not* provide any security as principals are very easy to forge.

`...servicecontexts` *sclist*

Match if the incoming request uses one or more IOP Service Contexts and if one or more of the Service Context IDs used is listed in the `sclist` parameter. IOP Service Contexts are a mechanism which, according to the IIOP specification, allows "service-specific context information" to be passed along with requests and replies. In keeping with the firewall philosophy of "anything not expressly permitted is denied", it is suggested that these are filtered out until a future stage when they become necessary[1] at which point each can be enabled on a specific basis.

The format of the `sclist` parameter is as follows:

- Each Context ID is represented as a positive integer. These integer IDs are assigned by the Object Management Group to uniquely denote a particular type of Service Context.
- Multiple Context IDs can be listed, separated by ',' (comma) characters.
- A range of Context IDs can be matched by listing the start ID, a '-' (dash) character, and the end ID.
- The string `all` is used to match one or more Context IDs, and the string `max` is used to denote the upper bound of the Context ID range.

For example:

```
deny servicecontexts 1-3,5,7,9-20
deny servicecontexts 0-5,7-max # all except 6
deny servicecontexts all # one or more Service Contexts
```

Note: If a rule allowing specific service contexts is followed by a wildcard deny rule, the effect is non-intuitive. A request containing both permitted and denied service contexts would be forwarded, as it would hit the `allow` rule first. Caution is advised here.

1. A service context is used, for example, by the CORBA Transaction Service.

```
...unlisted-object
```

Match if the object being accessed is an unlisted object. That is, if it has been dynamically specified as a target by the client-side ORB.

```
...proxify
```

Proxify the returned object reference produced by this operation.

Appendix C
Firewall Installation on UNIX

In this configuration, it is assumed that the installer wants as much control as possible over the Wonderwall setup.

To install Wonderwall:

- Copy the `iiopproxy` executable into whatever directory is used to store the binaries of your firewall proxies.
- Copy the `iiopproxy.cf` configuration file into the `/etc` directory or whatever directory you feel is appropriate.
- It is recommended that you create a directory to store the IORs used by the configuration file.

  For example, a sub-directory of the directory where the configuration file is stored can be used:

  ```
  /etc/iors
  ```
- Set up the starting mechanism for the Wonderwall process.

  You can do this before editing the configuration file since the configuration which comes with the distribution blocks all messages by default.
- Decide whether to run the Wonderwall from `inetd(8)` or as a standalone process.

  For most purposes, running Wonderwall from `inetd` is sufficient, but if you expect your cross-firewall IIOP usage to be particularly heavy, you can run it standalone.
- Pick a port to run Wonderwall on.

  For example, you can use the official Orbix daemon port 1570.

Setting Permissions

The IIOP proxy needs to be able to perform the following system interactions:

- It must be able to read its configuration file and any IORs specified therein.
- If not running from `inetd`, it must be able to bind to the port.

The IIOP proxy does not need to be able to open or write to any area of the file system, nor does it need to execute commands. On a normal UNIX system, the standard user `nobody` should be appropriate for these purposes.

External IIOP clients need to be able to contact Wonderwall on port `1570/tcp` (or whatever port is used in the configuration file). They also need to be able to open and use connections to the internal hosts named in the configuration file in order to contact Orbix daemons and servers running on these hosts.

Note: If the `persistent-object` keyword is used throughout the configuration file instead of the `object` keyword, the Orbix daemon does not need to be accessible.

In order to resolve the names of client hosts, it is preferable for Wonderwall to be able to contact external DNS servers (`port 53/udp`). However, if this is not possible, the hostnames of connecting clients will not be logged and no bad side effects will occur. Information from external DNS servers is not used in any security-critical context.

Setting up the Proxy From inetd

To run the proxy from `inetd`:

- Add the following line to `/etc/inetd.conf`:

  ```
  iiop stream tcp nowait nobody /usr/local/etc/
  iiopproxy iiopproxy -inetd -config /etc/
  iiopproxy.cf
  ```

Note: This line should not contain a line-break, there is no white space before or after this line, and you may need to change the paths if the Wonderwall binary and/or configuration file is installed in a different location.

- Add the following line to `/etc/services` (the white space between `iiop` and `1570` should be a tab):

  ```
  iiop 1570/tcp
  ```

To cause `inetd(8)` to read its configuration again, find its process ID using `ps` and `kill -1` it. Do this on Solaris 2.x or other SVR4-based systems as follows:

```
% ps -ef | grep inetd
root 117 1 0 09:18:38 ? 0:00 /usr/sbin/inetd -s
```

The PID is the second argument.

```
% kill -1 117
And here’s how to do it on SunOS 4.1.x, Linux,
or other BSD-based systems:
% ps auxww | grep inetd
root 117 0.0 0.5 1848 1200 ? S 09:18:38 0:00 /usr/etc/inetd
% kill -1 117
```

Setting up the Proxy to Run Standalone

If you choose to run Wonderwall standalone, things are easier. Run the `iiopproxy` binary and it will start listening for new connections on whatever port is specified in the configuration file. In order to ensure it has started when the machine boots, you will need to add its invocation to the system boot scripts.

Index