Orbix Administrator's Guide C++ Edition

IONA Technologies PLC
September 2000

M 2 4 6 6

Contents

Preface

Orbix is a software environment for building and integrating distributed, object-oriented applications. This guide explains how to configure and manage the components of the Orbix environment. Many Orbix components have associated graphical user (GUI) interfaces. This guide describes the Orbix GUI tools associated with Orbix configuration, the Implementation Repository, and the Interface Repository.

Orbix documentation is periodically updated. New versions between releases are available at this site:

`http://www.iona.com/docs/orbix/orbix33.html`

If you need assistance with Orbix or any other IONA products, contact IONA at `support@iona.com`. Comments on IONA documentation can be sent to `doc-feedback@iona.com`.

Audience

Read this guide if you are responsible for any of the following tasks:

- Configuring an Orbix installation.
- Registering servers in the Orbix Implementation Repository.
- Adding IDL definitions to the Orbix Interface Repository.

This guide describes how you can use the command line and Orbix GUI tools. It assumes that you are familiar with relevant sections of the *Orbix Programmer's Guide C++ Edition*, and the *Orbix Reference Guide*. Before reading this guide, you should read the *Introduction to Orbix C++ Edition* manual.

Organization of this Guide

This guide is divided into three parts as follows:

Part 1, Orbix C++ Administration

Chapter 1, "Overview of Orbix Administration"

This chapter introduces the main components of the Orbix environment. You should read this chapter first to familiarize yourself with terminology used throughout the guide.

Chapter 2, "Getting Started"

This is a quick start chapter on how to configure Orbix, start the Orbix daemon process, and how to register a server that automatically starts when it is needed.

Chapter 3, "Managing the Implementation Repository"

This explains more about using the Implementation Repository including registering servers, displaying and organising server entries, and security issues.

Chapter 4, "Managing the Interface Repository"

This chapter describes how to configure Orbix to store object interface definitions so that applications can learn about them at runtime.

Part II, Orbix C++ GUI Tools

Chapter 5, "The Orbix Configuration Explorer"

This chapter describes how you can configure an Orbix installation using the Orbix Configuration Tool.

Chapter 6, "The Orbix Server Manager"

This chapter describes how you can register servers in the Orbix Implementation Repository using the Orbix Server Manager.

Part III, Appendices

Appendix A, "Configuration Variables"

This appendix shows the configuration variables that Orbix recognizes.

Appendix B, "Orbix Daemon Options"

This appendix describes the start-up options that the Orbix daemon can use.

Appendix C, "Command Reference"

This describes the syntax and the options for each Orbix command you can use.

Appendix D, "Error Messages and Exceptions"

This describes how to modify error messages, shows the error formats, and lists tables of standard error messages that Orbix applications can return.

Document Conventions

This guide uses the following typographical conventions:

`Constant width`	Constant width (courier font) in normal text represents portions of code and literal names of items such as classes, functions, variables, and data structures. For example, text might refer to the `CORBA::Object` class. Constant width paragraphs represent code examples or information a system displays on the screen. For example: `#include <stdio.h>`
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 to commands or path names for your particular system. For example: `% cd /users/`*`your_name`* Note: Some command examples may use angle brackets to represent variable values you must supply. This is an older convention that is replaced with *italic* words or characters.

This guide may use the following keying conventions:

No prompt	When a command's format is the same for multiple platforms, no prompt is used.
%	A percent sign represents the UNIX command shell prompt for a command that does not require root privileges.
#	A number sign represents the UNIX command shell prompt for a command that requires root privileges.
>	The notation > represents the DOS, Windows NT, or Windows 95 command prompt.

... ⋮	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.

Note: Unless otherwise stated, all examples in this guide apply to Orbix on both UNIX and Windows platforms.

Part I

Orbix C++ Administration

1

Overview of Orbix Administration

Orbix is a software environment that allows you to develop distributed applications. This chapter introduces the main components of the Orbix environment.

As described in the *Orbix Programmer's Guide C++ Edition*, Orbix allows you to build distributed software systems composed of interacting objects. Orbix is a full implementation of the Object Management Group (OMG) Common Object Request Broker Architecture (CORBA).

An Orbix application consists of one or more *client* programs that communicate with *distributed objects* located in *server* programs. Clients can communicate with distributed objects from any host in a network through clearly-defined interfaces specified in the CORBA Interface Definition Language (IDL).

Orbix mediates the communication between clients and distributed objects. This mediation allows clients to communicate with objects without concern for details such as:

- The hosts on which the objects exist.
- The operating system that these hosts run.
- The programming language used to implement the objects.

The Orbix architecture includes several configurable components that support the mediation of communications between clients and objects.

Components of the Orbix Architecture

An Orbix client invokes IDL operations on a distributed object using normal C++ function calls, as if the object were located in the client's address space. Orbix converts these function calls to a series of network messages and sends these messages to the server process that contains the target object. At the server, Orbix receives these messages and translates them to function calls on the target object, as shown in Figure 1.1.

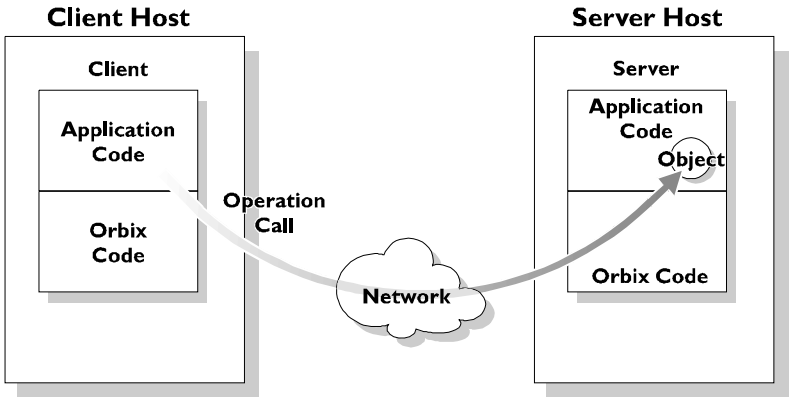

Figure 1.1: *An IDL Operation Call on a Distributed Object*

Servers and the Implementation Repository

Each Orbix server program has a name, unique within its host machine. A server can consist of one or more processes. When a client invokes a method on an object, a server process containing the target object must be available. If the process is not running, the Orbix daemon at the server host attempts to launch the server process automatically.

To allow an Orbix daemon to manage the server processes running in the system, Orbix provides an *Implementation Repository*. The Implementation Repository maintains a mapping from a server's name to the filename of the executable code implementing that server. The server code must therefore be registered with the Implementation Repository.

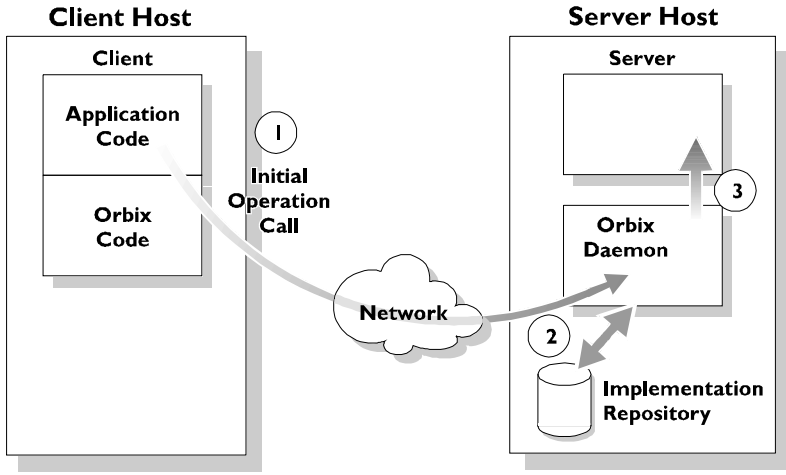

Figure 1.2: *Automatic Launch of an Orbix Server Process*

As shown in Figure 1.2, the Orbix daemon launches a server process as follows:

1. A client makes its first operation call to an object located in a server.
2. The Orbix daemon reads the server details from the Implementation Repository, including the server launch command.
3. If the required server process is not running, the Orbix daemon executes the server launch command.

To allow the daemon to launch server processes, you must maintain records in the Implementation Repository for each server in your system.

The Interface Repository

Orbix maintains object specifications by storing an object's IDL interface in a database called the *Interface Repository*. Some client applications use the Interface Repository to determine object interfaces and all information about those interfaces at runtime.

A client accesses the Orbix Interface Repository by contacting an Interface Repository server. This is a standard Orbix server that provides a programming interface, defined in IDL, to the Interface Repository.

To allow clients to obtain information about IDL definitions implemented in your system, you must add those definitions to the Interface Repository.

Administration of Orbix Components

To allow Orbix applications to run in your network, you must do the following:

- Configure Orbix for your network and environment, using the Orbix configuration files.
- Run the Orbix daemon process.
- Register servers in the Implementation Repository.

Part I Orbix C++ Administration presents the configuration files and command-line utilities that allow you to achieve each of these tasks.

Part II Orbix C++ GUI Tools presents the graphical user interfaces that provide an alternative way to manage Orbix components.

2

Getting Started

Several components of Orbix require administration. This chapter describes the basic Orbix administration steps required when running Orbix applications.

Orbix administration involves the following basic steps:

1. Configuring Orbix for your network and environment.
2. Starting the Orbix daemon (`orbixd`) on each host that Orbix servers run on.
3. Registering servers in the Implementation Repository so that Orbix can start them when needed.
4. Starting client applications that make object requests.
5. Monitoring Orbix to fine tune it and your clients and servers.

Steps 1 and 2 apply when you first install Orbix and only occasionally after that. Steps 3, 4, and 5 are iterative. This guide describes how to perform these steps. This chapter first gives you a quick start to using Orbix and its environment of distributed computing.

Basic Orbix Configuration

This section describes the configuration settings you may need to modify before starting the Orbix daemon. You can modify the main Orbix configuration settings by editing the Orbix configuration files, or by setting environment variables or by using the Orbix GUI tools. Refer to "The Orbix Configuration Explorer" on page 45 for details of configuring Orbix using GUI tools.

The paths in the following examples are for an NT installation. Configuration files for Unix should use the Unix syntax for directories. All values provided must be enclosed in double quotes (" "). Each line, except for blank lines and comments, must be terminated with a semi-colon (;).

The Orbix Configuration Files

The Orbix configuration files are located in the `config` directory of your Orbix installation. By default, these are named as follows:

- `iona.cfg`
- `common.cfg`
- `orbix3.cfg`
- `orbixnames3.cfg`

iona.cfg

The `iona.cfg` file is the root configuration file used by Orbix. This file contains links to all other IONA configuration files. You can edit this file to include links to your customized configuration files. The default `iona.cfg` file includes the following information:

```
// In file iona.cfg
cfg_dir = "d:\iona\config\";

include cfg_dir + "common.cfg";
include cfg_dir + "orbix3.cfg";
include cfg_dir + "orbixnames3.cfg";
```

You should set the `config_dir` variable to `<iona_install_dir>\config\`.

common.cfg

The common.cfg file contains a list of configuration variables that are common to multiple IONA products. The configuration variables in this file are declared within the scope Common{...}, for example:

```
// In file common.cfg
Common {
   # The port number for the Orbix daemon.
   IT_DAEMON_PORT = "1570";

   # The starting port number for daemon-run servers:
   IT_DAEMON_SERVER_BASE = "1570";

   # The full path name of the Implementation
   #Repository directory.
   IT_IMP_REP_PATH = cfg_dir + "Repositories\ImpRep";

   # The full path name of the Interface Repository
   #directory.
   IT_INT_REP_PATH = cfg_dir + "Repositories\IFR";

   # The local DNS domain name.
   IT_LOCAL_DOMAIN = "";

   # The full path name to the JRE binary
   # executable that installs with Orbix.
   IT_JAVA_INTERPRETER="C:\IONA\bin\jre.exe";

   # The default classpath to be used when java
   # servers are automatically launched by the daemon.
   IT_DEFAULT_CLASSPATH = cfg_dir +
   ";C:\IONA\bin\bongo.zip;C:\IONA\bin\marimba.zip;
    C:\IONA\bin\NSclasses.zip;C:\IONA\bin\utils.zip;
    C:\IONA\bin\rt.jar;C:\IONA\bin\orbixweb.jar;
    C:\IONA\Tools\NamingServiceGUI\NSGUI.jar";
};
```

Note: You can also use the prefix Common. to refer to individual entries in this file. For example, Common.IT_DAEMON_PORT.

After installation, the common.cfg file provides default settings for the main environment variables required by Orbix. You can change these default settings by manually editing the configuration file, or by using the Configuration Explorer, or by setting a variable in the user environment. An environment variable, if set, takes precedence over the value set in the configuration file. Environment variables are not scoped with a Common. prefix.

Format of Configuration Files

Each line of the common.cfg configuration file has the following form:

```
<entry name> = "<entry value>"
```

Each variable in your configuration file must start at the beginning of a line. Any line that does not start with a variable that Orbix recognizes is ignored. You can add comments to your configuration file in this way. Any entry value can use any desired environment variable.

orbix3.cfg

This file contains configuration varibles that are specific to Orbix only. By default, the configuration variables in this file are scoped with the Orbix. prefix. You can also use the scope Orbix{...}.

```
// In file orbix3.cfg
# The path name to the error messages file.
Orbix.IT_ERRORS = cfg_dir + "ErrorMsgs";

# The maximum number of retries Orbix makes to
# connect to a server.
Orbix.IT_CONNECT_ATTEMPTS = "10";
```

Note: Orbix uses the IT prefix, which represents "IONA Technologies", to distinguish its configuration and environment variables.

The orbixnames3.cfg file contains configuration variables that are specific to OrbixNames. Refer to the *OrbixNames Programmer's and Administrator's Guide* for more details.

Locating the Configuration Files

Orbix must be able to find its root configuration file before the Orbix daemon, the IDL compiler, or application processes run. The Orbix config directory is the default location for all configuration files. You can set a different directory or configuration file by setting the IT_CONFIG_PATH environment variable. If the IT_CONFIG_PATH variable is a directory, that directory should contain the iona.cfg file. If the IT_CONFIG_PATH environment variable is the full path name of a file, that file is used as the configuration file.

How Orbix Finds its Configuration

Orbix has a chain of configuration handlers that it looks in when asked for a configuration parameter. These are as follows (in order):

```
[Environment Handler (IT_Environment)] ->
[ScopedConfigFile Handler (IT_ScopedConfigFile)] ->
[OldConfigFileHandler (IT_ConfigFile)]
```

The Environment handler allows any configuration variables defined in your environment to take precedence over those defined in configuration files or other user-defined configuration handlers.

The ScopedConfigFile handler does the following when searching for the root configuration file (iona.cfg by default):

- Checks the environment variable IT_IONA_CONFIG_FILE.

  The configuration file does not need to be called iona.cfg.
- Checks the environment variable IT_CONFIG_PATH and appends iona.cfg.
- Searches for iona.cfg in the same directory as the Orbix runtime libraries.
- On Windows NT, checks the Registry to find where Orbix was installed and appends config\iona.cfg.
- Tries the default installation locations (c:\iona on Windows NT, or /opt/iona on UNIX systems).

The OldConfigFileHandler enables you to use Orbix.cfg files for backwards compatibility. However, it is recommended that you use the default files supplied with this version of Orbix.

The following sections describe more about the IT_DAEMON_PORT, IT_IMP_REP_PATH, and IT_LOCAL_DOMAIN variables. The IT_DAEMON_SERVER_BASE, IT_ERRORS, and IT_INT_REP_PATH variables are described in later chapters of the guide.

Locating the Orbix Library Directory on UNIX Platforms

On Solaris platforms, you must set the environment variable LD_LIBRARY_PATH to include the Orbix lib directory before the Orbix daemon, the IDL compiler, or the Orbix administration commands can run.

On HPUX platforms, you must set the SHLIB_PATH environment variable to include the Orbix lib directory.

In addition, you need to force the program to first search for libraries using the SHLIB_PATH environment variable by using the chatr command (see the chatr(1) man page).

The SHLIB_PATH environment variable must be given precedence for searching over the internal build list.

Setting the Orbix Daemon Port

Orbix uses the daemon process orbixd on each site running Orbix servers to await incoming requests for server activation and to connect new clients to existing server processes. This is not involved in subsequent client/server communications.

The daemon uses one Internet port, and by default this port number is given by the IT_DAEMON_PORT entry in common.cfg. This is a required variable.

The standard registered port number assigned to orbixd by the Internet Engineering Task Force (IETF) is the internet port number 1570. You must ensure that the IT_DAEMON_PORT number is the same for all of your network hosts.

However, when experimenting with the system, you may wish to install more than one Orbix daemon on a specific machine to isolate a particular set of servers. In this case you must specify a different port for each daemon, by setting the environment variable IT_DAEMON_PORT or by using a different root configuration file iona.cfg.

Locating the Implementation Repository

The data held in the Implementation Repository maps from server, application object, and operation names to the path names of executable server files. The location for storing this data is given by the required entry for `IT_IMP_REP_PATH` in the `common.cfg` configuration file. Each Orbix daemon has an associated Implementation Repository.

Occasionally it might be useful for a group of programmers to have their own Implementation Repository store on a particular host. For example, when running a separate daemon with a different daemon port. You can specify a different location by setting the `IT_CONFIG_PATH` to refer to a configuration file that specifies a different location for the `IT_IMP_REP_PATH` entry or by setting the `IT_IMP_REP_PATH` environment variable to override the one in the configuration file.

Specifying Your Local Internet Domain

You can specify the name of the local Internet domain by using the `IT_LOCAL_DOMAIN` variable.

An example is:

```
IT_LOCAL_DOMAIN iona.com
```

A value for this variable is not always required—however, it is advisable to provide one. For example, it is required if both the host's full name (for example, `alpha.iona.com`) and abbreviated name (for example, `alpha`) are used in Orbix applications.

Using the dumpconfig Utility

The `dumpconfig` utility enables you to obtain information about your Orbix configuration. This utility outputs the values of the configuration variables used by Orbix, and the location of the Orbix configuration files in your system. It also reports if there are any syntax errors in your configuration files that would normally go unrecognized by Orbix. The `dumpconfig` utility is especially useful if you need to know where Orbix is being configured from.

Starting The Orbix Daemon

An Orbix daemon runs on each host to control aspects of the distributed system. The daemon is responsible for the following tasks:

- Starting servers when appropriate.
- Connecting clients to servers.
- Managing the Implementation Repository. The daemon accepts requests from the Orbix Implementation Repository commands.
- Providing information from the Interface Repository about the supported interfaces for clients that request it.

A typical start of the Orbix daemon without options is as follows:

```
orbixd
```

Running the Orbix Daemon as an NT Service

On Windows NT platforms, you can install the Orbix daemon as an NT service as follows:

```
> orbixd -j
```

You must manually start the service on Windows NT platforms as follows:

1. Select **Start→Settings→Control Panel→Services**.
2. Highlight the Orbix daemon entry.
3. Click the **Start** button.

   NT starts the service as <path>\orbixd -b.

To uninstall this service on Windows NT platforms, do the following:

```
> orbixd -w
```

Using the -o Option to the Orbix Daemon

You should use the `-o` option if you are running `orbixd` as a super-user on UNIX platforms. This option indicates that if the daemon runs with super-user privileges, servers launched by the daemon should run using the specified user ID instead of the root ID.

You should run `orbixd` in this way for the following reasons:

- A client running as root on a remote machine could launch a server with root privileges on a different machine. This poses a serious security risk because a remote user could easily be faked. When the Orbix daemon is launched as `orbixd -o` *userId*, servers launched by the daemon run using the specified user ID instead of the root ID.
- When the daemon has super-user capabilities, the permissions of servers are indeterminate and depend on the permissions of the first remote user to start a specific server. For example, on UNIX the files written by a server may have different owners on different activations making it possible that the server would be unable to read or write files in future activations.

Refer to Appendix B, “Orbix Daemon Options” on page 85 for more details.

Note: Any changes you make to the configuration of Orbix do not take effect until you restart the Orbix daemon.

Registering a Server

The `putit` utility registers servers with the Orbix Implementation Repository. You can use the `putit` command in its simplest form as follows:

```
putit server_name command_line
```

For example:

```
putit BankSrv /usr/users/chris/banker
```

The executable file /usr/users/chris/banker is registered as the implementation code for the server called BankSrv at the local host. You should use the full path name and not a relative path name. This is because Orbix interprets relative path names with respect to the Orbix daemon's current directory, not the putit user's current directory.

The putit command does not execute the indicated file. The file is automatically launched by Orbix in response to an incoming operation invocation.

Note: You should ensure that the server name specified in the putit command matches exactly the server name used in the server application code.

Checking for an Orbix Daemon

Use the pingit utility to determine if an Orbix daemon is running on a particular host. For example:

```
pingit -h host_name
```

If the Orbix daemon is running at the target host, pingit displays a message to indicate this. Otherwise, pingit displays a CORBA COMM_FAILURE exception message.

Checking for Running Servers

Use the psit utility to display information about all of the running servers that a particular Orbix daemon knows about.

One line is output for each server process. Each line of output has the following fields:

Name	The server name.
Marker	The object marker pattern associated with the process.
Code	The data encoder used; for example, XDR.

Comms	The communications protocol used; for example, TCP.
Port	The port number used by the communications system.
Status	One of “automatic”, “manual” or “inactive”.
Per-Client?	Indicates whether the server is a per-client server.
OS-pid	The operating system process.

Configuring Orbix for Multi-Homed Hosts

Some machines have multiple hostnames or IP addresses (for example, those using multiple DNS aliases or multiple network cards). These machines are often termed multi-homed hosts. A multi-homed host is a host with one or more IP addresses.

Orbix is multi-home aware and can be used successfully with various different network and multi-homed package configurations. There are a number of basic configuration changes that need to be made. For all versions of Orbix the configuration is divided between two configuration variables, IT_LOCAL_HOST and IT_LOCAL_ADDR_LIST.

IT_LOCAL_HOST sets the host name a server will use in any IOR that it exports. Setting it to anything other than one hostname or IP address is not supported. IT_LOCAL_ADDR_LIST is used to make a server or client aware of its multi-homed settings. This is a colon-separated list of IP addresses, or hostnames, on which the Orbix servers and the Orbix daemon orbixd can expect to receive invocations.

For Orbix to enable its multi-home capabilities it is also necessary to use the configuration variable IT_ENABLE_MULTI_HOMED_SUPPORT, which must be available to the ORB at startup.

The following example indicates how you should set up the multi-homed configuration on Orbix:

In the file orbix3.cfg

```
Orbix{
IT_ENABLE_MULTI_HOMED_SUPPORT = "YES";
IT_LOCAL_HOST = "10.1.1.0";
IT_LOCAL_ADDR_LIST = "10.1.1.0:10.1.2.0";
};
```

Orbix will use the IP addresses returned by `gethostbyname()` by default so it may not be necessary to enable the multi-homed capabilities, but, if `gethostbyname()` does not return all IP addresses, you will need to use the variables above.

Multi-Homed Configuration Variables

`IT_LOCAL_HOST`

The local host hostname that a server will use in any IOR it exports.

`IT_LOCAL_ADDR_LIST`

A colon-separated list of IP addresses from which the server is willing to accept connections.

`IT_ENABLE_MULTI_HOMED_SUPPORT`

Set this variable to "YES" to enable Orbix for multi-homed machines. It is disabled by default.

For a complete list of Orbix C++ configuration parameters, refer to Appendix A, "Configuration Variables".

Configuring Orbix for Multiple Network Cards on Independent Networks

It is possible to configure a machine with multiple network cards which are interfaces for separate networks. This example is illustrated in Figure 2.1.

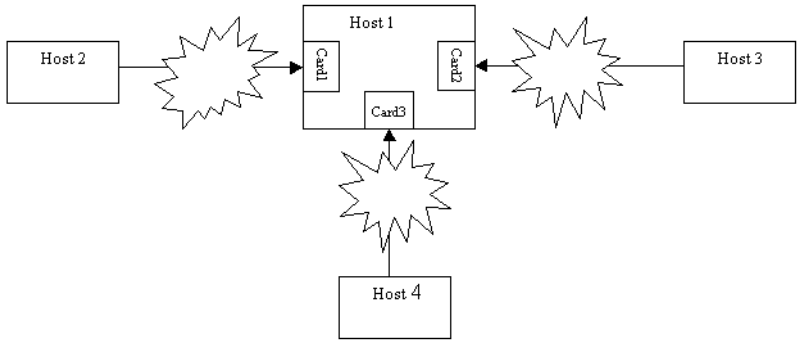

Figure 2.1: *Simple Example of a Multi-homed Host*

The multi-homed host (Host 1) can be on any of the networks in Figure 2.1. All of the networks are completely independent of each other, and any hosts on any of these networks are unaware of those on the other network(s).

The recommended configuration settings for the above example are:

```
Orbix{
   IT_ENABLE_MULTI_HOMED_SUPPORT = "YES";
   // MUST resolve in all domains to the IP address
   // used to connect.
   // No need to set if this is the same as the
   // default name.
   IT_LOCAL_HOST = "host1";
   IT_LOCAL_ADDR_LIST ="card1_IP_address:
             card2_IP_address:card3_IP_address";
   };
```

and

```
Orbix{
   IT_ENABLE_MULTI_HOMED_SUPPORT = "YES";
   // MUST resolve in all domains to the IP address
   // used to connect.
   // This will more than likely will common alias.
   IT_LOCAL_HOST = "host1";
   IT_LOCAL_ADDR_LIST = "card1_host_name:
                card2_host_name:card3_host_name";
   };
```

3

Managing the Implementation Repository

When you install server applications on a network host, you must register those servers in the Orbix Implementation Repository. This repository allows Orbix to direct client operation calls to objects in servers and to start server processes when necessary. This chapter describes how to manage servers in the Implementation Repository.

The chapter covers the following topics:

- The Implementation Repository and its entries.
- Basic usage of the Implementation Repository including registering servers, organizing server entries, removing server entries, listing registered servers, and displaying information about an entry.
- How to start a server manually.
- How to stop servers manually.
- The security of servers including how to change ownership of servers and how to modify access control lists.
- How to register servers in specialized activation modes other than simply one server process for all clients.
- How to manage the set of ports Orbix uses to run servers.

This chapter explains how to manage the Implementation Repository using Orbix command-line utilities. Refer to Chapter , "The Orbix Server Manager" for details of how you can use Orbix GUI tools.

Implementation Repository Entries

The Implementation Repository maintains a mapping from a server's name to the filename of the executable code implementing that server. A server must be registered with the Implementation Repository to make use of this mapping. Orbix automatically starts the server (if it is not already running) when a client binds to one of the server's objects or when an operation invocation is made on any object that names that particular server.

When a client first communicates with an object, Orbix uses the Implementation Repository to identify an appropriate server to handle the connection. If a suitable entry cannot be found in the Implementation Repository during a search for a server, an error is returned to the client.

The Implementation Repository maintains its data in entries that include the following information:

- The *server name*.

  Server names can be hierarchical so the Implementation Repository supports directories.

- The *server owner*—usually the user who registered the server.

- The server *permission values.*

  These specify which users have the right to launch the server and which users have the right to invoke operations on objects in the server.

- One or more *activation orders.*

  An activation order associates an object or group of objects with a launch command. A launch command specifies how Orbix starts the server.

Basic Implementation Repository Usage

Use the putit command to create or modify an Implementation Repository entry. For example, the following command registers a shared server called "FirstTrust" on the local host, with the specified executable file:

```
putit FirstTrust /work/bank/banker
```

Activation occurs when any of the objects managed by the FirstTrust server is used. In this example there is only one server process associated with this server and all clients share the same server process.

Registering a Server on a Remote Host

The following command registers a shared server called "FirstTrust" on the remote host "alpha", with the specified executable file and command-line argument:

```
putit -h alpha FirstTrust
   "/work/bank/banker -v 1.1"
```

Note: If the server requires parameters and options, you should use quotes so that the putit command does not try to interpret them.

Using the -h *hostname* option enables you to use all the utility commands for remote hosts. However, for simplicity, most of the examples in this guide do not use this option and use the local host default instead.

Organizing Servers Into Hierarchies

Server names may be hierarchically structured, in the same way as UNIX file names. Hierarchical server names are useful in structuring the name space of servers in Implementation Repositories. You can create hierarchical directories by using the mkdirit command. For example, you can make a new banking registration directory and make a registration within it as follows:

```
mkdirit banking
putit banking/Berliner /usr/users/joe/banker
```

Thus banking/Berliner is a valid, hierarchical server name.

The rmdirit command removes a registration directory. This command can take a -R option to recursively delete a directory and the Implementation Repository entries and sub-directories within it. The rmdirit command returns an error if it is called without the -R option on a non-empty registration directory.

For example:

```
lsit
   FirstTrust
   banking
rmdirit banking
   directory not empty
rmdirit -R banking
```

This example uses the lsit command to display the Implementation Repository entries and directories.

To move an entry in the hierarchy, first remove it with the rmit command and then re-register it with the putit command.

Removing a Registered Server

Use the rmit command to remove an Implementation Repository entry. For example, the following command removes a server entry:

```
rmit FirstTrust
```

This simplest format of the command removes the entry and all activation orders for the server.

You can also use the rmit command to remove specific activation orders. Use the -marker option for the shared or unshared activation modes to remove specific activation orders for individual objects. Use the -method option for the per-method call activation mode to remove specific activation orders for individual methods. Activation modes are described in section "Server Activation Modes" on page 29.

Listing Registered Servers

Use the lsit command to list registered servers and directories. For example:

```
# Register a server called International and
# one called printer
putit International /usr/users/joe/banker
putit printer /usr/users/joe/print  laser
# Register a server called Berliner.
# "Berlin 98-00-00" are parameters for the
# executable file.
putit Berliner
   /usr/users/joe/banker Berlin 98-00-00
lsit
   International
   Berliner
   printer
```

Use the -R option with the lsit command to recursively list all server entries in the given directory and its subdirectories.

Displaying A Server Entry

Use the catit command to display information about a specific server's registration entry. The following example assumes the server Berliner is registered from the previous example:

```
catit Berliner
   name:          Berliner
   Activation:    shared
   Owner:         smith
   Launch:        ;jones;developers;friends;
   Invoke:        ;all;
   Per-client:    false

   Marker  Launch_Command
   *       /usr/users/joe/banker Berlin 98-00-00
```

The output includes the following:

name	Server name.
Activation	Activation mode.

`Owner`	The user who put the in the entry.
`Launch`	The users and groups who have permission to start or launch the server.
`Invoke`	The users and groups who have permission to invoke operations on an object controlled by the server.
`Per-client`	A per-client indicator that indicates whether a new server is to be launched for each client that uses the server.

The final output is a table of activation orders. An activation order is identified with a marker. An asterisk (*) represents all objects and means that there is only one activation order for the server entry.

Starting Servers Manually

Most servers are designed to have Orbix start them automatically when a client uses an object. The majority of an administrator's work therefore involves registering servers in the Implementation Repository and managing the registration entries in the repository. However, some servers do need to be started before any clients attempt to use their objects.

Servers that are started by some mechanism external to Orbix are useful for a number of reasons. For example, if a server takes a long time to initialize and it starts when a client request a service, it may cause the client to timeout. In addition, some servers that are meant to run as long-lived daemons may require manual starting. Manual servers are also known as *persistent servers* in CORBA terminology.

Registering a Manual Server

All servers that are registered in the shared mode can also be started manually. Subsequent invocations on the objects are passed to the running process.

However, if you wish to prevent Orbix from starting a server and make it manual-only, use the following command:

```
putit FirstTrust -persistent
```

This command registers a manual-only server called "FirstTrust" on the local host. No start command is specified to putit, because this server cannot be started by Orbix automatically but can only start as a manual server.

The CORBA specification requires that unshared or per-method types of servers fail if an attempt is made to start them manually. This means that manual servers can only be registered as shared servers. Therefore, you cannot use the -persistent option with either the -unshared or -per-method options of the putit command. These unshared and per-method servers are described in section "Server Activation Modes" on page 29.

Starting the Orbix Daemon for Unregistered Servers

In some circumstances, it can be useful not to register servers with the Implementation Repository. Under normal operation, Orbix would know nothing about these servers. However, if you invoke the Orbix daemon with the -u option, it maintains an active record of unregistered Orbix servers and clients that may use these servers:

```
orbixd -u
```

When Orbix is started this way, any server process can be started manually. However, no access control is enforced and there is no record of the server in the Implementation Repository.

Stopping Servers

Just as most servers start automatically when needed, they are usually designed to stop automatically after some period. However, there may be other situations where you need to manually stop a server.

The killit command stops a server process by using the SIGTERM signal.

1. For example, the following command stops the Berliner server on the host omega:

   ```
   killit -h omega /Banking/Berliner
   ```

2. When there is more than one server process, use the marker option and argument to distinguish between different processes. To do this, use the following killit command format:

   ```
   killit -m marker server_name
   ```

Security of Registered Servers

For each Implementation Repository entry, Orbix maintains two access control lists (ACLs) as follows:

Launch	The users or groups that can launch the associated server. Users on this list, and users in groups on this list, can cause the server to be launched by invoking on one of its objects.
Invoke	The users and groups that can invoke operations on any object controlled by the associated server.

The entries in the access control list can be user names or group names. The owner of an Implementation Repository entry is always allowed to launch it and invoke operations on its objects. A client normally needs both launch and invoke access to use an automatically-launched server. The following sections describe how to modify ACLs by adding groups and users or removing groups and users from ACLs.

Modifying Server Access

Use the `chmodit` command to modify the launch or invoke access control lists (ACLs). For example:

1. The following command allows the user `chris` to launch the server `AlliedBank`:

   ```
   chmodit AlliedBank l+chris
   ```

2. The following command grants the user `chris` rights to launch any server in the directory `banks/investmentBanks`:

   ```
   chmodit -a banks/investmentBanks l+chris
   ```

3. The following command revokes `joe`'s right to invoke all servers in the Implementation Repository directory `banks/commercialBanks`:

   ```
   chmodit -a banks/commercialBanks i-joe
   ```

4. There is also a pseudo-group named `all` that you can use to implicitly add all users to an ACL. The following command grants all users the right to invoke the server `banks/commercialBanks/AlliedBank`:

   ```
   chmodit banks/commercialBanks/AlliedBank i+all
   ```

On UNIX, the group membership of a user is determined using the user's primary group as well as the user's supplementary groups as specified in the /etc/group file.

Changing the Owners of Registered Servers

Only the owner of an Implementation Repository entry can use the chmodit command on that entry. The original owner is the one who uses the putit command to register the server. Use the chownit command to change ownership. For example, use the following command to change the ownership of server AlliedBank to user mcnamara:

```
chownit -s AlliedBank mcnamara
```

An Implementation Repository directory may have more than one owner. An ownership ACL is associated with each directory in the Implementation Repository, and this ACL can be modified to give certain users or groups ownership rights on a directory. Only a user on an ownership ACL has the right to modify the ACL. Some other examples of changing ownership include the following:

1. To add the group iona to the ownership ACL on the Implementation Repository directory banks/investmentBanks, use the following command:

   ```
   chownit -d banks/investmentBanks + iona
   ```

2. To remove mcnamara from the same ACL, do the following:

   ```
   chownit -d banks/investmentBanks - mcnamara
   ```

3. Orbix supports the pseudo-group all that, when added to an ACL, grants access to all callers. The following command grants all users the ownership rights on directory banks/commercialBanks:

   ```
   chownit -d banks/commercialBanks + all
   ```

   Spaces *are* significant in this grammar; for example:

CORRECT	`chownit -d banks/investmentBanks + iona`
INCORRECT	`chownit -dbanks/investmentBanks + iona`
INCORRECT	`chownit -d banks/investmentBanks +iona`

Determining the User and Group IDs of Running Servers

On Windows platforms, the user ID (*uid*) and group ID (*gid*) of a server process launched by the Orbix daemon are the same as those of the daemon itself.

On UNIX platforms, the effective uid and gid of a server process launched by the Orbix daemon are determined as follows:

1. If orbixd is not running as a super-user, such as root on UNIX, the uid and gid of every activated server process is that of orbixd itself.

   If orbixd is running as root, it attempts to activate a server with the uid and gid of the, possibly remote, principal attempting to activate the server.
2. If the principal is unknown (not a registered user) at the local machine on which orbixd is running, orbixd attempts to run the new server with uid and gid of a standard user called "orbixusr".
3. If there is no such standard user "orbixusr", orbixd attempts to run the new server with uid and gid of a user "nobody".

   If there is no such user "nobody", the activation fails and an exception is returned to the caller.

The daemon must be able to execute the server's executable file.

Note: If you are running orbixd as super-user, you should use the -o option to the Orbix daemon. This prevents a client running as a super-user on a remote machine from launching a server with super-user privileges on your machine. Refer to "Using the -o Option to the Orbix Daemon" on page 13 for more details.

Server Activation Modes

Orbix provides a number of different *modes* for launching servers. You specify the mode of a server when it is registered. Usually, clients are not concerned with the activation details of a server or aware of what server processes are launched. The following primary activation modes are supported by Orbix.

Shared Activation Mode

In this mode, all of the objects with the same server name on a given machine are managed by the *same* server process on that machine. This is the default activation mode.

If the process is already running when an application invocation arrives for one of its objects, Orbix routes the invocation to that process; otherwise Orbix launches a process.

Unshared Activation Mode

In this mode, individual objects of a server are registered with the Implementation Repository. As each object is invoked, an individual process is run for that particular object—one process is created per active registered object. You can register each object managed by a server with a different executable file, or any number of objects can share the same executable file.

Per-method call Activation Mode

In this mode, individual operation names are registered with the Implementation Repository. Inter-process calls can be made to these operations—and each invocation results in the launch of an individual process. A process is launched to handle each individual operation call, and the process is destroyed once the operation has completed. You can specify a different executable file for each operation, or any number of operations can share the same executable file.

The shared mode is most common. The unshared and per-method modes are rarely used. Refer to your server documentation to determine the correct activation modes to use.

Registering Unshared Servers

The `-unshared` option registers a server in the unshared activation mode. For example:

```
putit -unshared
   NationalTrust /financial/banks/banker
```

This command registers an unshared server called "`NationalTrust`" on the local host, with the specified executable file. Each activation for an object goes to a unique server process for that particular object. However, all users accessing a particular object share the same server process.

Using Markers to Specify Named Objects

Each Orbix object has a unique *object reference* that includes the following information:

- A name that is usually referred to as a *marker*.

  An object's interface name and its marker uniquely identify the object within a server. A server programmer can choose the marker names for objects or they can be assigned automatically by Orbix.
- A server name identifying the server in which the object is located.
- A host name identifying the host on which the server is located.

For example, the object reference for a bank account would include the bank account name (marker name), the name of the server that manages the account, and the name of the server's host.

Since objects can be named, shared and unshared server activation policies can specify individual object marker names. For example:

1. ```
   putit -marker College_Green
      NationalBank /financial/banks/banker
   ```

   This command registers a shared server called "`NationalBank`" on the local host, with the specified executable file. However, activation only
   ```

occurs for the object whose marker matches "College_Green". There is at most one server process resulting from this registration request; although you can make other -marker registrations for server NationalBank. All users share the same server process.

2. `putit -unshared -marker College_Green FirstNational /banks/FNbank_CG`

   `putit -unshared -marker St_Stephens_Green FirstNational /banks/FNbank_STG`

   The first command registers an unshared server called "FirstNational" on the local host with the specified executable files and the second adds an activation order (marker and launch command) for the "St_Stephens_Green" marker. However, activation only occurs for objects whose marker name is "College_Green" or "St_Stephens_Green" and each activation for a specific object goes to a *unique* server process for that particular object. All users of a specific object share the same server process.

Using Pattern Matching

You can use pattern matching in activation policies when seeking to identify which server process to communicate with. In particular, you can register a server activation policy for a subset of the server's objects. Since the number of objects named can get very large, pattern matching also means you do not have to specify a separate policy for every possible object. You specify this object subset by using wildcard characters in a marker pattern. The pattern matching is based on regular expressions, similar to UNIX regular expressions.

You can use pattern matching to specify a set of objects for shared or unshared servers. For example, some registrations can be used as a means of sharing work between server processes in this case, between two processes:

```
putit -marker '[0-4]*'
   NationalBank /work/bank/NBBank
putit -marker '[5-9]*'
   NationalBank /work/bank/NBBank
```

If these two commands are issued, server NationalBank can have up to two active processes; one launched for objects whose markers begin with the digits 0 through 4 and the other for markers beginning with digits 5 through 9.

Refer to the entry for the putit command in Appendix C, "Command Reference" for a complete list of recognized patterns with examples.

Use the rmit command with -marker option to modify a server entry. This allows you to remove a specific activation order for a server without removing the entire server entry. You can also use pattern matching with the rmit command's marker option.

Registering Per-Method Servers

A per-method server processes each operation call in a separate process.

1. The following command registers a per-method server called "NationalTrust" on the local host with the specified executable file. The activation occurs only if the operation makeWithdrawal() is called.

   ```
   putit -per-method -method makeWithdrawal
     NationalTrust /financial/NTbank
   ```

2. If the -method option is used, Orbix assumes that the server is a per-method server.

   ```
   putit -method makeDeposit
      NationalTrust /financial/NTbank
   ```

   You can specify patterns for methods so that operation names matching a particular pattern causes Orbix to use a particular server activation. The use of pattern matching allows a group of server processes to share a workload between them, whereby each server process is responsible for a range of methods. The pattern matching is based on regular expressions similar to UNIX regular expressions.

3. The following command registers a per-method server called "FirstTrust" on the local host with the specified executable file.

   ```
   putit -per-method FirstTrust
      -method 'make*' /financial/banker
   ```

   The activation is to occur only if an operation matching the pattern "make*" is being called, for example makeDeposit() or makeWithdrawal(). A separate process is activated for each method call.

Note: You can only use method pattern matching in the per-method activation mode, thus the -per-method option is redundant.

Use the `rmit` command with `-method` option to modify a per-method server entry. This allows you to remove a specific activation order for a server without removing the entire server entry. You can also use pattern matching with the `rmit` command's `-method` option.

Secondary Activation Modes

For each of the primary activation modes, a server can be launched in one of the secondary activation modes described as follows:

Multiple-client Activation Mode

In this mode, activations of the same server by different users share the same process, in accordance with the selected primary activation mode. This is the default secondary activation mode. No `putit` option is required to specify this mode when registering a server.

Per-client Activation Mode

In this mode, activations of the same server by different users cause a different process to be launched for each end-user.

Use the `putit -per-client` option to register a server in this secondary activation mode.

Per-client-process Activation Mode

In this mode, activations of the same server by different client processes cause a different process to be created for each such client process.

Use the `putit -per-client-pid` option to register a server in this secondary activation mode. For example, the following command registers a shared, per-client-process server:

```
putit -per-client-pid
   FirstTrust /work/bank/banker
```

Activation occurs when any of the objects managed by the `FirstTrust` server are used; there is a separate server process for each different client process.

Managing Server Port Selection

When the Orbix daemon activates a server, the server is activated by the Orbix daemon, it is assigned a port so that clients can communicate with it. There are two ways to control the port numbers assigned to a server:

- Registering the server with a specified port number.
- Using configuration variables to control port numbers.

This section describes each of these approaches.

Registering Servers with Specified Ports

When registering a server, you can specify the port on which the server should listen using the `-port` option to `putit`. For example, to specify that shared server `FirstTrust` should communicate on port 1597, do the following:

```
putit -port 1597 FirstTrust /work/bank/banker
```

By default, all Orbix applications communicate over the CORBA standard Internet Inter-ORB Protocol (IIOP). The `-port` option is very important for such applications.

If an Orbix server that communicates over IIOP publishes an object reference, for example using the CORBA Naming Service, this reference is valid while the server continues to run. However, if the server exits and then recreates the same object, the published object reference is not valid unless the server always runs on the same port. If your servers require this functionality, you should register them using the `-port` option.

Controlling Port Allocation with Configuration Variables

You can control the range of server port numbers chosen by the Orbix daemon by using the configuration entries IT_DAEMON_SERVER_BASE and IT_DAEMON_SERVER_RANGE. The IT_DAEMON_SERVER_BASE must be set and the recommended value is 1590. You do not have to set IT_DAEMON_SERVER_RANGE which has a default value of 50.

When the Orbix daemon starts a server, the first server port assigned is IT_DAEMON_SERVER_BASE plus 1, and the last assigned is IT_DAEMON_SERVER_BASE plus IT_DAEMON_SERVER_RANGE. For example, using the default values the server ports range from 1591 to 1640.

Once the end of the range is reached, orbixd recycles the range in an attempt to find a free port. If no free port is found, an IMP_LIMIT system exception is raised to the client application attempting an invocation to the server.

You can set IT_DAEMON_SERVER_BASE and IT_DAEMON_SERVER_RANGE values by using their entries in the common.cfg configuration file, or by setting the corresponding environment variables. Values you set must be greater that 1024 and you should make sure that they do not conflict with other services. Make sure the range you choose is greater than the maximum number of servers you expect to run on the host.

Registering SSL-Enabled Servers

To register servers that are SSL-enabled use the putit utility with the additional SSL syntax highlighted below.

This is the full putit command syntax:

```
putit [-v] [-h <host>] [-per-client | -per-client-pid]
[ [-shared | -unshared] [-marker <marker>] ]
[ -j | -java [-classpath <classpath> | -addpath <path> ] ]
[ -oc <ORBclass> -os <ORBSingletonClass>] [ -jdk2]
| [-per-method [-method <method>] ]
[-port <iiop portnumber>]
[ -n <number of servers> ] [ -l ]
[ -ssl_secure | -ssl_semi_secure  [-ssl_client_auth] [-
ssl_support_null_enc | -ssl_support_null_enc_only] [-
ssl_support_null_auth | -ssl_support_null_auth_only] ]
<serverName> [ <commandLine> | -persistent ]
```

The ssl parameters are described in Table 3.1. To use them, you must specify either `-ssl_secure` or `-ssl_semi_secure` first.

putit Flag	Description
`-ssl_client_auth`	Indicates that the server authenticates clients.
`-ssl_support_null_enc`	This indicates that the NULL encryption SSL ciphersuites (which do not support confidentiality) are supported by the server.
`-ssl_support_null_enc_only`	This indicates that only the server supports the NULL encryption SSL ciphersuites.
`-ssl_secure`	This is the minimal flag needed to indicate that the server is SSL-enabled. If this flag or `-ssl_semi_secure` are not supplied then the server is insecure, and no SSL related data should be written to the IR. One of these two flags must be supplied before any other SSL flag is acceptable. An error should be presented to the user if they are not.
`-ssl_semi_secure`	This indicates a `SEMI_SECURE` server policy. If this flag or `-ssl_secure` are not supplied to `putit` then the policy is `INSECURE` and no SSL-related datashould be written to the IR. One of these two flags must be supplied before any other SSL flag is acceptable. An error should be presented to the user if they are not.
`-ssl_support_null_auth`	This flag indicates that the server supports null authentication. OrbixSSL servers do not currently support this; nevertheless you can code the flag now to save time in the future.
`-ssl_support_null_auth_only`	This flag indicates that the server supports null authentication. OrbixSSL servers do not currently support this; nevertheless you can code the flag now to save time in the future.

Table 3.1: *putit SSL Parameters*

Using the putit SSL Parameters

There are four groups of SSL parameters. If you want to use them, you must use one from Group 1, followed by one or none from each of the other three groups:

Group 1

```
-ssl_secure
-ssl_semi_secure
```

Group 2

```
-ssl_support_null_enc
-ssl_support_null_enc_only
<NOTHING>
```

Group 3

```
-ssl_support_null_auth
-ssl_support_null_auth_only
<NOTHING>
```

Group 4

```
-ssl_client_auth
<NOTHING>
```

As OrbixSSL supports per server process security policy settings, those settings specified by putit apply to all objects created by the server.

The most common use cases are:

```
Putit –ssl_secure demo/grid grid.exe
Putit –ssl_secure –ssl_client_auth demo/grid grid.exe
Putit –ssl_semi_secure demo/grid grid.exe
```

The following might be less common:

```
Putit –ssl_semi_secure –ssl_client_auth demo/grid grid.exe
```

4

Managing the Interface Repository

The Interface Repository is the component of Orbix that stores information about IDL definitions and allows clients to retrieve this information at runtime. This chapter describes how to manage the contents of the Interface Repository.

The Interface Repository maintains full information about the IDL definitions implemented in your system. Given an object reference, a client can determine at runtime the object's type and all information about that type by using the Interface Repository. Clients can also browse contents of the Interface Repository.

To allow a client to obtain information about a set of IDL definitions, you must add those definitions to the Interface Repository. Orbix supports commands that allow you to add IDL definitions to the repository, read the contents of the repository, and remove definitions from it. Each of these commands accesses the Interface Repository through the Interface Repository server.

Configuring the Interface Repository

The Interface Repository has its own directory, which is specified by the IT_INT_REP_PATH entry in the common.cfg configuration file or as an environment variable. IT_INT_REP_PATH is a required variable.

You must configure the Interface Repository before the IDL compiler or applications can use it. To configure the Interface Repository, do the following:

1. Specify a value for the IT_INT_REP_PATH entry in the common.cfg file or as an environment variable. For example:

   ```
   IT_INT_REP_PATH /orbix/IntRep
   ```

2. Create the corresponding directory if it does not already exist.

   ```
   mkdir /orbix/IntRep
   ```

3. If the Orbix daemon is running, stop it and then restart it so that it recognizes the new configuration variable:

   ```
   orbixd
   ```

Registering the Interface Repository Server

The Interface Repository is accessed through an Orbix server. The interfaces to the Interface Repository objects are defined in IDL and you must register the Interface Repository server using the putit command. For example:

```
putit IFR /orbix/ifr/bin/IFR
```

Orbix expects that the server is registered with the name IFR as a shared server. The Interface Repository's executable file is in the bin directory with the name IFR.

The Interface Repository server can be launched by the Orbix daemon, or it can be launched manually. For example, the server executable file can be explicitly run as a background process:

```
/orbix/ifr/bin/IFR
```

This has the advantage that the Interface Repository can initialize itself before any other processes need to use it.

The server executable file can take the following options:

`-h`	Print a summary of switches.
`-L`	Immediately load data from the Interface Repository data directory. The default is not to do this, but instead to load each file on demand at runtime as it is required.
`-t` *time*	Specify the timeout in seconds for the Interface Repository server. The default timeout is infinite.
`-v`	Print version information about the Interface Repository.

Adding IDL Definitions

The Orbix utility `putidl` allows you to enter all the definitions in a single IDL source file into the Interface Repository. This utility provides a simple and safe way to add IDL definitions to the repository.

For example, the following command adds the definitions in the file `banksimple.idl` to the Interface Repository:

```
putidl banksimple.idl
```

The `putidl` utility parses the definitions in the file `banksimple.idl` and integrates the definitions into the repository. If the file `banksimple.idl` uses definitions already registered in the repository, `putidl` checks that the definitions are used consistently before updating the repository contents.

If you modify the file `banksimple.idl`, you can update the contents of the Interface Repository by repeating the `putidl` command.

Although `putidl` takes an IDL file as an argument, the Interface Repository does not store information about the file itself. The Interface Repository has no knowledge of the file associated with specific IDL definitions. This means that you cannot remove definitions based on the file in which they were declared. For this reason, it is important that you use modules in your IDL definitions to group definitions in logical units.

Reading the Interface Repository Contents

The `readifr` utility allows you to read a specified IDL definition from the Interface Repository. For example, to view the definition of interface `Bank` defined in module `Finance`, do the following:

```
readifr Finance::Bank
```

This utility prints the IDL definition to the standard output.

If you use `readifr` to view an IDL interface definition, you can instruct it to also display all derived interfaces. To do this, specify the `-d` option, for example:

```
readifr -d Finance::Bank
```

Removing IDL Definitions

The `rmidl` utility allows you to remove an IDL definition from the Interface Repository. This utility takes a fully scoped name for an IDL definition as an argument.

For example, to remove information about the IDL operation `create_Account()` defined on interface `Bank` in module `Finance`, do the following:

```
rmidl Finance::Bank::create_Account()
```

The `rmidl` command removes definitions recursively. For example, to remove the module `Finance` and all definitions within this module, do the following:

```
rmidl Finance
```

Part II

Orbix C++ GUI Tools

5

The Orbix Configuration Explorer

Components of an Orbix system are configured using a number of configuration files, as described in Chapter 2, "Getting Started". The Orbix Configuration Explorer allows you to configure Orbix components without modifying the configuration files directly.

The Orbix configuration files configure the main components of Orbix, and each Orbix installation has at least one copy of each file. The Orbix Configuration Explorer allows you to modify any Orbix configuration file on your system.

The configuration files include settings that affect the configuration of Orbix and settings that affect the configuration of other Orbix products, for example OrbixNames. The Orbix Configuration Explorer allows you to modify all these settings, and to create additional settings. This tool integrates all Orbix configuration in a single user interface.

By default, the Configuration Explorer allows you to configure settings that are:

- Common to multiple IONA products.
- Orbix-specific
- OrbixNames-specific

Starting the Configuration Explorer

You can run the Configuration Explorer from the Windows **Start** menu, or by entering `configurationexplorer` at the command line. The Configuration Explorer appears as shown in Figure 5.1.

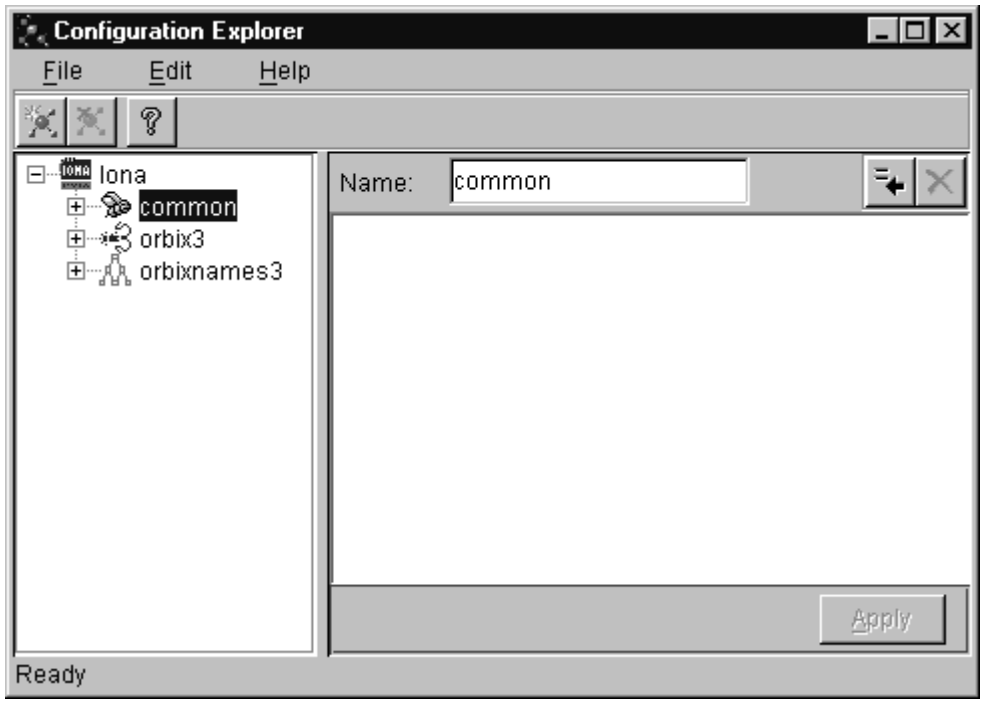

Figure 5.1: *Orbix Configuration Explorer*

This tool includes the following elements:

- *A menu bar*.
- *A toolbar*.
- *A navigation tree*.

  The navigation tree displays icons that represent each configuration file and configuration scope.

- *A text box*.

  The **Name** textbox displays the name of the current configuration file or scope.

- A *text pane*.

  The *text pane* control contains a **Name** column and a **Value** column as shown in Figure 5.2 on page 48. Each row corresponds to individual configuration file entries. The text pane enables you to view and modify these entries.

At startup, the Orbix Configuration Explorer opens the `iona.cfg` root configuration file. By default, this file is located in the `config` directory of your Orbix installation. The Configuration Explorer navigation tree displays icons that represent the configuration files included in `iona.cfg` as shown in Figure 5.1 on page 46.

Configuring Common Settings

To configure settings that are common to multiple IONA products, select the **Common** icon in the navigation tree. This icon represents the `Common` configuration scope in the file `common.cfg`. The `Common` variables stored in the default `common.cfg` configuration file then appear in the text pane, as shown in Figure 5.2 on page 48.

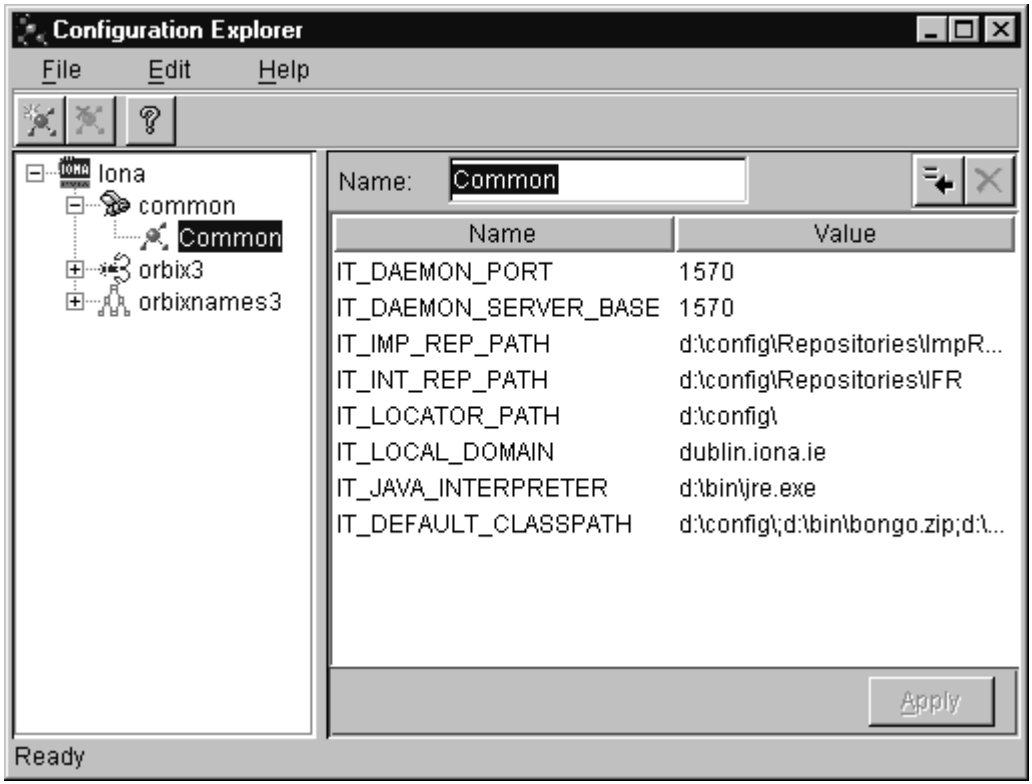

Figure 5.2: *Common Configuration Settings*

The default **Common** configuration settings are as follows:

IT_DAEMON_PORT	The TCP port number on which the Orbix daemon receives communications from clients.
IT_DAEMON_ SERVER_ BASE	The first TCP port number assigned by the daemon to a server. Each server listens on a single port number for client connection attempts.
IT_IMP_REP_PATH	The full path name of the Orbix Implementation Repository directory.
IT_INT_REP_PATH	The full path name of the Orbix Interface Repository directory.
IT_LOCAL_DOMAIN	The Internet domain name for your local network.
IT_JAVA_INTERPRETER	The full path name to the Java Runtime Environment binary executable. This installs with Orbix by default.
IT_DEFAULT_CLASSPATH	The default classpath used when Java servers are automatically launched by the daemon.

To update any of these settings, do the following:

1. Select the variable in the text pane.
2. Double-click on this variable in the **Value** column
3. Enter your new setting.
4. Select the **Apply** button to save your setting to the appropriate configuration file.

You cannot undo settings that you have saved to file.

Configuring Orbix-Specific Settings

To configure settings that apply to Orbix only, select the **Orbix** icon in the navigation tree. This icon represents the `Orbix` configuration scope in the file `orbix3.cfg`. The `Orbix` variables stored in the default `orbix3.cfg` configuration file then appear in the text pane, as shown in Figure 5.3.

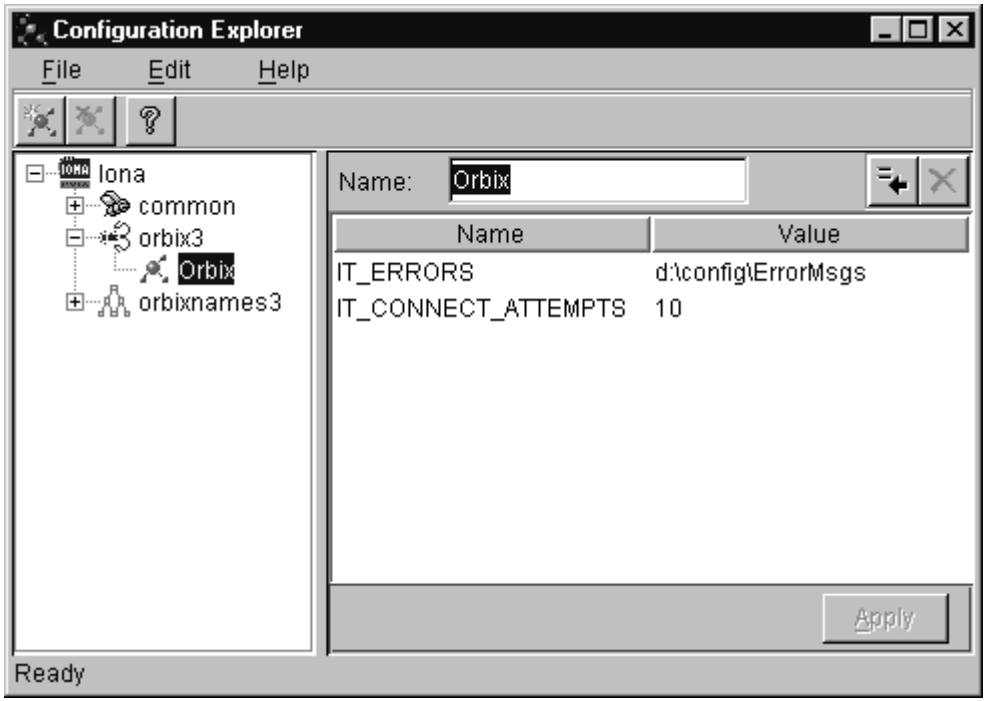

Figure 5.3: *Configuring Orbix-Specific Settings*

By default, the **Orbix** configuration settings include the following:

`IT_ERRORS`	The full path name of the Orbix error messages file.
`IT_CONNECT_ATTEMPTS`	If a client fails to connect to a server, Orbix retries the connection attempt every two seconds until the client succeeds. This value specifies the maximum number of retry attempts.

To update these settings, do the following:

1. Select the variable in the text pane.
2. Double-click on this variable in the **Value** column to enter your setting.
3. Select the **Apply** button to save your setting to the appropriate configuration file.

You can also modify OrbixNames-specific configuration variables by following these steps. Refer to the *OrbixNames Programmer's and Administrator's Guide* for details of configuration variables that are specific OrbixNames.

Customizing Your Configuration

By default, the Orbix Configuration Explorer displays the configuration variables contained in the default configuration files. You can use the Orbix Configuration Explorer to customize your configuration by:

- Creating configuration variables.
- Creating configuration scopes.
- Creating configuration files.

Creating Configuration Variables

By default, the Configuration Explorer displays a default subset of the available configuration variables. You can also create additional configuration variables, as shown in Figure 5.4.

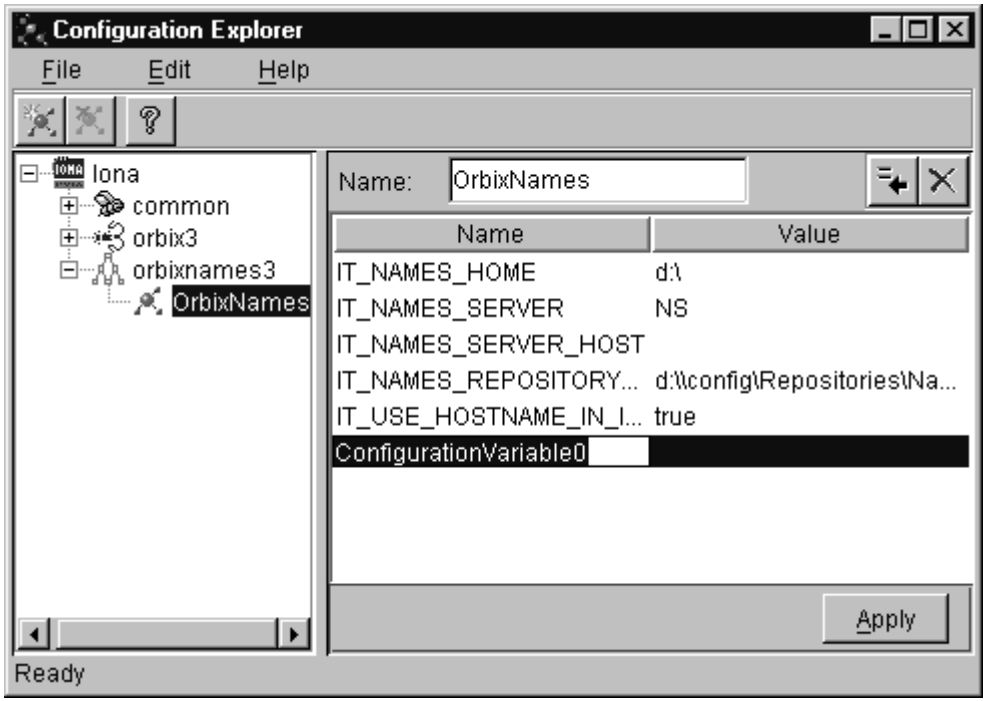

Figure 5.4: *Creating Configuration Variables*

To create a configuration variable, perform the following steps:

1. Select the **Create Configuration Variable** button, shown in Figure 5.5 on page 53.
2. Double-click the new entry in the **Name** column of the text pane.
3. Enter a name for your configuration setting.

4. Double-click the entry in the **Value** column.
5. Enter a value for your configuration variable
6. Select the **Apply** button to save your setting to the appropriate configuration file.

Figure 5.5: *Creating and Deleting Configuration Variables*

Valid Names for Configuration Variables and Scopes

You can use the following characters when naming configuration variables and scopes:

```
["_", "-"], ["a"-"z","A"-"Z"], ["0"-"9"]
```

Note: You cannot uses spaces when naming configuration variables and configuration scopes.

There are no restrictions on the valid characters for configuration values.

Deleting Configuration Variables

You cannot delete the configuration variables included in the default configuration files. You can only change the values of these variables. However, you can delete any additional variables that you may have created.

To delete a configuration variable, do the following:

1. Select the setting to be deleted from the text pane.
2. Select the **Delete Configuration Variable** button, shown in Figure 5.5.
3. Select the **Apply** button to save your setting to the appropriate configuration file.

Refer to Appendix A, “Configuration Variables” on page 79 for a complete list of both common and Orbix-specific configuration variables.

Creating Configuration Scopes

The Configuration Explorer displays the configuration variables contained in the default configuration files. You can customize your configuration by creating additional configuration *scopes*. Configuration scopes are containers for configuration variables. Refer to "The Orbix Configuration Files" on page 6 for more details.

In the navigation tree, user-defined configuration scopes are displayed as branching from default configuration scope icons, as shown in Figure 5.6 on page 55.

To create a user-defined configuration scope, do the following:

1. Select **Edit→Create Scope** from the menu bar. Alternatively, you can use the **Create Scope** toolbar.
2. In the **Name** text box, enter the name of your configuration scope.
3. Select the **Apply** button to save your setting to the appropriate configuration file.

You can then create new configuration variables within your configuration scope, as described in "Creating Configuration Variables" on page 52.

Deleting Configuration Scopes

You cannot delete the default configuration scopes included in the default configuration files. However, you can delete any additional scopes that you may have created.

To delete a configuration scope, do the following:

1. From the navigation tree, select the scope to be deleted.
2. Select the **Edit→Delete Scope** menu option. Alternatively, you can use the **Delete Scope** button on the toolbar**.**

Select the **Apply** button to save your setting to the appropriate configuration file.

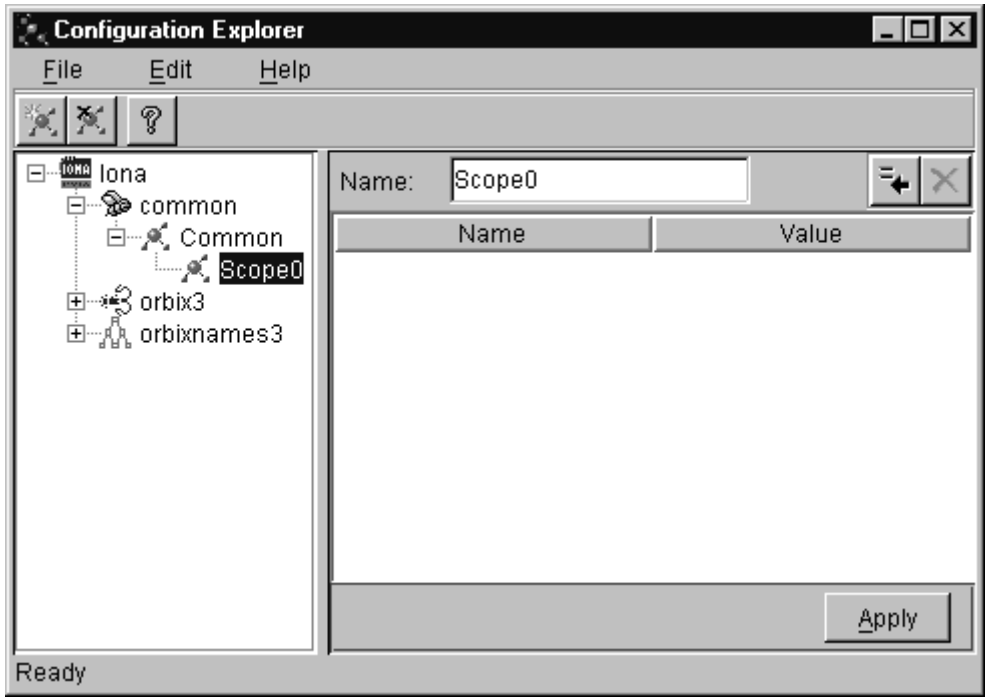

Figure 5.6: *Creating Configuration Scopes*

Creating Configuration Files

You can extend the Configuration Explorer to display custom configuration files. To create a configuration file you should edit your `iona.cfg` file to include the additional configuration file. An icon associated with this configuration file then appears in the Configuration Explorer navigation tree.

You can then create new configuration scopes and variables within your new configuration file as usual, as described in "Creating Configuration Variables" on page 52 and "Creating Configuration Scopes" on page 54.

6

The Orbix Server Manager

The Implementation Repository is the component of Orbix that maintains registration information about servers and controls their activation. The Orbix Server Manager allows you to manage the Implementation Repository.

The Implementation Repository maintains a mapping from a server name to the executable code that implements that server. In an Orbix system, the Orbix daemon on each host has an associated Implementation Repository. The Implementation Repository allows the daemon to launch server processes in response to operation calls from Orbix clients.

The Orbix Server Manager allows you to do the following:

- Browse an Implementation Repository.
- Register new servers.
- Modify existing server registration details.

The *Orbix Programmer's Guide C++ Edition* describes the Implementation Repository in detail. This chapter assumes that you are familiar with this description.

Starting the Server Manager

You can run the Server Manager from the Windows **Start** menu or by entering `srvmgr` at the command line. The main Server Manager window appears as shown in Figure 6.1.

Figure 6.1: *Server Manager Main Window*

The **Server Manager** window includes the following elements:

- *A menu bar*.
- *A toolbar*.
- *A navigation tree*.

  This tree displays a graphical representation of the contents of an Implementation Repository.
- *A server information pane*.

  If you select an item in the navigation tree, the pane to the right of the tree displays detailed information about that item. Information about servers is displayed in a tabbed folder.
- *A status bar*.

You can use the toolbar icons in place of the menu options described in this chapter.

Connecting to an Implementation Repository

To connect to an Implementation Repository, do the following:

1. Select **Host→Connect**. The **Connect to Host** dialog box appears, as shown in Figure 6.2.

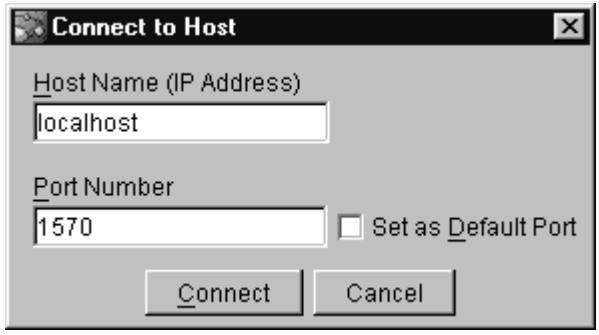

Figure 6.2: *Connect to Host Dialog Box*

2. In the **Host Name** text box, enter the name or IP address of the host on which the required Orbix daemon runs. The default is the local host.
3. In the **Port Number** text box, enter the TCP/IP port number on which the Orbix daemon runs. To make a port number the default, select the **Set as Default Port** check box. The default port number is initially set to 1570.
4. Select the **Connect** button. The main Server Manager window displays the contents of the Implementation Repository. For example, Figure 6.3 shows an Implementation Repository on the local host .

You can disconnect from an Implementation Repository at any time. To disconnect, in the main window, select the required host and then select **Host→Disconnect**.

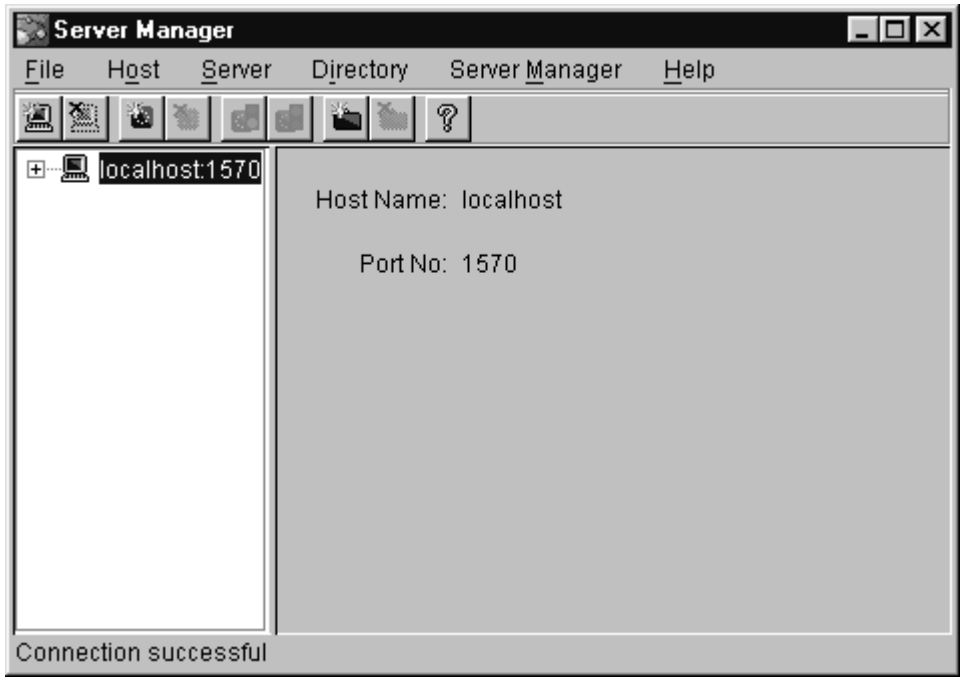

Figure 6.3: *Connection to an Implementation Repository*

Creating a New Directory

The Implementation Repository supports the concept of directories. This allows you to structure server names hierarchically, and organize the contents of an Implementation Repository.

To create an Implementation Repository directory, do the following:

1. Select the Implementation Repository on the appropriate host.
2. Select **Directory→New**. The **Directory Name** text box appears in the right hand pane of the main window, as shown in Figure 6.4 on page 62.
3. In the **Directory Name** text box, enter the name of the new directory.
4. Select the **Apply** button. The main Server Manager window now includes the new directory when displaying the contents of the Implementation Repository. For example, if you create a Bank directory, this directory is displayed in the directory tree after the **Apply** button is selected. This is shown in Figure 6.4 on page 62.

To delete a directory, select the directory in the main **Server Manager** window and then select **Directory→Delete**.

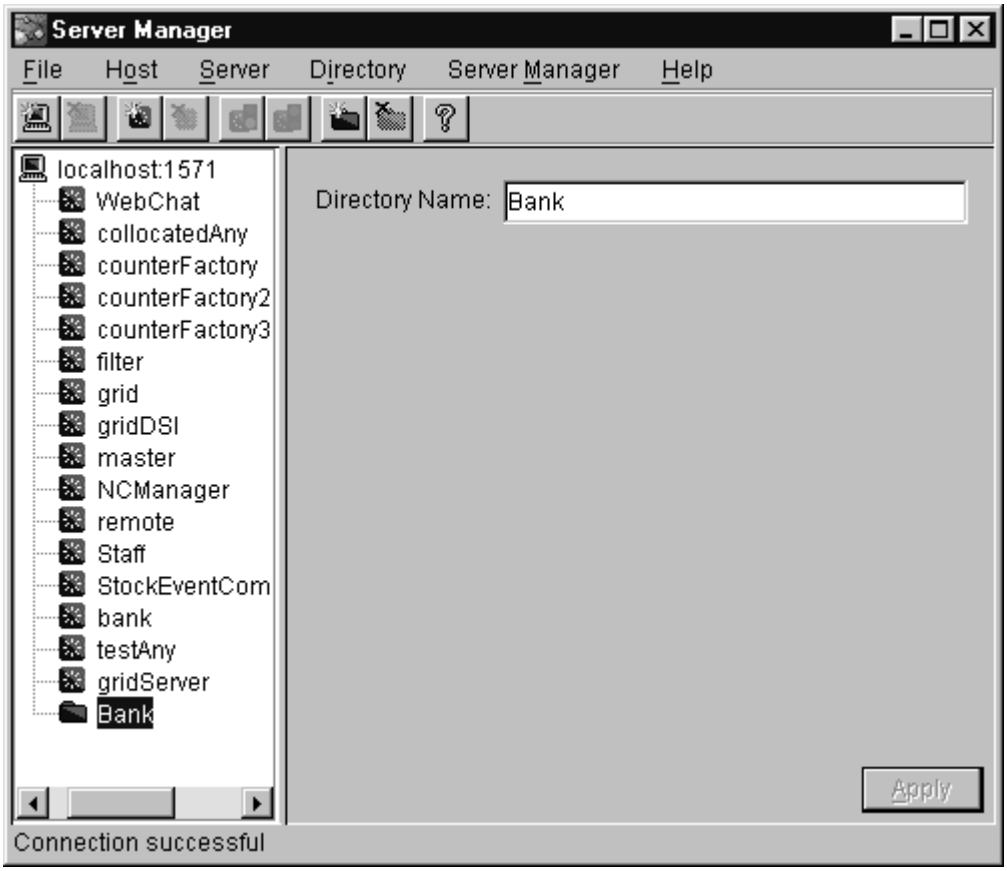

Figure 6.4: *Creating a New Directory*

Registering a Server

To register a server, do the following:

1. Select the Implementation Repository directory in which you wish to register the server. For example, to register a server in directory Bank, select the icon for this directory in the main window.
2. Select **Server→New**. A tabbed folder appears in the right pane of the main window as shown in Figure 6.5 on page 64. This folder is used to record a server's registration details.
3. Enter the server name in the **Server Name** text box on the **General** tab.
4. If the server is an Java server, select the **OrbixWeb Server** check box.
5. By default, only the user who registers the server can run clients that launch the server or invoke operations on server objects.

   To provide server access rights to other users, select the **Rights** tab. The **Rights** tab is described in "Providing Server Access Rights to Users" on page 65.
6. The default server primary activation mode is shared. The default secondary activation mode is normal.

   To modify the server activation details, select the **Activation** tab. The **Activation** tab is described in "Specifying Server Activation Details" on page 67.

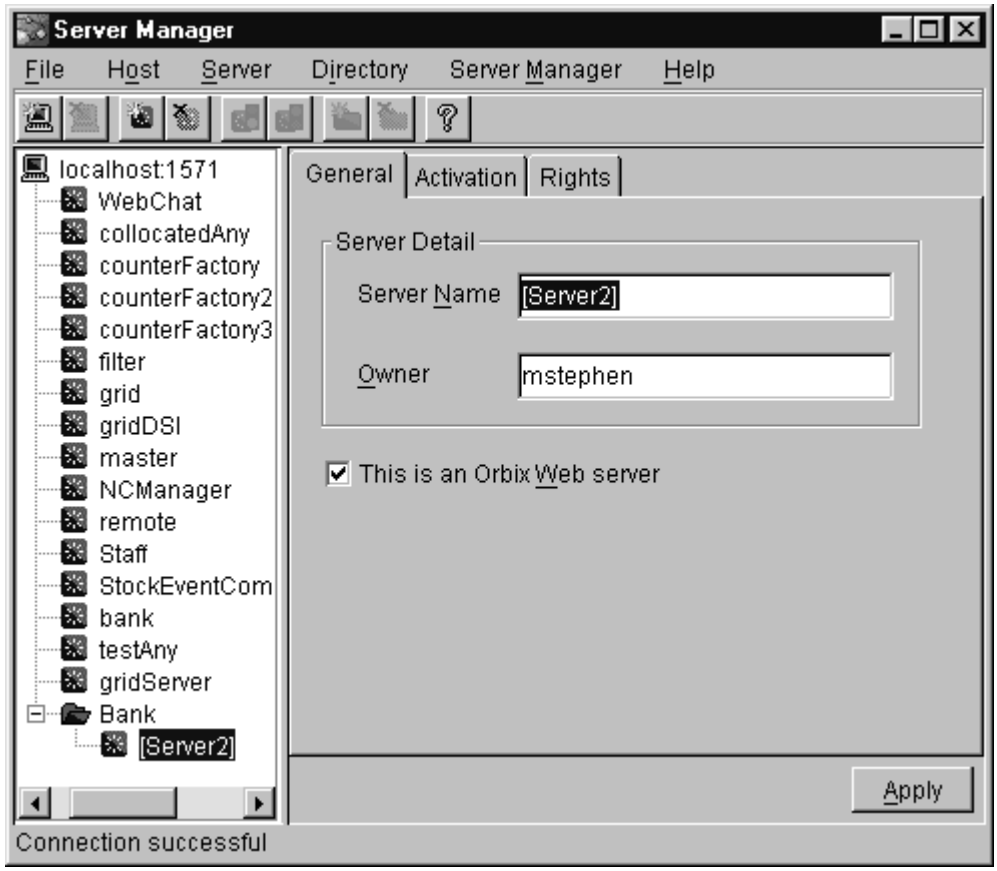

Figure 6.5: *Registering a New Server*

Providing Server Access Rights to Users

During server registration, you can provide server access rights to other users by selecting the **Rights** tab in the main window. The **Rights** tab appears as shown in Figure 6.6 on page 66.

Orbix offers two types of access rights:

- Launch rights
- Invoke rights

Launch rights allow clients owned by a specified user to cause the Orbix daemon to activate the server.

Invoke rights allow clients owned by a specified user to invoke operations on objects in the server.

To provide launch or invoke rights to a user, do the following:

1. In the appropriate area, enter the user identifier in the text box. To grant these rights to all users, enter the user name `all`.
2. Select **Add**.

To remove launch or invoke rights for a user, do the following:

1. In the appropriate user list, select the required user identifier.
2. Select **Remove**.

When you have added or removed the required users from the access rights lists, select **Apply** to commit the changes.

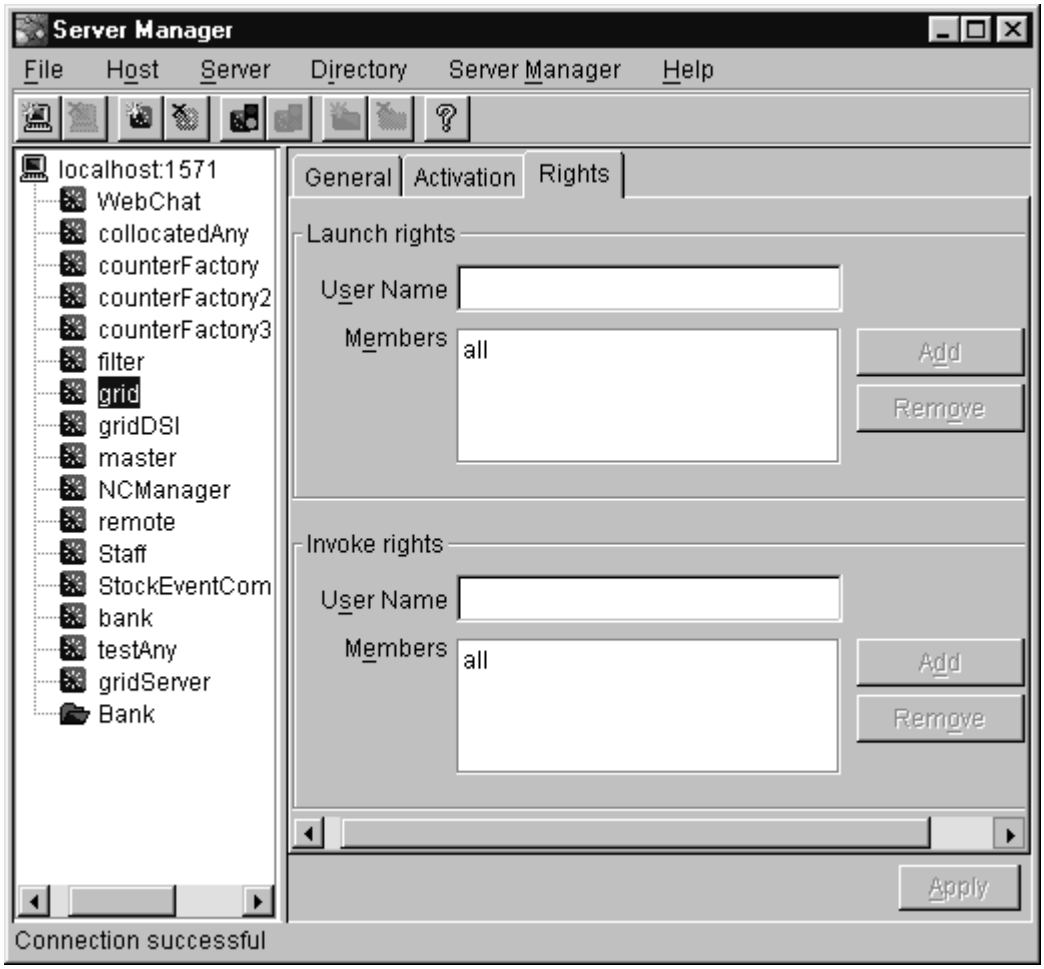

Figure 6.6: *Providing Server Access Rights*

Specifying Server Activation Details

During server registration, you can specify the server activation details by selecting the **Activation** tab in the Server Manager main window. The **Activation** tab appears as shown in Figure 6.7.

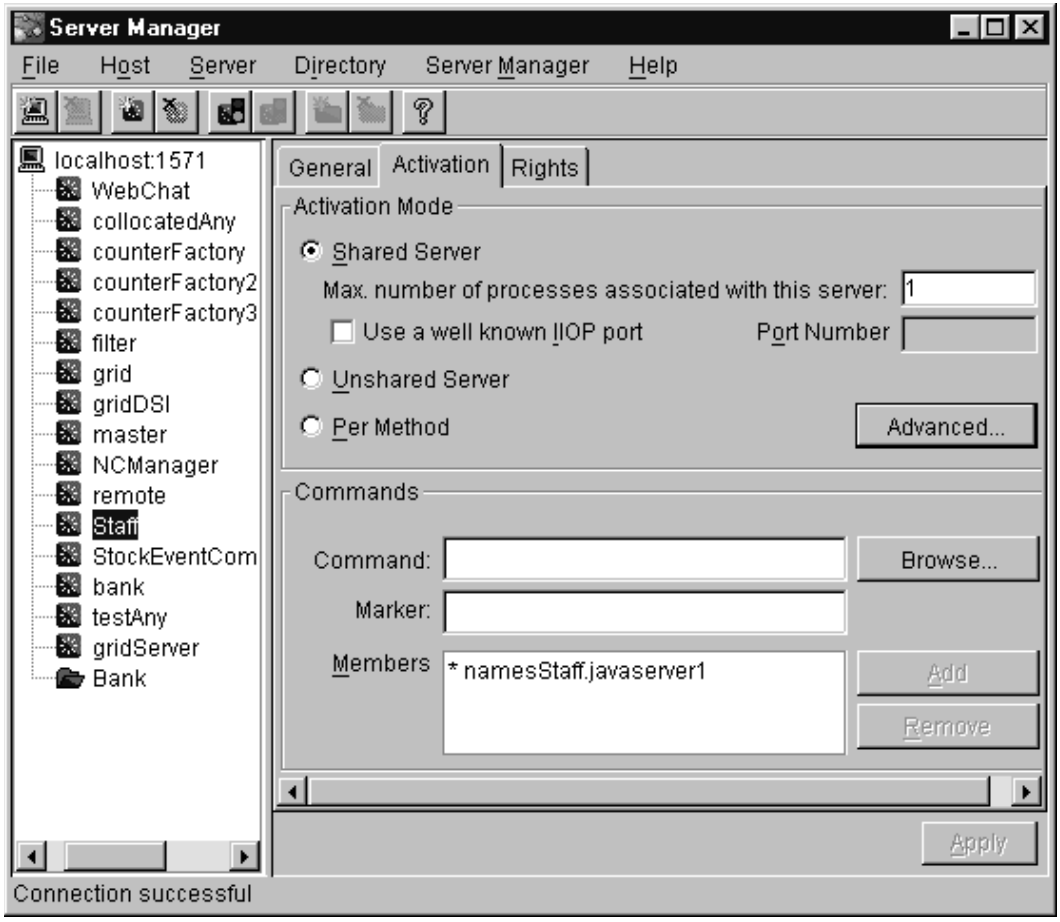

Figure 6.7: *Specifying Server Activation Details*

Activation Modes

To specify a server's *primary activation mode*, use the radio buttons in the **Activation Mode** section of the **Activation** tab. The default server primary activation mode is shared.

To specify a server's *secondary activation mode* select the **Advanced** button in the **Activation Mode** section. This launches the **Secondary Activation Modes** dialog box, as shown in Figure 6.8. The default secondary activation mode is normal.

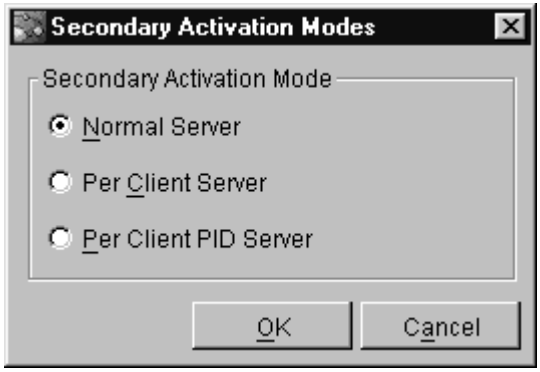

Figure 6.8: *Secondary Activation Modes*

A server registered in shared activation mode can have an associated maximum number of processes. The Orbix daemon launches up to the specified number of processes for that server.

Each new client connection results in a new server process until the maximum number of processes is available. Subsequent client connections are routed to existing server processes using a round-robin algorithm. This provides a primitive form of load-balancing for shared servers.

To specify the number of processes associated with a shared server, enter a positive integer value in the **Max. number of processes associated with this server** text box.

You can associate a well-known TCP/IP port number with servers that communicate using the CORBA-defined Internet Inter-ORB Protocol (IIOP). To specify a well-known IIOP port for a server, select the **Use a Well known IIOP Port** check box and enter a value in the **Port Number** text box.

When you have specified the server activation details, select **OK** to confirm these details.

Launch Commands

The **Commands** section on the **Activation** tab allows you to modify the launch commands associated with a server. Launch commands depend on the server activation mode, as follows:

Shared Activation Mode

If the server activation mode is *shared*:

1. Enter the server launch command in the **Command** text box.
2. Enter a * character in the **Marker** text box.
3. Select **Add**.

UnShared Activation Mode

If the server activation mode is *unshared*:

1. Enter a marker pattern in the **Marker** text box.
2. Enter the launch command for this marker pattern in the **Command** text box.
3. Select **Add**.

Repeat this process for each marker pattern you wish to register.

Note: A server registered in the Implementation Repository must have at least one launch command.

Per-method Activation Mode

If the server activation mode is *per-method*:

1. Enter a method name in the **Marker** text box.
2. Enter the launch command for this method in the **Command** text box.
3. Select **Add**.

Repeat this process for each method you wish to register.

Modifying Server Registration Details

When you register a server, the Orbix daemon creates a server registration record in the Implementation Repository. This record stores detailed information about the server.

To modify a server registration record, do the following:

1. Select the server you wish to modify.

   The Server Manager displays the tabbed folder containing all the registration details for the selected server.
2. Select the required tab from the following:
   - **General**
   - **Activation**
   - **Rights**
3. Enter the value in the appropriate section of the tab, as described in "Registering a Server" on page 63.
4. Select the **Apply** button.

Launching a Persistent Server

Orbix allows you to launch shared servers manually. A manually-launched server is known as a *persistent server*.

To launch a persistent server process, do the following:

1. Select the server you wish to launch. The server must be registered in shared mode.
2. Select **Server→Launch**. If successful, this starts the server executable file specified in the server launch command. The icon for the selected server displays a green traffic light while the server process runs, as shown in Figure 6.9.

To kill a shared server process, select **Server→Kill**.

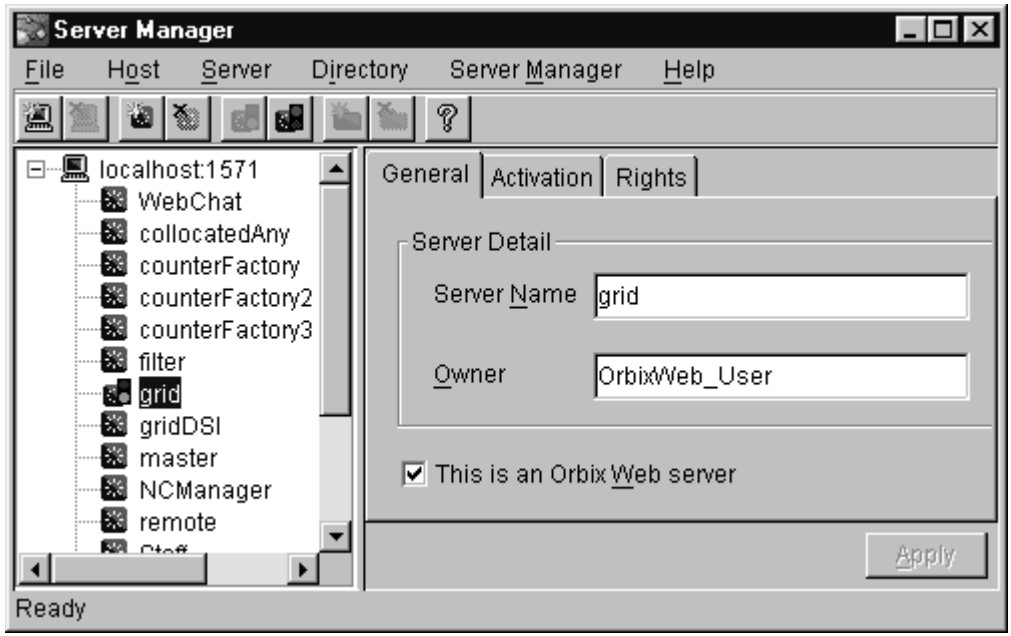

Figure 6.9: *Launching a Persistent Server*

Configuring the Server Manager

To configure the Server Manager, do the following:

1. In the main Server Manager window, select **Server Manager→Options**. The **Options** dialog box appears, as shown in Figure 6.10.

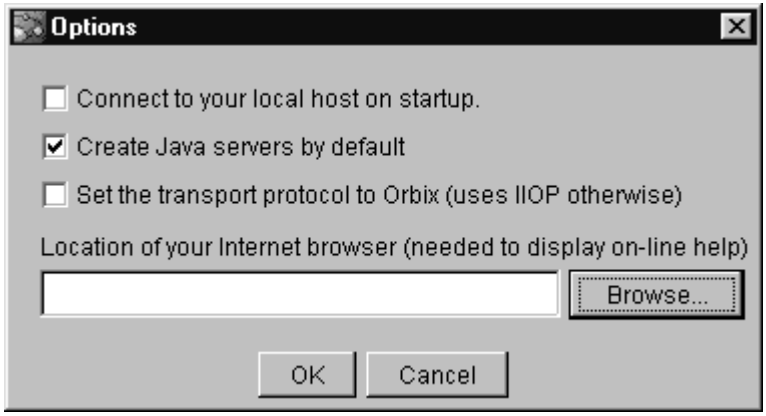

Figure 6.10: *The Options Dialog Box*

2. By default, the Server Manager does not connect to an Orbix daemon at startup. To specify that the Server Manager should connect to the Orbix daemon at the local host, select the **Connect to your local host on startup** check box.
3. The Server Manager allows you to register C++ or Java servers. By default, the Server Manager assumes that all servers are C++ servers.

   To change this default, select **Create Java Servers by default.**

4. You can also select the transport protocol used. The default protocol is IIOP (Internet Inter-Orb Protocol). To change this default, select the check box labelled **Set the transport protocol to use Orbix.**
5. To enable on-line help, enter the **Location of your internet browser** in the text box provided.
6. Select **OK** to commit the new configuration.

Note: The main Server Manager window refreshes itself automatically, reflecting updates as they occur. This means that the **Refresh Time** option, used in earlier versions of the Orbix Server Manager, is no longer necessary.

Part III

Appendices

Appendix A
Configuration Variables

There are two forms of Orbix configuration variables: those that are common to multiple IONA products and variables that are specific to Orbix only.

Common Configuration Variables

You can set the following variables as environment variables using the Configuration Explorer GUI tool, or by editing the `common.cfg` configuration file. Alternatively, you can modify some of these configuration variables at runtime using the `SetConfigValue()` series of APIs, and you must preface the configuration variable with "Common.", e.g. `Common.IT_DAEMON_PORT`.

Variable	Description
`IT_DAEMON_PORT`	TCP port number for the Orbix daemon.
`IT_DAEMON_SERVER_BASE`	The starting TCP port number for servers launched by the Orbix daemon.
`IT_DAEMON_SERVER_RANGE`	The number set in this variable is used together with that set in `IT_DAEMON_SERVER_BASE` to determine the range of port numbers available for Orbix servers.
`IT_DEFAULT_CLASSPATH`	This is a colon-separated list of full path names specifying the location of class files for the Java Interpreter. Default value points to the `CLASSPATH` environment variable.
`IT_IMP_REP_PATH *`	The full path name of the Implementation Repository directory.

Variable	Description
IT_INT_REP_PATH *	The full path name of the Interface Repository directory.
IT_JAVA_INTERPRETER	The number set in this variable is used together with that set in IT_DAEMON_SERVER_BASE to determine the range of port numbers available for Orbix servers.
IT_LOCAL_DOMAIN	The name of the local internet domain, for example, iona.com.

* These configuration variables can be set using the SetConfigValue() API. See "CORBA::ORB::SetConfigValue()" on page 221 of the *Orbix C++ Programmer's Reference*.

Orbix-Specific Configuration Variables

You can set the following variables using the Configuration Explorer GUI tool, or by editing the common.cfg configuration file. Alternatively, you can modify the configuration variables at runtime using the SetConfigValue() series of APIs, and you must preface the configuration variable with "Orbix.", e.g. Orbix.IT_CONNECT_ATTEMPTS.

Variable	Description
IT_ACT_POLICY *	The activation policy (or mode) to be used for launching servers.
IT_COLLOCATED	Set to TRUE if a client is using a collocated server object.
IT_CONNECT_ATTEMPTS *	The maximum number of retries Orbix makes to connect a client to a server. The value specified is only used if the API function CORBA::ORB::maxConnectRetries(CORBA::ULong) is called with a value of zero for the parameter.

Variable	Description
IT_DAEMON_PROTOCOL	Defines the protocol that Orbix uses to talk to the daemon. Valid values are POOP (Orbix protocol) or IIOP. This may be required for clients connecting to servers through a firewall. You should use this variable carefully and should not use it with _bind( ).
IT_DEFAULT_CODE	The default encoder to be used, for example, XDR.
IT_DEFAULT_COMMS	The default communications protocol to be used, for example, TCP/IP.
IT_DEF_NUM_NW_THREADS	The initial number of threads used in the new threading model for the internal network thread pool.
IT_DIAGNOSTICS_LEVEL	Controls the level of the diagnostic messages reported by Orbix.
IT_ENABLE_ANON_BIND_SUPPORT *	Allows a client built using an earlier version of Orbix to use anonymous binds omitting the marker name.
IT_ENABLE_MULTI_HOMED_SUPPORT	Enables multi-homed support for machines with multiple IP addresses. This is disabled by default and impacts on performance when enabled.
IT_ERRORS *	The full path name of the error messages file.
IT_FD_WARNING_NUMBER *	The number of file descriptors, which when exceeded, will cause an IOCallback warning to be generated if a callback has been registered. See CORBA::IT_IOCallback::AtFDLowLimit( )
IT_FD_STOP_LISTENING_POINT *	The number of file descriptors, which when exceeded, will stop the server from listening for new connections. An associated IOCallback warning may be generated if a callback has been registered. See CORBA::IT_IOCallback::StopListeningAtFDHigh( ) and CORBA::IT_IOCallback::ResumeListeningBelowFDHigh ( ).
IT_GIOP_VERSION	The version number of the GIOP protocol to be used.

Variable	Description
IT_IIOP_PORT	The port number to be used for server client connections when using IIOP.
IT_IIOP_VERSION	The IIOP version of IORs generated by Orbix servers, and IIOP messages understood by Orbix. Valid values are 10 and 11, representing IIOP 1.0 and IIOP 1.1, respectively. The default value is 11.
IT_LISTEN_QUEUE_SIZE *	The internal listener thread's queue size.
IT_LOCAL_ADDR_LIST	This is a colon-seperated list of IP addresses, or hostnames, on which Orbix servers and the Orbix daemon (orbixd) can expect to receive invocations. This variable is used as a part of multi-homed support.
IT_LOCAL_HOST	The name of the local host that is used in any IOR that is exported.
IT_MARKER_PATTERN *	Contains the marker pattern name that caused the server to be launched.
IT_ONEWAY_RESPONSE_REQUIRED *	A boolean variable that specifies if an IIOP reply is expected for an outgoing IIOP request containing a oneway operation. A response to a oneway is useful when you wish to catch system exceptions, or to enable the client to receive IIOP replies with LOCATION_FORWARD status. The default value is FALSE.
IT_SERVER_CODE *	The name of the encoder to be used by this server, for example, XDR.
IT_SERVER_COMMS *	The name of the communications protocol to be used by this server, for example, TCP/IP.
IT_SERVER_MARKER *	Contains the server marker name that caused the server to be launched.

Variable	Description
IT_SERVER_METHOD *	Contains the server method name that caused the server to be launched.
IT_SERVER_NAME *	Used to set the name of the server.
IT_SERVER_PORT *	The port number being used by the server when listening for new connections.
IT_USE_HOST_IN_IOR	Specifies whether the hostname or the host's IP address will appear in any exported IORs.
IT_USE_ORBIX3_STYLE_SYS_EXC *	Used to determine if Orbix 3.x style exceptions or new interoperable exceptions should be used. Specifically, it is used to determine if an OBJ_NOT_EXIST exception or an INV_OBJREF exception should be raised when an object is not found for a given IOR. It is also used to distinguish COMM_FAILURE and TRANSIENT errors.
IT_USE_REVERSE_LOOKUP *	Specifies if reverse lookup (i.e. determining the hostname from an IP address) is enabled.

* These configuration variables can be set using the SetConfigValue() API. See "CORBA::ORB::SetConfigValue()" on page 221 of the *Orbix C++ Programmer's Reference*.

Note: Entries in IONA configuration files are scoped with a prefix; for example, Common.IT_DAEMON_PORT or Orbix.IT_CONNECT_ATTEMPTS. Environment variables are not scoped. The scoped entries are also used by the SetConfigValue() and GetConfigValue() APIs.

For details of OrbixNames-specific configuration variables, refer to the *OrbixNames Programmer's and Administrator's Guide*.

Appendix B
Orbix Daemon Options

The daemon process, `orbixd`, takes the following options:

`-c` *filename*	Specifies the log file to use for check-point information. In the event that a daemon is terminated, this allows a new daemon to recover information about existing running servers. Unless an absolute path name is specified, the file is placed in a directory relative to that from which the daemon is launched.
`-f` *filename*	(NT-only) Redirects 'stdout' to the file when Orbix Daemon is started as an NT service. Unless an absolute path name is specified, the file is placed in a directory relative to that from which the daemon install command is given.
`-i` *filename*	Outputs the daemon's interoperable object reference (IOR) to the specified file. Unless an absolute path name is specified, the file is placed in a directory relative to that from which the daemon is launched.
`-j`	(NT-only) Installs the daemon as an NT service. The service must be started manually using the **Services** Control Panel. This starts the daemon with `<path>\orbixd -b`.

`-l` *number*	(UNIX-only) Specifies the maximum number of socket descriptors, and thus, the maximum number of connections to the daemon. This option does not exist on AIX machines. By default, the maximum number of descriptors is determined by the system's limit. This is normally `64`, although this can be increased by a system administrator. The current maximum number that can be set is `1024`.
`-o` *userId*	(UNIX only) Indicates that if the daemon runs with super-user privileges, servers launched by the daemon should run using the specified user ID instead of the root ID. Without this switch, a client running as root on a remote machine could launch a server with root privileges on a different machine. Using the -o switch reduces the security risks associated with easily faked remote user IDs, If the remote user is not root, the server is launched under the user ID of the client process sending the request. This is the default when the -o switch is not used.
`-p`	Runs the daemon in protected mode. In this mode, only clients running as the same user as the daemon are allowed to modify the Implementation Repository. No updates are accepted from remote hosts.
`-r` *time*	Specifies the frequency (in seconds) at which `orbixd`'s child processes should be reaped. The default is `60` seconds.
`-s`	Runs the daemon in silent mode. By default the daemon outputs some trace information.
`-t`	Outputs more than the default trace information while the daemon is running.
`-u`	Allows invocations on a manually-launched unregistered server. This means that the manually-launched (persistent) server does not have to be registered in the Implementation Repository.

`-x` *number*	Sets the time limit in seconds for establishing that a connection to the daemon is fully operational. The default is `30` seconds.
`-v`	Outputs the daemon version number and a summary of the configuration details that a new daemon process would use. Specifying `-v` does not cause a new daemon to be run.
`-w`	(NT only) Uninstalls the daemon as an NT service.

Appendix C
Command Reference

This appendix acts as a reference for the command-line interface to Orbix. The commands described in this appendix allow you to manage the Implementation Repository and the Interface Repository.

Command Summary

The following table shows the available commands:

Purpose	Commands
Configuration	`dumpconfig`
Server Registration	`putit`, `rmit`
Listing Server Information	`lsit`, `psit`, `catit`
Process Management	`pingit`, `killit`
Implementation Repository Directories	`mkdirit`, `rmdirit`
Security	`chownit`, `chmodit`
Interface Repository Management	`putidl`, `readifr`, `rmidl`

This appendix describes each command in alphabetical order.

catit

The `catit` command outputs full information about a given Implementation Repository entry.

Syntax

```
catit [-v] [-h host] [-x] server_name
```

Options

`-v`	Outputs the command version information.
`-h host`	Indicates which host to use.
`-x`	Display the port number / number of servers associated with this server. This information would have been registered with the putit utility using the -port/ -n switch

chmodit

The `chmodit` command modifies access control for a server. For example, use it to grant launch and invoke rights on a server to users other than the server owner.

Syntax

```
chmodit [-v] [-h host]
   { server | -a directory }
   { i{+,-}{user, group}|
    l{+,-}{user, group} }
```

Options

`-v`	Outputs the command version information.
`-h` *`host`*	Indicates which host to use.
`-a`	The `-a` option specifies that a user or group is to be added to an access control list (ACL) for a directory of servers.
`i+` `i-`	By specifying the `i` option, you can add a user or group to (`i+`) or removed from (`i-`) the invoke ACL.
`l+` `l-`	By specifying the `l` option, you can add a user or group to (`l+`) or removed from (`l-`) the launch ACL.

By default, only the owner of an Implementation Repository entry can launch or invoke the registered server. However, launch and invoke access control lists (ACLs) are associated with each entry in the Implementation Repository and you can modify these ACLs to give certain users or groups the right to launch or invoke a specific server, or a directory of servers.

There is also a pseudo-group name called `all` that you can use to implicitly add all users to an access control list.

chownit

The `chownit` command makes changes to the ownership of Implementation Repository entries and directories.

Syntax

```
chownit [-v] [-h host]
   { -s server_name  new_owner |
     -d directory { +, - } {user, group} }
```

Options

`-v`	Outputs the command version information.
`-h` *`host`*	Indicates which host to use.
`-s`	The `-s` option enables you to change the ownership of an Implementation Repository entry.
`-d`	The `-d` option modifies the ACL on a directory allowing you to add (+) or remove (-) a user or group from the list of owners of a directory.

Only the current owner of an Implementation Repository entry has the right to change its ownership.

An Implementation Repository directory can have more than one owner. An ownership access control list (ACL) is associated with each directory in the Implementation Repository, and this ACL can be modified to give certain users or groups ownership rights on a directory. Only a user on an ownership ACL has the right to modify the ACL.

Note: Spaces *are* significant in this grammar. Spaces must exist between an option and its argument, and on either side of the + or - that follows a directory.

Orbix supports the pseudo-group `all` which, when added to an ACL, grants access to all callers.

dumpconfig

The `dumpconfig` utility outputs the values of the configuration variables used by Orbix, and the location of the Orbix configuration files in your system. It also reports if there are any syntax errors in your configuration files.

Syntax

```
dumpconfig [-v]
```

Options

`-v`	Outputs the command version information.

killit

The `killit` command kills (stops) a running server process.

Syntax

`killit [-v] [-h` *`host`*`] [-m` *`marker`*`]` *`server_name`*

`-v`	Outputs the command version information.
`-h` *`host`*	Indicates which host to use.
`-m`	Specifies a marker value to identify a specific object, or set of objects, to which the `killit` command applies.

Where there is more than one server process, use the marker parameter to select between different processes. You require the marker parameter when killing a process in the unshared mode.

The `killit` command uses the `SIGTERM` signal. This command does not remove the entry from the Implementation Repository.

lsit

The `lsit` command lists entries in an Implementation Repository directory.

Syntax

`lsit [-v] [-h` *`host`*`] [-R] [`*`directory`*`]`

Options

`-v`	Outputs the command version information.
`-h` *`host`*	Indicates which host to use.
`-R`	Recursively lists all subdirectories and entries.

mkdirit

The `mkdirit` command creates a new registration directory.

Syntax

`mkdirit [-v] [-h` *`host`*`]` *directory*

Options

`-v`	Outputs the command version information.
`-h` *`host`*	Indicates which host to use.

Hierarchical names are extremely useful in structuring the name space of servers in Implementation Repositories.

pingit

The `pingit` command tries to contact an Orbix daemon to determine if it is running.

Syntax

`pingit [-v] [-h` *`host`*`]`

Options

`-v`	Outputs the command version information.
`-h` *`host`*	Indicates which host to use.

psit

The psit command outputs a list of server processes known to an Orbix daemon.

Syntax

```
psit [-v] [-h host]
```

Options

-v	Outputs the command version information.
-h *host*	Indicates which host to use.

One line is output for each server process. Each line has values for the following fields:

```
Name Marker Code Comms Port Status Per-Client? OS-pid
```

The fields are as follows:

Name	The server name.
Marker	The object marker pattern associated with the process.
Code	The data encoder used; for example, xdr.
Comms	The communications protocol used; for example, tcp.
Port	The port number used by the communications system.
Status	One of “automatic”, “manual” or “inactive”.
Per-Client?	Indicates whether the server is a per-client server.
OS-pid	The operating system process.

putidl

The putidl command allows you to add a set of IDL definitions to the Interface Repository. This command takes the name of an IDL file as an argument. All IDL definitions within that file are added to the repository.

The Interface Repository server must be available for this command to succeed.

Syntax

```
putidl {[-?] | [-v] [-h host] [-s] file}
```

Options

-?	Displays the allowed options for this command.
-v	Outputs the command version information.
-h *host*	Indicates the host at which the Interface Repository server is available.
-s	Indicates that the command should run in silent mode.

putit

The putit command creates an entry in the Implementation Repository that represents how Orbix can start a server.

Syntax

```
putit [-v] [-h host] [-per-client | -per-client-pid]
   [ [-shared | -unshared] [-marker marker] ]
   [-per-method [-method method] ]
   [ -j | -java [-classpath classpath | -addpath path ] ]
   [ -oc ORB_class -os ORB_singleton_class] [ -jdk2]
   [ -port iiop portnumber]
   [ -n number_of_servers ] [ -l ]
   serverName [<commandLine> | -persistent ]
```

Options

Executing `putit` without any arguments outputs a summary of its options. The options are as follows:

`-v`	Outputs the utility's version information without executing the command. This option is available on all of the utilities.
`-h` *host*	Specifies the host name on which to execute the `putit` command. By default, the command is executed on the local host.
`-per-client`	Specifies that a separate server process is used for each user. You can use this activation mode with the shared, unshared, or per-method modes.
`-per-client-pid`	Specifies that a separate server process is used for each client process. You can use this activation mode with the shared, unshared, or per-method modes.
`-shared`	Specifies that all active objects managed by a given server on a given machine are contained in the same process. This is the default mode.
`-unshared`	Specifies that as an object for a given server is invoked, an individual process is activated to handle all requests for that object. Each object managed by a server can (but does not have to) be registered with a different executable file—as specified in *commandLine*.

`-java`	The `-java` switch is an extension of the standard Orbix putit command. This indicates that the specified server should be launched via the Java interpreter. You can truncate this switch to `-j`.
`-oc` *ORB_class*	Passes `-Dorg.omg.CORBA.ORBClass=`*ORB_class* to the Java interpreter. You should use this switch with the `-os` switch. For OrbixWeb servers, the parameter to this switch should be as follows: `IE.Iona.OrbixWeb.CORBA.ORBClass.` You should pass this string to the Java interpreter before the server class name.
`-os` *ORB_singleton_class*	Passes `-Dorg.omg.CORBA.ORBSingletonClass=`*ORB_singleton_class* to the Java interpreter. You should use this switch with the `-oc` switch. For OrbixWeb servers the parameter to this switch should be `IE.Iona.OrbixWeb.CORBA.ORBSingletonClass`. This string must be passed to the Java interpreter before the servers class name. The `-os` and `-oc` switches provide foreign ORB support.

`-jdk2`	Passes the following system properties to the Java interpreter:
	`Dorg.omg.CORBA.ORBClass=`
	`IE.Iona.OrbixWeb.CORBA.ORB`
	`-Dorg.omg.CORBA.ORBSingletonClass=`
	`IE.Iona.OrbixWeb.CORBA.singletonORB`
	You must pass this string to the Java interpreter before the server class name. You should use this switch for OrbixWeb servers being executed by JDK1.2.

`-per method`	Specifies that each invocation to a server results in a process being activated to handle that request. Each method can (but does not have to) be registered with a different executable file—as specified in *command_line*.
`-port` *port*	Specifies a well-known port number for a server so that Orbix, if necessary, activates a server that communicates on the specified port number. Often required by servers that communicate over the CORBA Internet Inter-ORB Protocol (IIOP).

The following options apply to the shared mode:

`-n` *number_of_servers*	This switch is applicable only to servers registered in shared activation mode. It instructs the daemon to launch up to the specified number of servers. Each new client connection results in a new server being launched as long as the number of clients is less than the number specified in *number_of_servers*. When the number of clients equals the number of servers specified in *number_of_servers*, new clients are connected to running servers using a round robin algorithm. The default number of servers is 1.
`-persistent`	Specifies that the server can only be launched persistently, that is, manually. The server is never automatically launched by Orbix. If the `-u` option is passed to the Orbix daemon, such servers do not have to be registered in the Implementation Repository.

The following option applies to the shared and unshared modes:

`-marker` *marker*	Specifies a marker value to identify a specific object, or set of objects, to which the `putit` applies. Marker names specified using `putit` cannot contain white space.

The following option applies to the per-method mode:

`-method` *method*	Specifies a method name to identify a specific method, or set of methods, to which the `putit` applies.

Notes

The putit command is often used in its simplest form:

```
putit server_name command_line
```

The *command_line* is an absolute path name specifying the executable file that implements the server. Any command-line parameters to be given to the executable file are appended after the absolute path name in the putit command. These parameters are given to the server *every* time it is run by Orbix. However, the parameters must be given explicitly if the executable file is executed manually.

The default settings for putit mean that the simplest form of the command is equivalent to any of the following:

```
putit -shared server_name command_line
putit -shared -marker '*' server_name command_line
putit -marker '*' server_name command_line
```

By default, putit uses the Implementation Repository on the local host. By default, putit uses the shared activation mode. Therefore, on any given host, all objects with the specified server name are controlled by the same process. By default also, putit registers a server in the multiple-client activation mode. This means that all client processes bind to the same server process.

Server Activation Modes

Activation modes control how servers are implemented when they become processes of the underlying operating system.The primary activation modes are *shared*, *unshared*, and *per-method*:

- In shared mode, all of the objects with the same server name on a given machine are managed by one process on that machine. If a server is registered in shared mode, it can also be launched manually prior to any invocation on its objects.
- In unshared mode, individual objects are registered with the Implementation Repository, and a process is launched for each such object.

- In per-method mode, individual operations are registered with the Implementation Repository, and each invocation on an operation results in a separate process.

You should note the following:

- The default activation mode is shared.
- For a given server name, you can select only one of shared, unshared, or per-method.
- For each of the modes shared or unshared, a server can be registered in a secondary activation mode:
  - multiple-client
  - per-client
  - per-client-process

  The default is multiple-client activation, with the effect that a server process is shared between multiple principals and multiple client processes.

  Per-client activation results in a separate server process for each principal (end-user). Per-client-process activation results in a separate server process for each separate client process. Per-client and per-client-process activation is orthogonal to shared, unshared and per-method modes.
- Manually-launched servers behave in a similar way to shared activation mode servers. If a server is registered as unshared or per-method, the server fails if it is launched manually. This is in line with the CORBA specification.

Note: Per-method servers are activated for a single IDL operation call. As a result, the per-client flag is ignored for per-method servers.

Pattern Matching for Markers and Methods

Pattern matching specifies a set of objects for the `-marker` option, or a set of methods for the `-method` option. Pattern matching allows a group of server processes to share a workload between them, whereby each server process is responsible for a range of object marker values. The pattern matching is based on regular expressions, as follows:

*	Matches any sequence of characters.
?	Matches any single character.
[SET]	Matches any characters belonging to the specified set, for example, [abc].
[!SET]	Matches any characters *not* belonging to the specified set, for example, [!abc].
[^SET]	Equivalent to [!SET], for example, [^abc].

A SET, as presented above, is composed of characters and ranges. A range is specified using a hyphen character -.

Finally, since each of the characters *?!^-[]\ is special, in the sense that it is interpreted by the pattern matching algorithm; each of them can be preceded by a \ character to suppress its interpretation.

Examples of patterns are:

hello	matches “hello”.
he*	matches any text beginning with “he”, for example, “he”, “help”, “hello”.
he?	matches any three character text beginning with “he”, for example, “hec”.
[abc]	matches “a”, “b” or “c”.
he[abc]	matches “hea”, “heb” or “hec”.
[a-zA-Z0-9]	matches any alphanumeric character.
[!a-zA-Z0-9]	matches any non alphanumeric character.
_[gs]et_balance	matches _get_balance and _set_balance.
make*	matches makeDeposit and makeWithdrawal.

If an activation order exists in an Implementation Repository entry for a specific object marker or method and another exists for an overlapping set of markers or methods, the particular server that is activated for a given object is non-deterministic. This means that no attempt is made to find an entry registered for best or exact match.

readifr

The `readifr` command allows you to view an IDL definition stored in the Interface Repository. This command takes the fully scoped name of the IDL definition as an argument and displays that definition. Calling `readifr` with no arguments lists the contents of the entire Interface Repository.

The Interface Repository server must be available for this command to succeed.

Syntax

`readifr {[-?] | [-v] [-h` *`host`*`] [-d] [-t] [-c] [`*`definition_name`*`]}`

Options

`-?`	Displays the allowed options for this command.
`-v`	Outputs the command version information.
`-h` *`host`*	Indicates the host at which the Interface Repository server is available.
`-d`	Displays all derived types of an IDL interface.
`-c`	Indicates that the command should not prompt the user for input. This is useful when running `readifr` with no other arguments.

rmdirit

The `rmdirit` command removes a registration directory.

Syntax

`rmdirit [-v] [-h` *`host`*`] [-R]` *`directory`*

Options

`-v`	Outputs the command version information.
`-h` *`host`*	Indicates which host to use.
`-R`	Recursively deletes the directory and all of the Implementation Repository entries and sub-directories within it.

The `rmdirit` command returns an error if it is called without the `-R` option on a non-empty registration directory.

rmidl

The `rmidl` command allows you to remove an IDL definition from the Interface Repository. This command takes the fully scoped name of the IDL definition as an argument.

The Interface Repository server must be available for this command to succeed.

Syntax

`rmidl {[-?] | [-v] [-h` *`host`*`]` *`definition_name`*`}`

Options

`-?`	Displays the allowed options for this command.
`-v`	Outputs the command version information.
`-h` *`host`*	Indicates the host at which the Interface Repository server is available.

rmit

Removes an Implementation Repository entry or modifies an entry.

Syntax

```
rmit [-v] [-h host]
   [-marker marker | -method method] server_name
```

Options

`-v`	Outputs the command version information.
`-h` *`host`*	Indicates which host to use.
`-marker` *`marker`*	Specifies a marker value to identify a specific object, or set of objects, to which the `rmit` command applies.
`-method` *`method`*	Specifies a method name to identify a specific method, or set of methods, to which the `rmit` applies.

This command does not kill any currently running processes associated with a server.

You can use pattern matching for markers and methods as described in the `putit` command reference. Refer to putit on page 96.

Appendix D
Error Messages and Exceptions

Orbix has an external text file containing an explanation of all error messages, both for IDL compiler errors and system exceptions. Orbix outputs error messages from the file named by the IT_ERRORS environment variable or entry in the orbix3.cfg configuration file. This file contains Orbix-specific configuration variables.

Setting Error Messages

The standard error file can be edited for a particular installation if required. For example, by translating all of the text into French or German, or by providing more verbose explanations of errors than those provided in the standard Orbix release.

Each error is assigned a unique number, and the file contains a line for each error in the form:

error_number: *error_message_text*

Rather than changing the standard file distributed with the Orbix release, you can specify an alternative file by using the IT_ERRORS entry in the orbix3.cfg configuration file. You can also specify a file on a per user basis by setting the value of the IT_ERRORS environment variable to a file which contains the list of system error messages.

Within the IT_ERRORS file, comments can be inserted using "//", and "\" can be used as a continuation character if the message needs to extend past the end of a line. IDL compiler errors have been divided into pre-processing, syntax and semantic errors, and their error numbers are arranged within these divisions.

System Exceptions Defined by CORBA

The following table lists the system exceptions defined in the CORBA specification:

Identifier	Exception	Description
10000	UNKNOWN	The unknown exception.
10020	BAD_PARAM	An invalid parameter was passed.
10040	NO_MEMORY	Dynamic memory allocation failure.
10060	IMP_LIMIT	Violated implementation limit.
10080	COMM_FAILURE	Communication failure.
10100	INV_OBJREF	Invalid object reference.
10120	NO_PERMISSION	No permission for attempted operation.
10140	INTERNAL	ORB internal error.
10160	MARSHAL	Error marshalling parameter/result.
10180	INITIALIZE	ORB initialization failure.
10200	NO_IMPLEMENT	Operation implementation unavailable.
10220	BAD_TYPECODE	Bad TypeCode.
10240	BAD_OPERATION	Invalid operation.
10260	NO_RESOURCES	Insufficient resources for request.
10280	NO_RESPONSE	Response to request not yet available.
10300	PERSIST_STORE	Persistent storage failure.
10320	BAD_INV_ORDER	Routine invocations out of order.
10340	TRANSIENT	Transient failure; reissue the request.
10360	FREE_MEM	Cannot free memory.
10380	INV_IDENT	Invalid identifier syntax.
10400	INV_FLAG	Invalid flag was specified.
10420	INTF_REPOS	Error accessing interface repository.

Identifier	Exception	Description
10440	BAD_CONTEXT	Error processing context object.
10460	OBJ_ADAPTOR	Failure detected by object adaptor.
10480	DATA_CONVERSION	Data conversion error.

System Exceptions Specific to Orbix

The following table lists system exceptions specific to Orbix:

Identifier	Orbix Exception	Description
10500	FILTER_SUPPRESS	Suppress exception raised in per-object pre-filter.
10540	ASCII_FILE	ASCII file error.
10560	LICENCING	Licensing error.
10600	IIOP	IIOP error.
10620	NO_CONFIG_VALUE	No configuration value set for one of the mandatory configuration entries.

Index

K

L

M

N

T

U

V

X