MICRO FOCUS

Orbix 3.3.16

Administrator's Guide Java Edition

Micro Focus
The Lawn
22-30 Old Bath Road
Newbury, Berkshire RG14 1QN
UK

http://www.microfocus.com

2020-11-26

Contents

Preface

The ***Orbix Administrator's Guide Java Edition*** describes the command-line utilities and GUI tools used during Orbix Java setup and administration.

Audience

The ***Orbix Administrator's Guide Java Edition*** is designed as an introduction for Orbix Java administrators and programmers. It is assumed that you are familiar with relevant sections of the ***Orbix Programmer's Guide Java Edition*** and the ***Orbix Programmer's Reference Java Edition***.

Organization of this Guide

This guide is divided into the following three parts:

Part I "Orbix Java Administration"

- Overview of Orbix Java Administration

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

  This chapter describes how to configure Orbix Java and how to use the Orbix Java configuration Advanced Programming Interfaces (APIs).
- Managing the Implementation Repository

  This chapter explains more about using the Implementation Repository including registering servers, displaying and organizing server entries, and security issues.
- Managing the Interface Repository

  This chapter describes how to configure Orbix Java to store object interface definitions so that the applications can learn about them at runtime.
- Using Orbix Java on the Internet

  This chapter describes how client applets can overcome security restrictions using signed applets.

Part II "Orbix Java GUI Tools"

- Orbix Java Configuration Explorer

  This chapter describes how you can configure an OrbxWeb installation using the Orbix Java Configuration Tool.
- The Orbix Java Server Manager

  This chapter describes how you can register servers in the Implementation Repository using the Orbix Java Server Manager.
- The Interface Repository Browser

  This chapter describes how you can add IDL definitions to the Interface Repository using the Interface Repository browser.

Part III "Appendices"

- Orbix Java Configuration Variables

  This appendix shows the configuration parameters that Orbix Java recognizes.
- Orbix Java Daemon Options

  This appendix describes the start-up options that the Orbix Java daemon can use.
- Orbix Java Command-Line Utilities

  This appendix describes the syntax and the options for each Orbix Java command you can use.
- System Exceptions

  This appendix outlines the system exceptions defined by CORBA, and the system exceptions that are specific to Orbix Java.

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 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.

Contacting Micro Focus

Our Web site gives up-to-date details of contact numbers and addresses.

Further Information and Product Support

Additional technical information or advice is available from several sources.

The product support pages contain a considerable amount of additional information, such as:

- The WebSync service, where you can download fixes and documentation updates.
- The Knowledge Base, a large collection of product tips and workarounds.
- Examples and Utilities, including demos and additional product documentation.

To connect, enter http://www.microfocus.com in your browser to go to the Micro Focus home page.

Note:
Some information may be available only to customers who have maintenance agreements.

If you obtained this product directly from Micro Focus, contact us as described on the Micro Focus Web site, http://www.microfocus.com. If you obtained the product from another source, such as an authorized distributor, contact them for help first. If they are unable to help, contact us.

Information We Need

However you contact us, please try to include the information below, if you have it. The more information you can give, the better Micro Focus SupportLine can help you. But if you don't know all the answers, or you think some are irrelevant to your problem, please give whatever information you have.

- The name and version number of all products that you think might be causing a problem.

- Your computer make and model.
- Your operating system version number and details of any networking software you are using.
- The amount of memory in your computer.
- The relevant page reference or section in the documentation.
- Your serial number. To find out these numbers, look in the subject line and body of your Electronic Product Delivery Notice email that you received from Micro Focus.

Contact information

Our Web site gives up-to-date details of contact numbers and addresses.

Additional technical information or advice is available from several sources.

The product support pages contain considerable additional information, including the WebSync service, where you can download fixes and documentation updates. To connect, enter http://www.microfocus.com in your browser to go to the Micro Focus home page.

If you are a Micro Focus SupportLine customer, please see your SupportLine Handbook for contact information. You can download it from our Web site or order it in printed form from your sales representative. Support from Micro Focus may be available only to customers who have maintenance agreements.

You may want to check these URLs in particular:

- http://www.microfocus.com/products/corba/orbix/orbix-6.aspx (trial software download and Micro Focus Community files)
- https://supportline.microfocus.com/productdoc.aspx. (documentation updates and PDFs)

To subscribe to Micro Focus electronic newsletters, use the online form at:

http://www.microfocus.com/Resources/Newsletters/infocus/newsletter-subscription.asp

Part I

Orbix Java Administration

In this part

This part contains the following:

Overview of Orbix Java Administration

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

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

An Orbix Java 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 Java Architecture

An Orbix Java client invokes IDL operations on a distributed object using normal Java function calls, as if the object were located in the client's address space. Orbix Java 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 Java receives these messages and translates them to function calls on the target object, as shown in Figure 1.

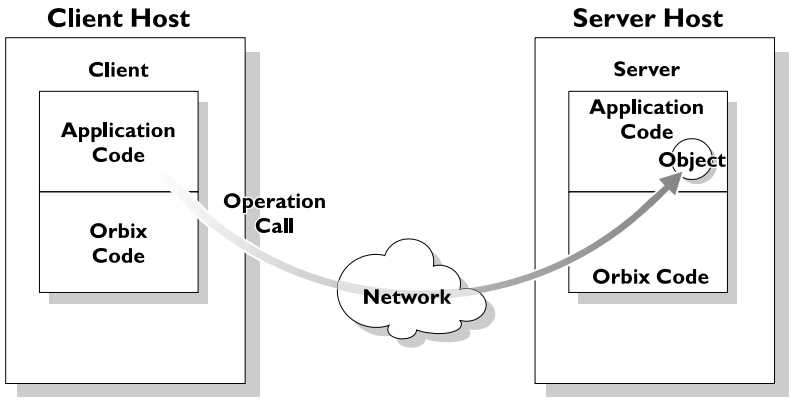

Figure 1: *An IDL Operation Call on a Distributed Object*

Servers and the Implementation Repository

Each Orbix Java 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 Java daemon at the server host attempts to launch the server process automatically.

To allow an Orbix Java daemon to manage the server processes running in the system, Orbix Java 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.

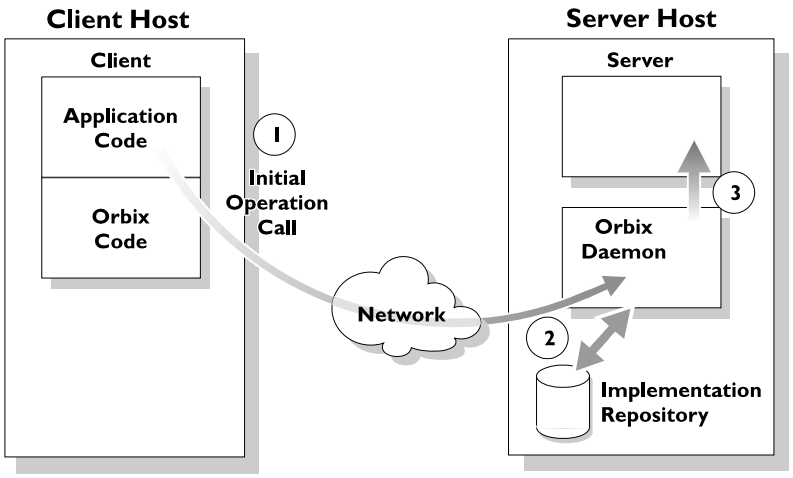

Figure 2: *Automatic Launch of an Orbix Server Process*

As shown in Figure 2, the Orbix Java 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 Java 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 Java 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 Java 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 Interface Repository by contacting an Interface Repository server. This is a standard Orbix Java 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 Java applications to run in your network, you must do the following:

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

Part I of this guide, Orbix Java Administration, presents the configuration files and command-line utilities that allow you to achieve each of these tasks.

Part II of this guide, Orbix Java GUI Tools, presents the graphical user interfaces that provide an alternative way to manage Orbix components.

Configuring Orbix Java

You may need to change the default Orbix Java configuration settings. Orbix Java provides several mechanisms to aid configuration. This chapter describes the Orbix Java configuration format and how to use the Orbix Java configuration APIs.

You may need to change default configuration settings for a variety of reasons, including the following:

- Enabling or disabling parts of the functionality.
- Altering the use of specific port numbers.
- Optimizing the size of tables used to track objects in servers.
- Reducing the number of classes downloaded for use in applets.

Accessing Configuration Parameters

You can get and set the values of Orbix Java configuration parameters using the following mechanisms.

- Using the Configuration Explorer to access configuration files.
- Using Orbix Java system properties.
- Using an applet's `<param>` HTML tag.
- Using an application's command-line parameters.
- Using Java system properties; for example, loaded from a file.

Configuration Parameter Formats

The various configuration parameter-retrieval mechanisms need to use slightly different formats to store the parameters and their values. In the examples that follow, the string *`IT_PARAMETER`* represents the Orbix Java configuration parameter being set, while *`value`* represents the value it is set to.

Mechanism	Format

Configuration Files	`OrbixWeb {`
	`  IT_PARAMETER=value`
	`}`

System Properties	`-DOrbixWeb.IT_PARAMETER=value`

Applet Tags	`<PARAM NAME="OrbixWeb.IT_PARAMETER"`
	`VALUE="value">`
	The applet tags must be used in the HTML document that loads the applet, between the `<APPLET>` and `</APPLET>` HTML tags.

Command-Line Arguments	`-OrbixWeb.IT_PARAMETER=value`

Note: You can use the CODEBASE attribute of the <APPLET> tag to specify the location of files required by the applet. These include packages such org.omg.CORBA and the Orbix Java configuration files. Refer to "Developing Applets with Orbix Java" in the ***Orbix Programmer's Guide Java Edition***. You will need to use the ARCHIVE attribute to specify the Orbix Java runtime OrbixWeb.jar.

Scoped Configuration Format

Configuration parameters common to multiple Micro Focus CORBA products are scoped within the Common prefix; for example, Common.IT_DAEMON_PORT. Orbix Java-specific configuration parameters are scoped within the OrbixWeb prefix; for example, OrbixWeb.IT_HTTP_TUNNEL_PORT.

Using Orbix Java Configuration Files

By default, the Orbix Java configuration files are located in the config directory of your installation. Orbix Java provides a convenient configuration editor in the form of the Orbix Java Configuration Explorer GUI tool. Refer to "Orbix Java Configuration Explorer" on page 37 for details. This is the recommended way to access Orbix Java configuration files.

By default, the configuration files are named as follows:

- iona.cfg
- common.cfg
- orbixweb3.cfg
- orbixnames3.cfg
- orbix3.cfg

For backwards compatibility, Orbix Java can also use OrbixWeb.properties and Orbix.cfg files that shipped with previous versions of Orbix Java.

Configuring Root Settings

You can configure your root settings by editing the iona.cfg file. This is the root configuration file used by Orbix Java. This file uses the include command to link to all other IONA configuration files. You can also edit this file to include links to customized configuration files.

The default, iona.cfg file contains the following information:

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

include cfg_dir + "common.cfg";
include cfg_dir + "orbixnames3.cfg";
include cfg_dir + "orbixweb3.cfg";
include cfg_dir + "orbix3.cfg";
```

You should set the cfg_dir parameter to <orbix_install_dir>\config\.

Configuring Common Parameters

You can configure your common settings by editing the `common.cfg` file. This file contains a list of configuration parameters that are common to multiple Micro Focus CORBA products. The configuration parameters 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:\JDK\bin\java.exe";

   # The default classpath used when Java servers
   # are automatically launched by the daemon.
   IT_DEFAULT_CLASSPATH = cfg_dir +
   "C:\JDK\jre\lib\rt.jar;C:\microfocus\lib\orbixweb.jar";
};
```

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 parameters required by Orbix Java. You can change these default settings by manually editing the configuration file, or by using the Configuration Explorer, or by setting a parameter in the user environment. An environment parameter, if set, takes precedence over the value set in the configuration file. Environment parameters are not scoped with a `Common.` prefix.

Configuring Orbix Java-Specific Parameters

You can configure your Orbix Java-specific settings by editing the `orbixweb3.cfg` file. This file contains configuration parameters that are specific to Orbix Java only. The configuration parameters in this file are declared within the scope `OrbixWeb{...}`.

You can also use the prefix `OrbixWeb.` to refer to individual entries in this file. For example, `OrbixWeb.IT_ANY_BUFFER_SIZE`.

Note: Orbix Java uses the `IT` prefix to distinguish its configuration parameters.

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

Finding Orbix Java Configuration Information

The `dumpconfig` utility enables you to obtain information about your Orbix configuration. This utility outputs the values of the configuration parameters used by Orbix Java, and the location of the Orbix Java 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 Java. The `dumpconfig` utility is especially useful if you need to know where Orbix Java is being configured from.

The `orbixdj -V` command also enables you to obtain information about your Orbix Java configuration. This outputs the current values of the configuration parameters used by Orbix Java.

Using Configuration API Calls

You can get and set Orbix Java configuration variables using the methods provided in class `IE.Iona.OrbixWeb.Features.OrbConfig`. Orbix Java configuration is on a per-ORB basis, allowing support for multiple ORBs.

Accessing Configuration Items

You can use the following methods to get and set specific configuration parameters by passing the name of the parameter as a string:

```
public String getConfigItem(String);

public synchronized void setConfigItem(String, String);
```

Note: Because Orbix Java configuration is on a per-ORB basis, `OrbConfig` calls should be made on the object returned by calling `config()` on the selected ORB; for example, `myOrb.config().getConfigItem("IT_BIND_USING_IIOP")`.

Accessing Configuration Properties

You can use the following methods to get and set multiple configuration parameters at once, using the `java.util.Properties` object:

```
public synchronized Properties getConfiguration();

public synchronized void setConfiguration(Properties);
```

The `getConfiguration()` method returns the configuration parameters that you set programmatically.

To set configuration, you must first set your configuration parameters programmatically and then pass your `Properties` object to the `setConfiguration()` method.

Accessing Configuration Files

You can use the following method to set your configuration from a specified configuration file:

```
public synchronized void setConfiguration(String);
```

Your specified configuration file must be included on the classpath.

To obtain all of the currently set parameters, use the following method:

```
public Hashtable getConfigFile();
```

There is also an API call available for emergency use, if you accidentally delete your configuration file. A call to this API returns a string containing the default values:

```
public String defaultConfigFile()
```

Refer to the ***Orbix Programmer's Reference Java Edition*** for more details on class `OrbConfig`.

Using Orbix Java System Properties

You can use the `ORB.init()` call to configure Orbix Java using system properties. The `ORB.init()` method is a standard part of the OMG Java mapping, and should be used by all Orbix Java applications and applets.

The API calls are as follows:

```
org.omg.CORBA.ORB.init (Applet app, Properties
                        props);

org.omg.CORBA.ORB.init (String[] args, Properties
                        props);

org.omg.CORBA.ORB.init (Properties props);

org.omg.CORBA.ORB.init ();
```

Note: Calling `ORB.init()` without parameters returns a singleton ORB with restricted functionality. Refer to the class `omg.org.CORBA.ORB` in the ***Orbix Programmer's Reference Java Edition***.

If any of the parameters are null, they are not used for configuration. If the `props` parameter is null, the default system properties are used instead.

You should pass the initialization method for applets a `this` parameter, representing the applet object itself. This allows the Orbix Java code to search for Orbix Java-specific applet tags.

Using Command-Line Arguments

The call to initialize Orbix Java from an application's `main()` method is as follows. This sample code also illustrates how an application that wishes to use other command-line arguments can skip over the ORB parameters, since the Orbix Java arguments all start with the string `"-OrbixWeb."`.

```
// Initialize the ORB.
org.omg.CORBA.ORB.init (args, null);
// Now read in the command-line parameters, and
// ignore any of the OrbixWeb ones.
    for (int i = 0; i < args.length; i++) {
    String ignore = "-OrbixWeb.";
    if (args[i].length() < ignore.length() ||
                    !(args[i].substring (0,

ignore.length())).equalsIgnoreCase
                            (ignore)){
    // This is a non-OrbixWeb command-line
    // parameter, take appropriate action.
                }
                }
// Your application initialization code can now
// continue...
```

An alternative is to simply parse your own command-line argument format and set the parameters using the API calls. However, the above command-line parsing mechanism provides a built-in way to do this.

Using Java System Properties

You can also use the Java system properties to pass configuration parameters. However, there is no standard way to set Java system properties. The JDK, for example, uses a file containing a list of the property names and values, and most web browsers do not allow properties to be set at all. The most useful way to use this functionality is by passing in parameters using the JDK Java interpreter's `-D` command-line switch. This approach supplements the command-line argument support.

Refer to "Orbix Java Configuration Variables" for a full table of Orbix Java configuration parameters.

Managing the Implementation Repository

When you install server applications on a network host, you must register those servers in the Implementation Repository. This repository allows Orbix Java 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 using the Orbix Java command-line utilities.

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 (ACLs).
- How to register servers in specialized activation modes rather than simply one server process for all clients.
- How to manage the set of ports Orbix Java uses to run servers.

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

  Because server names can be hierarchical, 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 Java starts the server.

Basic Implementation Repository Usage

Use the `putitj` command to register or modify an Implementation Repository entry. The general form of the `putitj` command is as follows:

```
putitj switches server_name command_line
```

where `<command line>` is usually an absolute path name specifying an executable file that implements the server. This can also be a shell command or script.

Note: The availability of a given feature depends on which Orbix Java daemon is running `orbixd` or `orbixdj`. Features labelled `orbixd` are currently not supported by `orbixdj`. Refer to the ***Orbix Programmer's Guide Java Edition*** for details of the differences between `orbixd` and `orbixdj`.

Registering a Server using Putitj

Orbix Java servers are implemented as Java classes and should be registered using the `-java` switch to `putitj`. This switch allows you to specify a class name (and an optional classpath) as follows:

```
putitj switches server_name -java
                    class_name class _arguments
```

This command specifies that the Orbix Java daemon, when launching the server, should invoke the Java interpreter on the specified bytecode. Any command-line parameters to the target class are appended after the class name in the `putitj` command. These parameters are passed to the server *every* time it is run by Orbix Java. However, the parameters must be stated explicitly if the server is launched manually.

Specifying a Classpath for an Orbix Java Server

The Orbix Java configuration variable `IT_DEFAULT_CLASSPATH` specifies the default classpath used by the Orbix Java daemon when launching all Java servers. The `putitj` command enables you to override `IT_DEFAULT_CLASSPATH` for a given server.

To do this, you should register the server with the -classpath switch, followed by the full class path for that server:

```
putitj switches server_name -java
                -classpath full_classpath
                  class_name class_arguments
```

For example:

```
putitj BankSrv -java -classpath
     /vol/jdk/classes:/orbixweb/classes BankerClass
```

Specifying a Partial Classpath for an Orbix Java Server

As an alternative, Orbix Java also allows a partial classpath to be specified during server registration. This partial class path will be appended to the value of IT_DEFAULT_CLASSPATH if the Orbix Java daemon attempts to launch the specified server. Use the -addpath switch to specify a partial class path:

```
putitj switches server_name -java
                -addpath partial_classpath
                     class_name class_arguments
```

For example, you can achieve the functionality of the -classpath example given above by setting IT_DEFAULT_CLASSPATH to the value /vol/jdk/classes and registering the server BankSrv as follows:

```
putitj BankSrv -java -addpath
                /orbixweb/classes BankerClass
```

Specifying the Location of the Java Interpreter

The Orbix Java daemon must be able to locate the Java interpreter to launch Java servers registered in the Implementation Repository. To enable this, you must set the value of the configuration variable IT_JAVA_INTERPRETER in the common.cfg file, as described in "Configuring Orbix Java".

Passing Parameters to the Java Interpreter

Conceptually, the classpath details, class name and class arguments specified during the registration of an Orbix Java server are passed directly to the Java Interpreter when the server is launched. If specific parameters also need to be passed to the Java interpreter, you can add these to the putitj command as follows:

```
putitj switches server_name -java
                -- interpreter_switches class_name
                     class_parameters
```

The string after the -- switch is passed to the Java interpreter instead of the standard class name and class arguments. You must insert a space after the -- switch, as shown in the following example:

```
putitj -java GridSrv -- -ms200m -mx200m
                              grid.javaserver1
```

Although registering a full Java Interpreter command as an *executable* file for an Orbix Java server appears to achieve similar functionality, this is *not* an acceptable alternative. The -java switch significantly alters the internal server launch strategy of the Orbix Java daemon, and an Orbix Java server should not be registered without this switch.

Registering a Server on a Remote Host

The following command registers a shared server called `FirstTrust` on the remote host `alpha`, with the specified class name:

```
putitj -h alpha FirstTrust -java BankClass arg1
```

Using the `-h` *hostname* option enables you to use all the 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.

The following command registers the same shared server and also sets the "`OrbixWeb.setDiagnostics`" property to "255".

```
putitj -h alpha FirstTrust -java
      -- -DOrbixWeb.setDiagnostics=255 BankClass
```

Organizing Servers into Hierarchies

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

```
mkdiritj banking
putitj banking/Berliner -java BankClass
```

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

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

For example:

```
lsitj
                  FirstTrust
                  banking
rmdiritj banking
                  directory not empty
rmdiritj -R banking
```

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

To move an entry in the hierarchy, first remove it with the `rmitj` command and then re-register it with the `putitj` command.

Removing a Registered Server

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

```
rmitj FirstTrust
```

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

You can also use the `rmitj` 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 "Server Activation Modes" on page 22.

Listing Registered Servers

Use the `lsitj` command to list registered servers and directories. For example, if you have registered a server called `International` and another called `printer`:

```
putitj International -java
                     -classpath /usr/users/joe banker
putitj printer -java laser
```

the output of the `lsitj` command is as follows:

```
International
printer
```

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

Displaying a Server Entry

Use the `catitj` command to display information about a specific server's registration entry. The following example assumes that the `International` server is registered as in the previous example, and that `catitj International` is entered at the command line:

```
Details for server : International

                    Comms. Protocol   : tcp
                    Code              : cdr
                    Activation Mode   : shared
                    Owner             : jbloggs
                    Launch            : ;all;
                    Invoke            : ;all;

                    Marker Launch Command

                       ###ORBIXWEB### banker
```

The output can include the following:

`Owner`	The user who put 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 (orbixd)`	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.

Contacting an Orbix Java Daemon

Use the `pingitj` utility to contact an Orbix Java daemon to determine if it is running on a specified host. This outputs a success or failure message, as appropriate; for example:

```
[ New Connection (joe.dublin.iona.ie,IT_daemon,*,,pid=230)
   ]
Trying to contact daemon at joe.dublin.iona.ie and it is
   running.
```

Starting Servers Manually

Most servers are designed to have Orbix Java 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 Java are useful for a number of reasons. For example, if a server takes a long time to initialize and it starts when a client requests 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. Manually launched servers are also known as *persistent servers* in CORBA terminology.

Registering a Manual Server

(orbixd)

All servers 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 Java from starting a server and make it manual-only, use the following command:

```
putitj FirstTrust -persistent
```

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

The CORBA specification requires that unshared or per-method types of server 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 `putitj` command. These unshared and per-method servers are described in "Server Activation Modes" on page 22.

Starting the Orbix Java Daemon for Unregistered Servers

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

```
orbixdj -u
```

When Orbix Java 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. The daemon does not check if this is a server name known to it.

A disadvantage of this approach is that an unregistered server is not known to the daemon. This means that the daemon cannot automatically invoke the Java interpreter on the server bytecode when a client binds to, or invokes an operation on, one of its objects. If a client invocation is to succeed, the server must be launched in advance of the invocation.

In a Java context, a more significant disadvantage of this approach is that the Orbix Java daemon is involved in initial communications between the client and server, even though the server is not registered in the Implementation Repository. This restriction applies to all Orbix Java servers that communicate over the standard Orbix communications protocol, and limits such servers to running on hosts where an Orbix Java daemon process is available.

Refer to "Activation Issues Specific to IIOP Servers" on page 27 for more information on unregistered servers.

Stopping Servers

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

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

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

   ```
   killitj -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 `killitj` command format:

   `killitj -m` *marker server_name*

Security of Registered Servers

For each Implementation Repository entry, Orbix Java 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 ACL 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 to ACLs, or removing groups and users from ACLs.

Note: The Java daemon (`orbixdj`) does not support access rights for user groups. An exception to this is the pseudo-user group `all`.

Modifying Server Access

Use the `chmoditj` command to modify the launch or invoke ACLs. For example:

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

   ```
   chmoditj AlliedBank l+chris
   ```

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

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

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

   ```
   chmoditj -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`:

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

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

Changing Owners of Registered Servers

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

```
chownitj -s AlliedBank mcnamara
```

An Implementation Repository directory can 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 are as follows:

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

   ```
   chownitj -d banks/investmentBanks + microfocus
   ```

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

   ```
   chownitj -d banks/investmentBanks - mcnamara
   ```

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

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

   Spaces *are* significant in this command. For example, the following command is correct:

   ```
   chownitj -d banks/investmentBanks + microfocus
   ```

   However, the following command is incorrect:

   ```
   chownitj -dbanks/investmentBanks + microfocus
   ```

Refer to "Orbix Java Command-Line Utilities" for a complete list of the Orbix Java utilities and their switches.

Determining the User and Group IDs of Running Servers

(orbixd)

On Windows platforms, the user ID `uid` and group ID `gid` of a server process launched by the Orbix Java 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 Java daemon are determined as follows:

- If `orbixd` is not running as a superuser, 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.
- 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`.
- 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.

You should *not* run `orbixd` as `root`. This would allow a client running as `root` on a remote machine to launch a server with `root` privileges on a different machine.

You can avoid this security risk by setting the `set-uid` bit of the `orbixd` executable and giving ownership of the executable to a user called, for example, `orbixusr` who does not have `root` privileges. Then `orbixd`, and any server launched by the daemon, do not have root privileges. Any servers that must be run with different privileges can have the `set-uid` bit set on the executable file.

Server Activation Modes

Orbix Java 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 Java.

Note: The availability of a given activation mode depends on which Orbix Java daemon is running `orbixd` or `orbixdj`. Activation modes labelled `orbixd` are currently not supported by the Orbix Java daemon `orbixdj`.

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 Java routes the invocation to that process; otherwise, Orbix Java launches a process.

Unshared Activation Mode
(orbixd)

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 for each 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 Activation Mode
(orbixd)

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 activation mode is the most commonly used. 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

(orbixd)

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

```
putitj -unshared NationalTrust -java 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 Java 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 Java.
- 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.

Server activation policies can specify individual object marker names; this is because objects can be named shared and unshared.

For example:

1. `putitj -marker College_Green NationalBank -java BankClass`

   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. ```
   putitj -unshared -marker College_Green FirstNational
                    -java BankClass
   putitj -unshared -marker St_Stephens_Green
                    FirstNational -java BankClass
   ```

   The first command registers an unshared server called `FirstNational` on the local host with the specified executable files. 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. Specifically, you can register a server activation policy for a subset of the server's objects. Because the number of objects named can become 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:

```
putitj -marker '[0-4]*' NationalBank -java NBBank
putitj -marker '[5-9]*' NationalBank -java 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 `putitj` command in "Orbix Java Command-Line Utilities" for a complete list of recognized patterns with examples.

Use the `rmitj` 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 `rmitj` command's marker option.

Registering Per-Method Servers

(orbixd)

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.

   ```
   putitj -per-method -method
   makeWithdrawal
      NationalTrust -java NTbank
   ```

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

   ```
   putitj -method makeDeposit
   NationalTrust
      -java NTbank
   ```

   You can specify patterns for methods so that operation names matching a particular pattern cause Orbix Java 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:

```
putitj -per-method FirstTrust -method
'make*'
   -java 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 `rmitj` 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 `rmitj` 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 `putitj` option is required to specify this mode when registering a server.

Per-Client Activation Mode
(orbixd)

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

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

Per-Client-Process Activation Mode
(orbixd)

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

Use the `putitj -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:

```
putitj -per-client-pid FirstTrust -java 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 Java daemon activates a server, 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

(orbixd)

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

```
putitj -port 1597 FirstTrust
    -java -classpath /work/bank banker
```

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

If an Orbix Java 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 Java daemon by using the configuration entries `IT_DAEMON_SERVER_BASE` and `IT_DAEMON_SERVER_RANGE` in the `common.cfg` configuration file. 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 Java 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`.

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 should set `IT_DAEMON_SERVER_BASE` and `IT_DAEMON_SERVER_RANGE` values using the Orbix Java Configuration Explorer—refer to "Orbix Java Configuration Explorer" for details. You should ensure that the values you set 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.

Activation Issues Specific to IIOP Servers

You do not have to register all Orbix Java servers communicating over IIOP in the Implementation Repository. An IIOP server can publish Interoperable Object References (IORs) for the implementation objects it creates, and then await incoming client requests on those objects without contacting an Orbix Java daemon.

Unregistered IIOP servers are important in a Java domain. This is because they can be completely independent of any supporting processes that may be platform-specific. In particular, any server that relies on the `orbixd` daemon to establish initial connections depends on the availability of the daemon on specific platforms. However, you can overcome this problem by using the Java daemon, `orbixdj`, which is platform-independent. An Orbix Java unregistered IIOP server is completely self-contained and platform-independent.

However, an unregistered IIOP server does have an important disadvantage. The TCP/IP port number on which a server communicates is embedded in each IOR that a server creates. If the port is dynamically allocated to a server process on start-up, the port may differ between different processes for a single server. This may invalidate IORs created by a server if, for example, the server is killed and relaunched. Orbix Java addresses this problem by allowing you to assign a well-known IIOP port number to the server.

These issues are discussed in more detail in the ***Orbix Programmer's Guide Java Edition*** .

Managing the Interface Repository

The Interface Repository is the component of Orbix Java 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.

This chapter explains how to manage the Interface Repository using Orbix command-line utilities. Refer to "The Interface Repository Browser" for details of how you can use Orbix GUI tools.

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.

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 using the Orbix Java Configuration Explorer GUI tool. 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 Java daemon is running, stop it and then restart it so that it recognizes the new configuration variable.

Registering the Interface Repository Server

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

```
putitj IFR /opt/microfocus/orbix33/bin/ifr
```

Orbix Java 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:

```
/opt/microfocus/orbix33/bin/ifr
```

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

The `IFR` server executable file takes the following options:

`-?`	Print a summary of switches.
`-h`	Specify an IFR server host name.
`-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` *seconds*	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 Java 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.

The syntax for the `putidlj` command is:

```
putidl { [-?] | [-v] [-h <hostname>]
                    [-s <filename for output>]
                    [-I<path>] <IDL file name> }
```

Refer to "Orbix Java Command-Line Utilities" for a full description of each option.

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`, enter the following:

```
readifr Finance::Bank
```

This utility prints the IDL definition to the standard output. (Note that the C++ scoping operator is used in IFR scoped names.)

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
```

You can also invoke `readifr` with no arguments, in which case the default is to output the whole repository. Because the repository may be very large, you are prompted to confirm this operation.

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
```

You should only use the `rmidl` utility to remove old or incorrect entries.

Note: Refer to "Orbix Java Command-Line Utilities" for a full description of the Orbix Java utilities and their options.

Using Orbix Java on the Internet

Orbix Java client applets are, like any applet, subject to security restrictions imposed by the browser in which they execute. The most fundamental of these restrictions include the inability to access local disks and the inability to contact an arbitrary Internet host. This chapter describes how client applets can get around these restrictions in a secure manner, using signed applets.

Applet Signing Technology

For security reasons, an applet is prevented from accessing the local file system and connecting to a host other than the host from which it was downloaded. Often these restrictions must be relaxed, in order for an applet to be fully functional. It is possible to achieve this using signed applet technology.

A signed applet has a digital signature which is interpreted as a sign of good intent. An applet that has been signed with a trusted digital signature may therefore be treated more permissively by a browser, and may even be granted the permission of a full application.

The following section provides a brief overview of signed applet technology.

Overview

There is no single standard implementation of applet-signing technology, however the implementation offered by Microsoft is widely adopted. Specific details of these vendors implementations are available from their corporate Web sites. In this section, discussion is limited to the implementation independent characteristics of the technology.

How Applets are Signed

Applets may be signed using public key cryptography technology. Distributors of the applet must digitally sign the applet with their private key. When a signed applet is downloaded by a browser, it can determine the identity of the signing entity by consulting a Certification Authority. A Certification Authority is a trusted third party that verifies the identify of a key holder. The browser may also determine whether the applet has been tampered with. Assuming there are no problems, the browser may assume that the applet is not malicious, and grant it extended privileges.

The user must ultimately grant the applet these extended privileges, either by configuring browser security settings or responding at runtime to individual requests for privileges from the applet. In some circumstances it may be the case that an applet does not function correctly unless it is granted extended privileges.

The benefits of signed applet technology to the Orbix Java applet programmer include the following:

- The ability to contact any host.
- The ability to cache information locally on disk.
- The ability to access system properties.

It is common for the applet, other classes it requires and associated files to be bundled into a single `archive` file. In this case, it is the archive that is signed and downloaded to the browser, thereby reducing download time.

Looking Ahead

It is expected that browsers will be able to support multiple archives in the future. Deployment should then become more flexible and efficient as applications can be split into a number of archives, each containing classes pertaining to a particular area of functionality. For example, an Orbix Java applet may be split into archives containing the Orbix Java runtime, the Java classes generated by the IDL compiler, the applet code and finally third party archives.

The Orbix Java installation includes Microsoft `CAB` (signed) and Netscape `JAR` (unsigned) compatible archives. They can be found in the `classes` directory of your Orbix Java installation.

Part II

Orbix Java GUI Tools

In this part

This part contains the following:

Orbix Java Configuration Explorer

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

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

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

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

- Common to multiple Micro Focus CORBA products.
- Orbix Java-specific.
- OrbixNames-specific.

Starting the Configuration Explorer

You can run the Orbix Configuration Explorer from the Windows **Start** menu, or by entering `configurationexplorer` at the command line. The Configuration Explorer appears as shown in Figure 3.

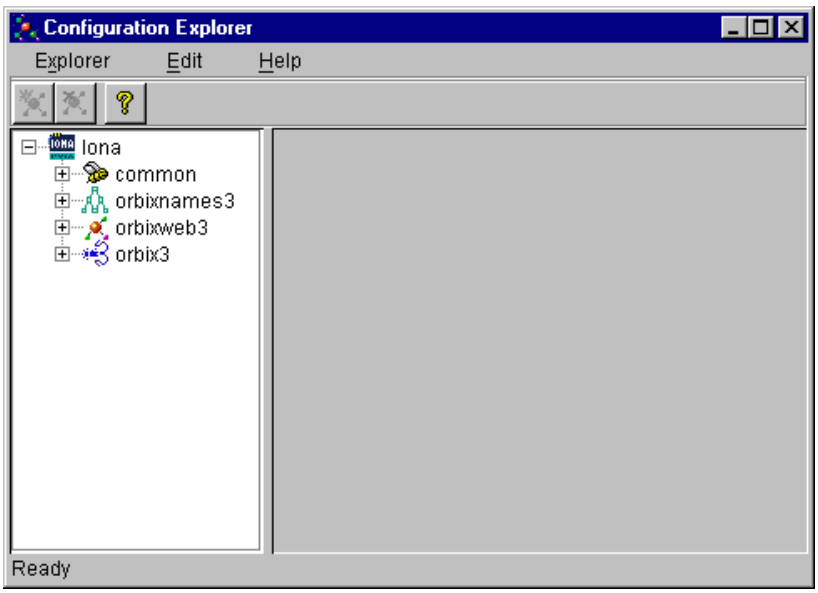

Figure 3: *Orbix Java Configuration Explorer*

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 textbox*.

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

  The *textpane* control contains a **Name** column and a **Value** column as shown in Figure 4. Each row corresponds to individual configuration file entries. The text pane enables you to view and modify these entries.

At startup, the Orbix Java Configuration Explorer opens the `iona.cfg` root configuration file. By default, this file is located in the `config` directory of your Orbix Java installation. The Configuration Explorer navigation tree displays icons that represent the configuration files included in `iona.cfg` as shown in Figure 3.

Configuring Common Settings

To configure settings that are common to multiple Micro Focus CORBA 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 4 on page 38.

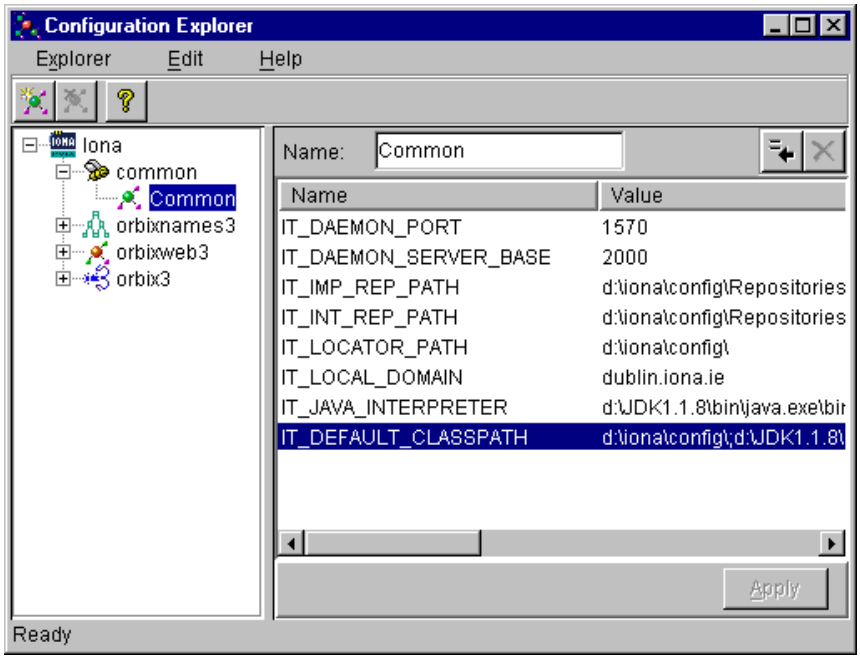

Figure 4: *Common Configuration Settings*

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

IT_DAEMON_PORT	The TCP port number on which the Orbix Java 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 Java Implementation Repository directory.
IT_INT_REP_PATH	The full path name of the Orbix Java 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 Java 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 Java-Specific Settings

To configure settings that apply to Orbix Java only, select the **Orbix Java** icon in the navigation tree. This icon represents the `OrbixWeb` configuration scope in the file `orbixweb3.cfg`. The `OrbixWeb` variables stored in the default `orbixweb3.cfg` configuration file appear in the text pane, as shown in Figure 5.

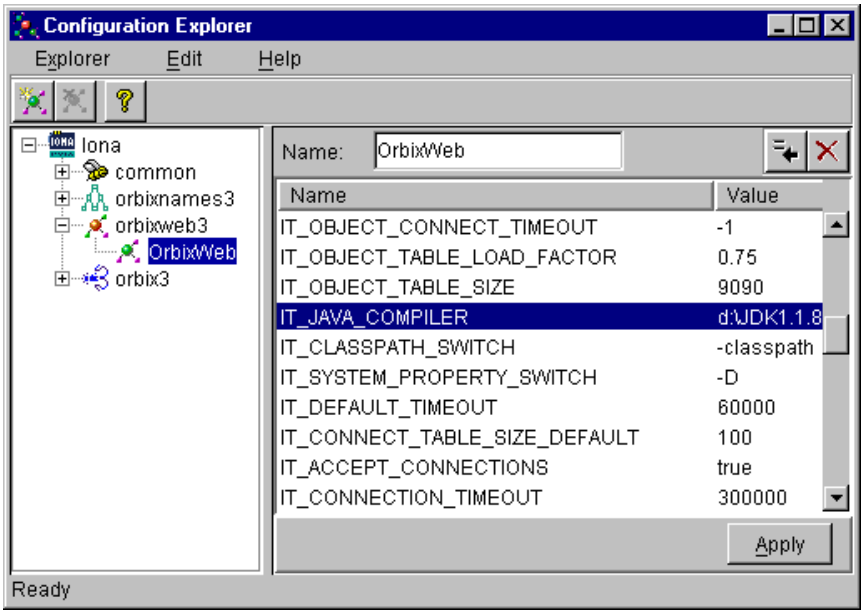

Figure 5: *Configuring Orbix Java-Specific Settings*

For example, the **Orbix Java** configuration settings include the following:

`IT_JAVA_COMPILER`	The path to the Java compiler executable.
`IT_CLASSPATH_SWITCH`	The switch used by the Java interpreter to specify a classpath.

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 configuration variables specific to other Orbix Java components by following these steps. Refer to the ***OrbixNames Programmer's and Administrator's Guide*** for details of configuration variables that are specific to OrbixNames.

Customizing Your Configuration

By default, the Orbix Java Configuration Explorer displays the configuration variables contained in the default configuration files. You can use the Orbix Java 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 6.

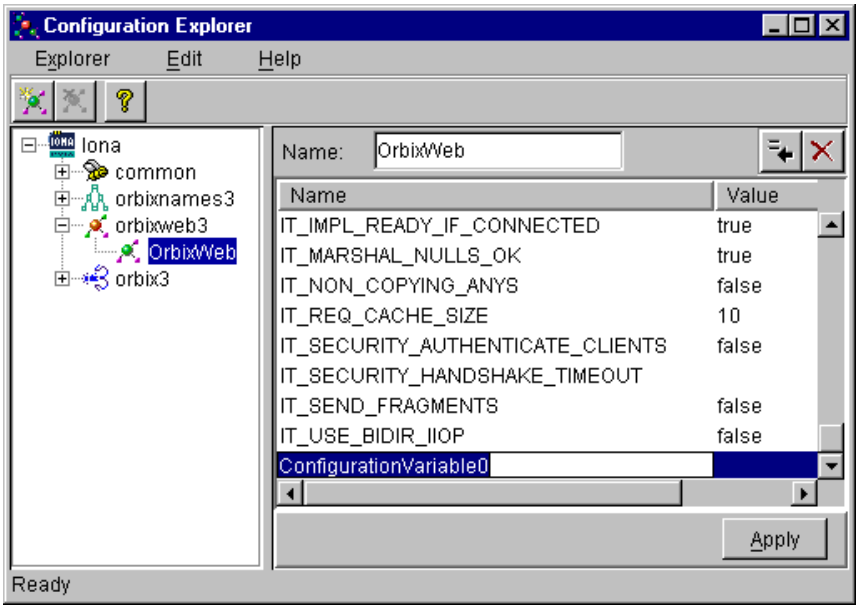

Figure 6: *Creating Configuration Variables*

To create a configuration variable, perform the following steps:

1. Select the **Create Configuration Variable** button, shown in Figure 7 on page 42.
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 7: *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 use 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 7.
3. Select the **Apply** button to save your setting to the appropriate configuration file.

Refer to "Orbix Java Configuration Variables" for a complete list of both common and Orbix Java-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 "Using Orbix Java Configuration Files" for more details.

In the navigation tree, user-defined configuration scopes are displayed as branching from default configuration scope icons, as shown in Figure 8 on page 43.

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 41.

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.

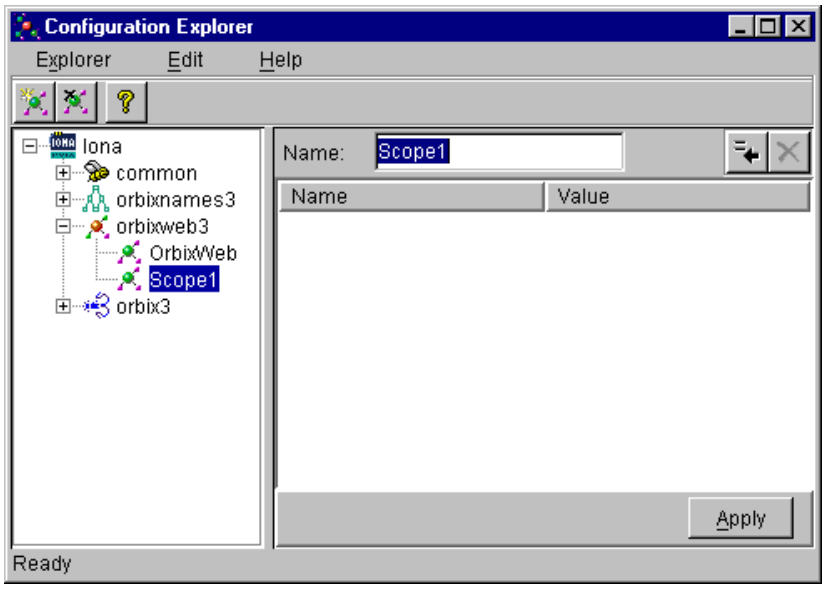

Figure 8: *Creating Configuration Scopes*

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 41 and "Creating Configuration Scopes" on page 43.

The Orbix Java Server Manager

The Implementation Repository is the component of Orbix Java that maintains registration information about servers and controls their activation. The Orbix Java 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 Java system, the Orbix Java 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 Java clients.

The Orbix Java 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 Java Edition*** describes the Implementation Repository in detail. This chapter assumes that you are familiar with this description.

Starting the Orbix Java Server Manager

To start the Orbix Java Server Manager, choose the **Server Manager** option in the Orbix Java menu. Alternatively, enter `srvmgr` at the command line.

The main Server Manager window appears as shown in Figure 9.

Figure 9: *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** dialog box appears, as shown in Figure 10.

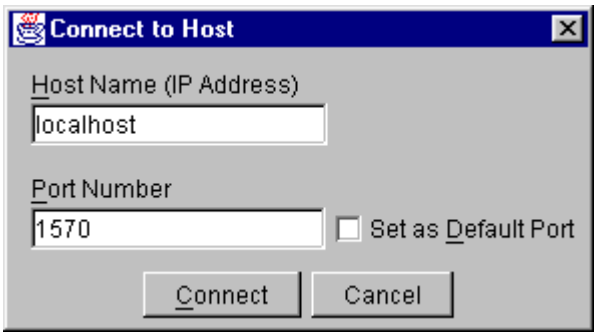

Figure 10: *Connect Dialog Box*

2. In the **Host Name** text box, type the name or IP address of the host on which the required Orbix Java daemon runs. The default is the local host.
3. In the **Port Number** text box, type the TCP/IP port number on which the Orbix Java daemon runs. To make a port number the default, click the **Set as Default Port** check box. The default port number is initially set to 1570.
4. Click **Connect**.

   The main Server Manager window then displays the contents of the Implementation Repository. For example, Figure 11 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**.

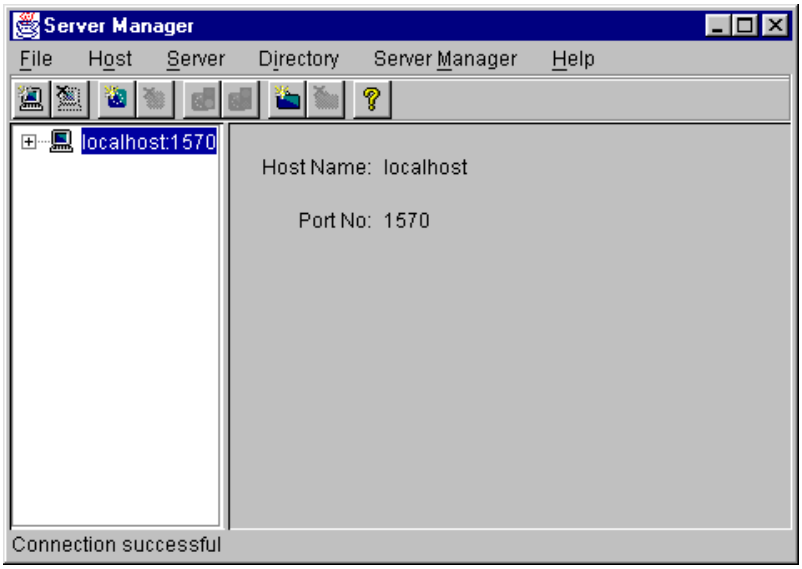

Figure 11: *Connection to an Implementation Repository*

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 12 on page 48.
3. Type the name of the new directory in the **Directory Name** text box.
4. Click **Apply**.

   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 clicked. This is shown in Figure 12 on page 48.

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

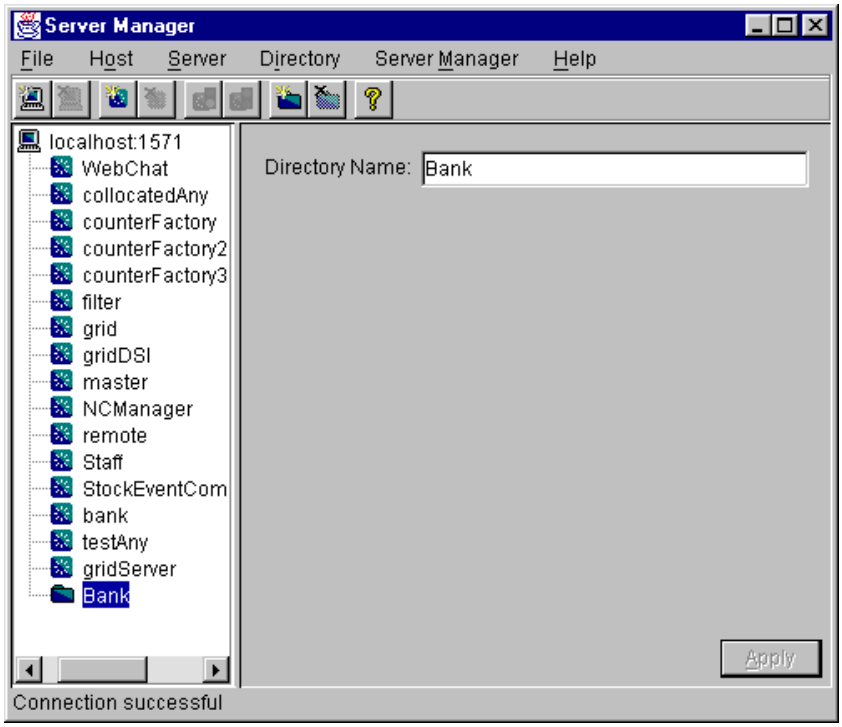

Figure 12: *Creating a New Directory*

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 13. 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 Orbix Java server, click the **Orbix Java 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, click the **Rights** tab. The **Rights** tab is described in "Providing Server Access Rights to Users" on page 49.
6. The default server primary activation mode is shared. The default secondary activation mode is normal.

   To modify the server activation details, click the **Activation** tab. The **Activation** tab is described in "Specifying Server

Activation Details” on page 51.

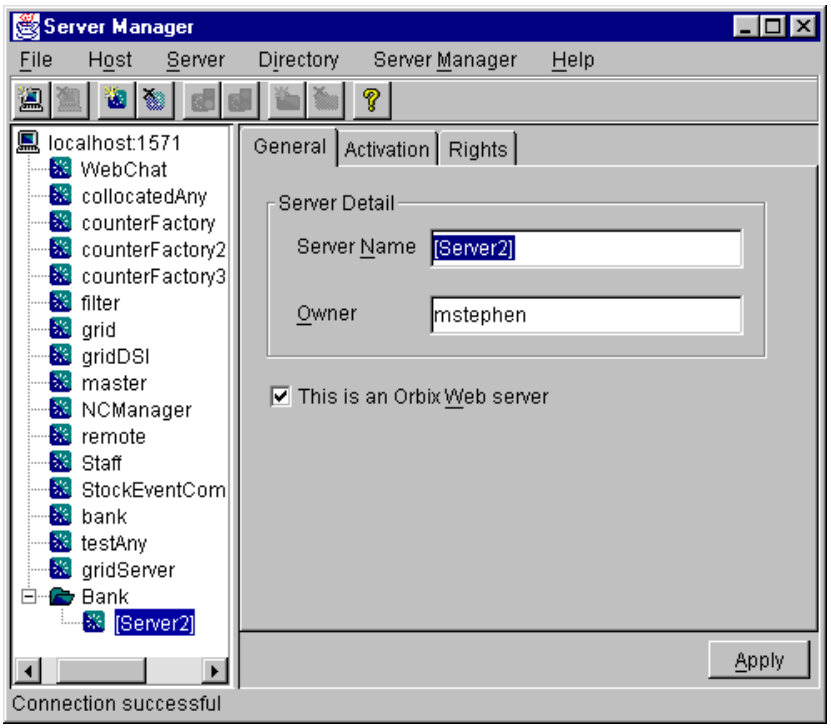

Figure 13: *Registering a New Server*

Providing Server Access Rights to Users

During server registration, you can provide server access rights to other users by clicking the **Rights** tab in the main window. The **Rights** tab appears as shown in Figure 14 on page 50.

Orbix Java offers two types of access rights:

- Launch rights
- Invoke rights

Launch rights allow clients owned by a specified user to cause the Orbix Java 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, type the user identifier in the text box. To grant these rights to all users, type the user name `all`.
2. Click **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. Click **Remove**.

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

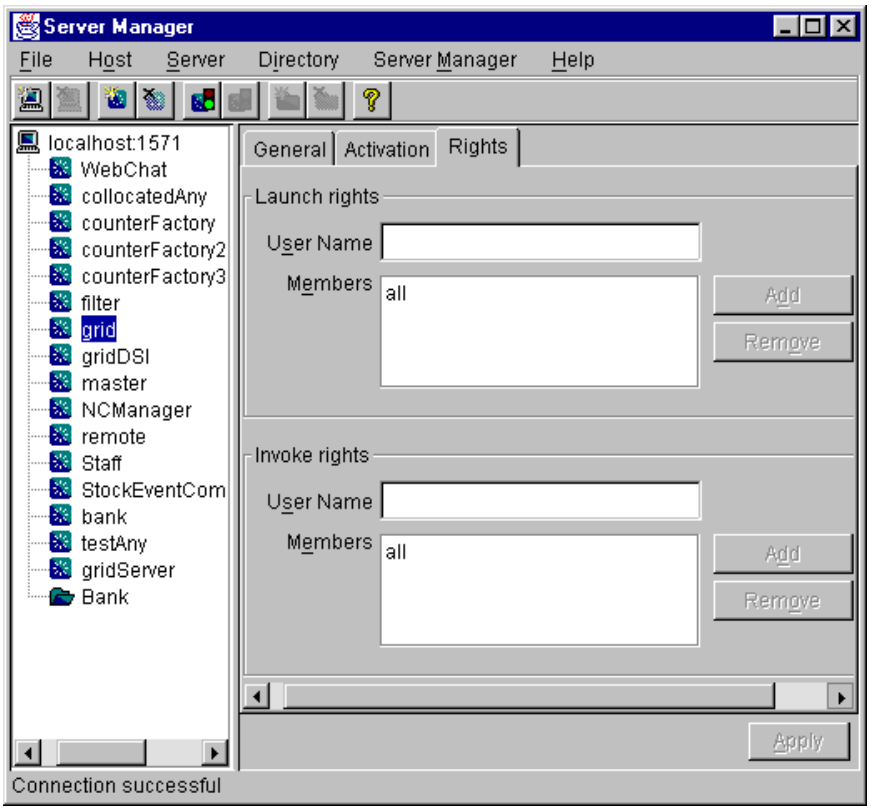

Figure 14: *Providing Server Access Rights*

Specifying Server Activation Details

During server registration, you can specify the server activation details by clicking the **Activation** tab in the Server Manager main window. The **Activation** tab appears as shown in Figure 15.

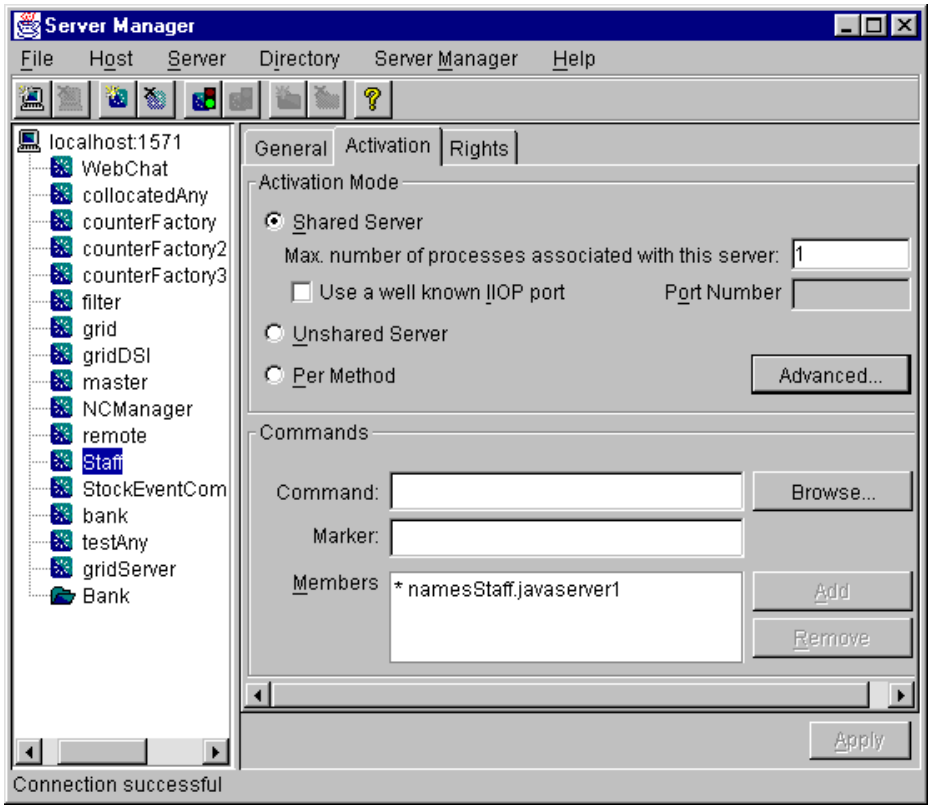

Figure 15: *Specifying Server Activation Details*

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* click the **Advanced** button in the **Activation Mode** section. This launches the **Secondary Activation Modes** dialog box, as shown in Figure 16. The default secondary activation mode is normal.

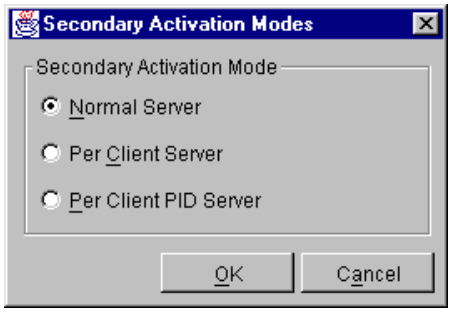

Figure 16: *Secondary Activation Modes*

A server registered in shared activation mode can have an associated maximum number of processes. The Orbix Java 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, click 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, click **OK** to confirm these details.

Note: The Orbix Java daemon currently supports shared primary activation mode and normal secondary activation mode only.

Launch Commands

The **Commands** section on the **Activation** tab allows you to modify the launch commands associated with a server. A registered server must have at least one launch command.

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. Click **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. Click **Add**.

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

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. Click **Add**.

Repeat this process for each method you wish to register.

Modifying Server Registration Details

When you register a server, the Orbix Java 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 48.
4. Click the **Apply** button.

Launching a Persistent Server

Orbix Java 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 17.

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

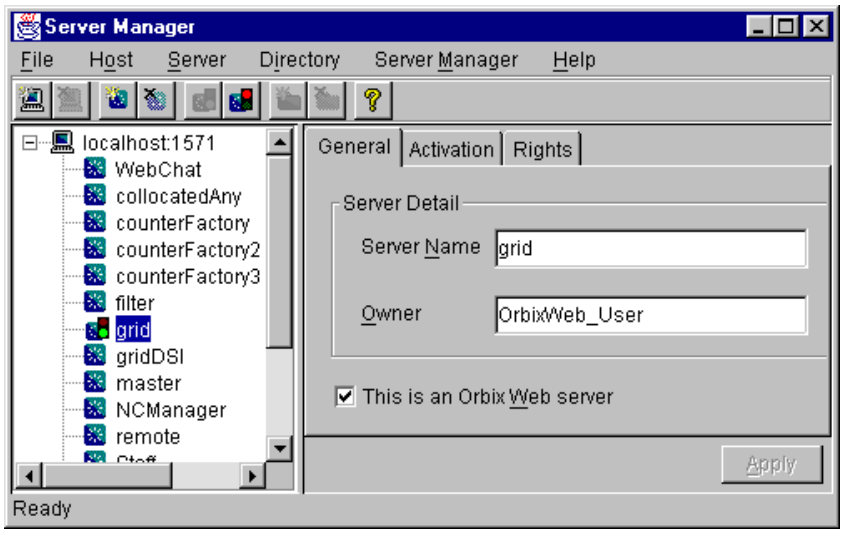

Figure 17: *Launching a Persistent Server*

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 18.

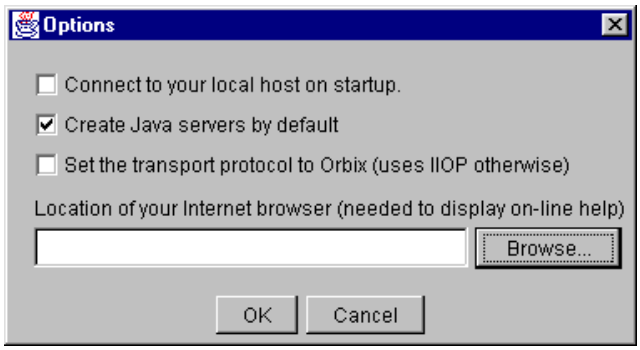

Figure 18: *The Options Dialog Box*

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

   To change this default, check **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, click the check box labelled **Set the transport protocol to Orbix.**
5. To enable online help, enter the **Location of your Internet browser** in the text box provided.
6. Click **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 Server Manager, is no longer necessary.

The Interface Repository Browser

The Interface Repository provides persistent storage of IDL definitions and allows CORBA applications to retrieve information about those definitions at runtime. The Interface Repository Browser allows you to manage IDL definitions in the Interface Repository.

Some CORBA applications, for example applications that use the Dynamic Invocation Interface (DII) to invoke operations, require runtime access to information about IDL definitions. The Interface Repository allows you to store IDL definitions for retrieval by these applications.

The Interface Repository Browser allows you to add IDL definitions to the Interface Repository and view information about those definitions. CORBA applications can retrieve information about those definitions using standard IDL interfaces implemented by the Interface Repository.

The Interface Repository Browser also allows you to export IDL definitions from the Interface Repository to a file. This feature makes the Interface Repository Browser a useful development tool for managing the availability of IDL definitions in your system.

The ***Orbix Programmer's Guide Java Edition*** describes the Interface Repository in detail. The remainder of this chapter assumes that you are familiar with this description.

Starting the Interface Repository Browser

You can start the Interface Repository Browser from the Windows Start menu. Alternatively, enter the `orbixifr` command at the command line.

The main Interface Repository Browser window appears as shown in Figure 19.

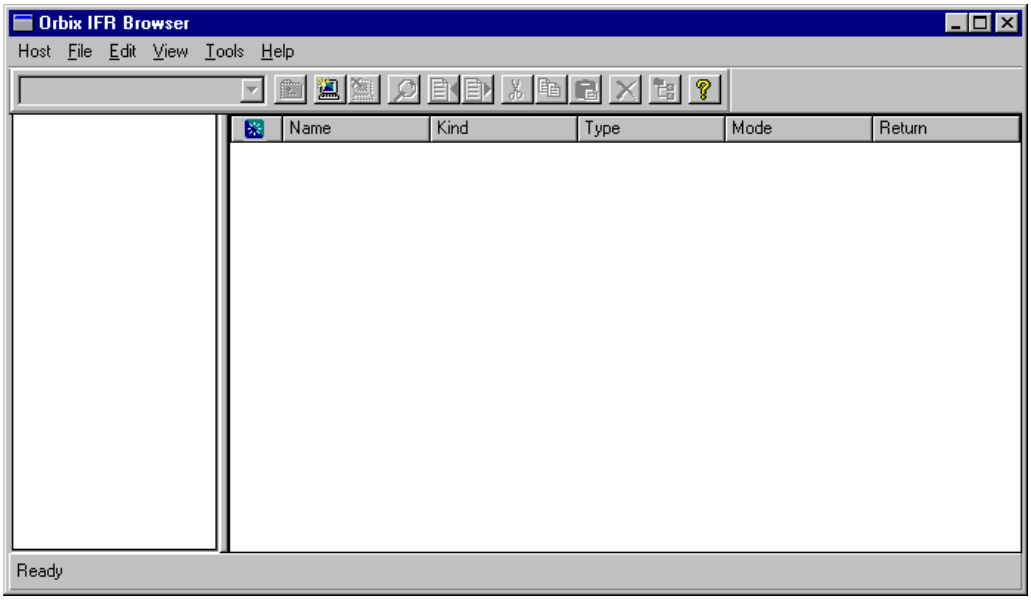

Figure 19: *The Main Interface Repository Browser Window*

The browser interface includes the following elements:

- *A menu bar*.
- *A tool bar*.
- *A navigation tree*. This tree displays a graphical representation of the contents of an Implementation Repository.
- *A multi-columned list box*. This list box displays information about IDL definitions selected in the navigation tree.
- *A status bar*.

Note: You can use the tool bar icons in place of the menu options described in this chapter.

Connecting to an Interface Repository

The Interface Repository is implemented as an Orbix server. The ***Orbix Programmer's Guide Java Edition*** describes how you make an Interface Repository server available to your system.

To connect to an Interface Repository server, do the following:

1. Select **Host/Connect**. The **Connect** dialog box appears as shown in Figure 20.

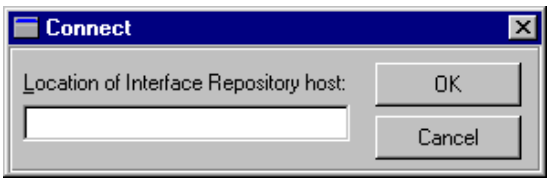

Figure 20: *The Connect Dialog Box*

2. In the text box, enter the name or IP address of the host on which the Interface Repository server runs.
3. Click **OK**. The navigation tree in the main browser window displays the contents of the Interface Repository.

Adding IDL to the Interface Repository

The Interface Repository Browser allows you to import IDL definitions from a source file. This is a safe mechanism for adding IDL definitions to the Interface Repository which maintains the Interface Repository in a consistent state.

To add IDL definitions to the Interface Repository, do the following:

1. Select **File/Import**. The standard **Open File** dialog box for your operating system appears.
2. In the dialog box, enter the name of the source file in which your IDL is defined.
3. Click **OK**. In the main browser window, the navigation tree control displays the contents of the Interface Repository including the new IDL definitions.

Consider the following example IDL source file:

```
// IDL
interface Grid {
    readonly attribute short height;
    readonly attribute short width;

    long get (in short row, in short col);
    void set (in short row, in short col, in long value);
};
```

If you import this file into an empty Interface Repository, the main browser window appears as shown in Figure 21 on page 57.

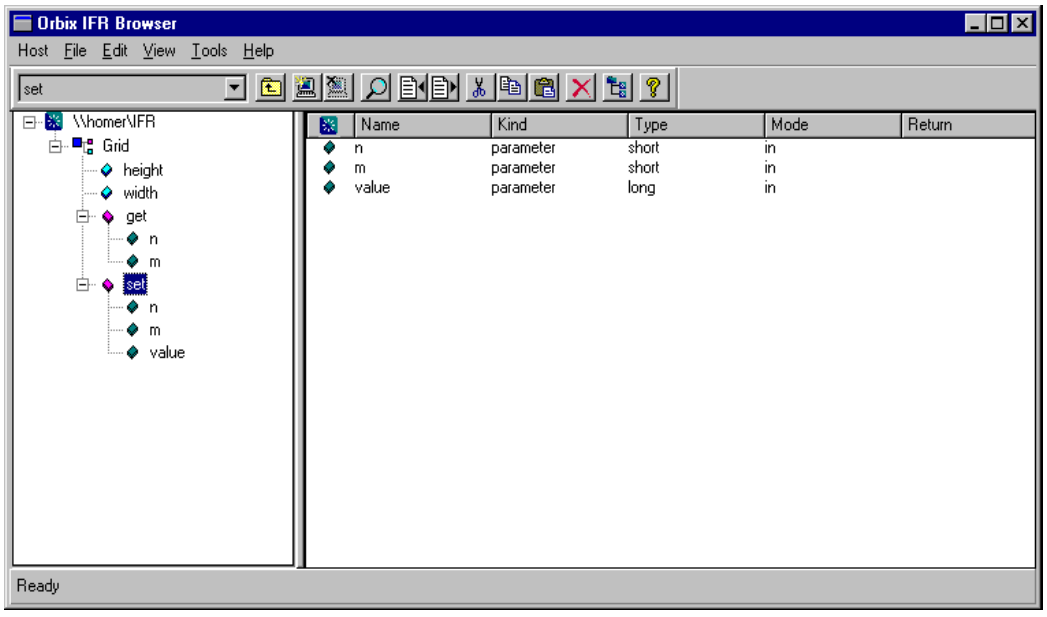

Figure 21: *IDL Definitions in the Interface Repository Browser*

Viewing the Interface Repository Contents

The navigation tree in the main browser window represents the contents of the Interface Repository in terms of containment relationships. As described in the ***Orbix Programmer's Guide Java Edition*** , the Interface Repository uses containment relationships to represent the nested structure of IDL definitions.

Consider the following example IDL source file:

```
// IDL
    module Finance {
    interface Account {
    readonly attribute float balance;
    void makeDeposit (in float amount);
    void makeWithdrawal (in float amount);
    };
    interface Bank {
    Account newAccount ();
    };
};
```

If you import this file into an Interface Repository, the browser navigation tree illustrates that the definition of module `Finance` contains interfaces `Account` and `Bank` which in turn contain attribute and operation definitions, as shown in Figure 22.

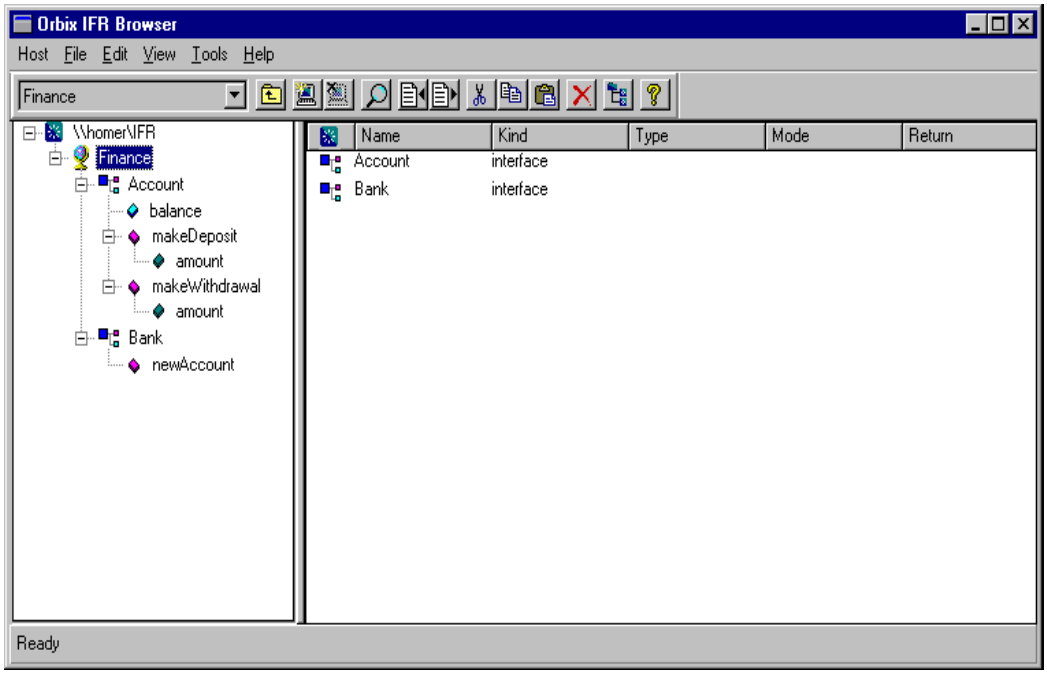

Figure 22: *Containment Relationships in the Interface Repository Browser*

Viewing Information about IDL Definitions

The list box in the main browser window displays information about selected IDL definitions. To view information about an IDL definition, select the navigation tree icon of the container in which the definition is contained. The list box displays information about the contents of the container, including the type and name of each contained definition.

For example, if you select the icon for module `Finance`, the list box displays information about the IDL interface definitions contained within this module, as shown in Figure 22.

Viewing Source Code for IDL Definitions

To view the source for an IDL definition, do the following:

1. Navigate to the required IDL definition.
2. Select **View/View CORBA IDL**. The **View Interface Definition Language** dialog box displays the IDL source associated with the selected definition.

For example, if you view the source for interface `Bank`, the **View Interface Definition Language** dialog box appears as shown in Figure 23.

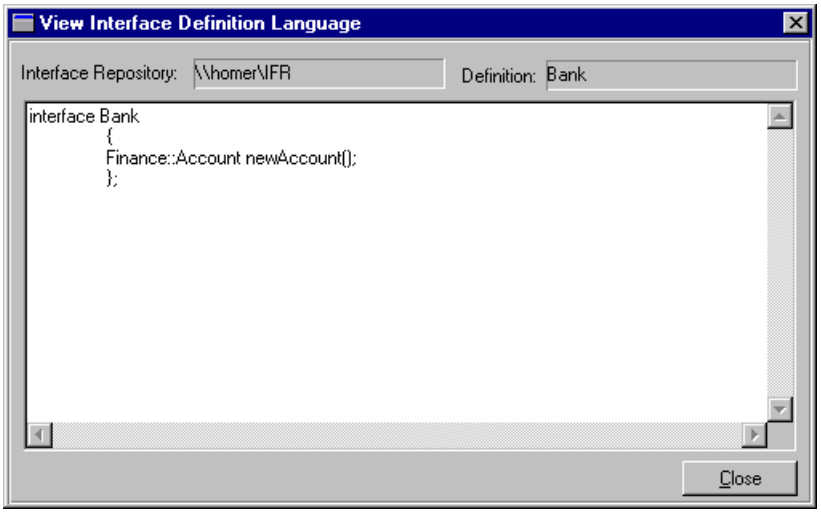

Figure 23: *The View Interface Definition Language Dialog Box*

Exporting IDL Definitions to a File

The Interface Repository Browser allows you to save an IDL definition to a file. To export an IDL definition from the Interface Repository to a file, do the following:

1. Navigate to the required IDL definition.
2. Select **File/Export**. The standard **Save File As** dialog box for your operating system appears.
3. In the dialog box, enter the name of the target file in which you wish to save the IDL definition.
4. Click **OK** to save the definition to the specified file.

Configuring the Interface Repository Browser

To configure the Interface Repository Browser, do the following:

1. Select **Network/Options**. The **Interface Repository Options** dialog box appears as shown in Figure 24.

Figure 24: *The Interface Repository Options Dialog Box*

2. By default, the main browser window refreshes every seven seconds. To modify this refresh time, enter a positive integer value in the **Refresh Time** text box.
3. By default, the browser does not connect to an Interface Repository at startup. To specify that the browser should connect to the Interface Repository at the local host, click the **Connect to local host on startup** button.
4. Click **OK** to commit the new configuration.

Note that you can manually refresh the main browser window at any time. To do this, select **View/Refresh**.

Part III

Appendices

In this part

This part contains the following:

Orbix Java Configuration Variables

There are two types of Orbix Java configuration variables: those that are common to multiple Micro Focus CORBA products, and variables that are specific to Orbix Java 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.

Table 1: *Common Configuration variables*

Variable	Type	Description
`IT_DAEMON_PORT`	Integer	TCP port number for the Orbix Java daemon.
`IT_DAEMON_SERVER_BASE`	Integer	A server that is launched in separate processes listens on its own port. The first server port assigned is `IT_DAEMON_SERVER_BASE` plus 1, subsequently allocated ports increment until `IT_DAEMON_SERVER_BASE` plus `IT_DAEMON_SERVER_RANGE`.
`IT_DAEMON_SERVER_RANGE`	Integer	Refer to the entry for `IT_DAEMON_SERVER_BASE`. The default value is `2000`.
`IT_DEFAULT_CLASSPATH`	String	This is the classpath the daemon uses to find Java servers when launching them. You can supplement this on a per-server basis using the `-addpath` variable to `putitj`. There is no default.
`IT_IMP_REP_PATH`	String	The full path name of the Implementation Repository directory.
`IT_INT_REP_PATH`	String	The full path name of the Interface Repository directory.

Table 1: *Common Configuration variables*

Variable	Type	Description
IT_JAVA_INTERPRETER	String	The path to the Java interpreter executable. Used by the "owjava" tool when starting servers or other Java applications. Also used by the Orbix Java daemon when starting servers.
IT_LOCAL_DOMAIN	String	The name of the local Internet domain; for example, microfocus.com.

Orbix Java-Specific Configuration variables

You can set these variables using the Configuration Explorer GUI tool, or by editing the `orbixweb3.cfg` configuration file.

The available configuration variables are listed here in alphabetical order. Infrequently-used variables are marked with an asterisk (*); these generally do not need to be changed.

Table 2: *Orbix Java-Specific Configuration Variables*

Variable	Type	Description
`IT_ACCEPT_CONNECTIONS`	Boolean	Allow connections to be opened from remote ORBs so that operations can be called on this ORB's objects. The default value is `true`. (*)
`IT_ALWAYS_CHECK_LOCAL_OBJS`	Boolean	A `true` value here indicates that when an object reference arrives, always check to see if this is a reference for a local object. The default value is `false`. (*)
`IT_ANY_BUFFER_SIZE`	Integer	The initial size of the internal buffer used for marshalling `anys`. The default value is `512`. (*)
`IT_BIND_IIOP_VERSION`	String	This controls the IOR (Interoperable Object Reference) version used in `bind()` calls. Orbix Java supplies a separate version control for `bind()` calls because they create their own IORs, and do not return IORs created by servers. This defaults to `10` (version 1.0). You should only set this to `11` if you are sure that the target server supports IIOP 1.1.
`IT_BIND_USING_IIOP`	Boolean	Use the IIOP protocol to `bind()` instead of the Orbix protocol. The default is `true`.
`IT_BUFFER_SIZE`	Integer	The initial size of the internal buffer used for marshalling operation variables. The default value is `8192`. (*)
`IT_CLASSPATH_SWITCH`	String	The switch used by the Java interpreter to specify a classpath. Used by the `owjava` tool when starting servers or other Java applications. This defaults to `-classpath`. (*)
`IT_CONNECT_ATTEMPTS`	Integer	The maximum number of retries Orbix Java makes to connect a client to a server. The default value is `10`. (*)

Table 2: *Orbix Java-Specific Configuration Variables*

Variable	Type	Description
IT_CONNECTION_ORDER	String	Specifies the order in which clients try different connect mechanisms to servers. You can specify direct, iiopproxy or http. If SSL is enabled, the SSL version of the connection mechanism is used. The default is iiopproxy.
IT_CONNECTION_TABLE_PER_THREAD	Boolean	This variable allows you to specify a connection table for each thread as opposed to for each ORB.This prevents multi threaded HTTP connections from being locked. This setting is independent of the IT_MULTI_THREADED_SERVER. variable. You must set both to true for multi threaded HTTP to work. The default is false.
IT_CONNECTION_TIMEOUT	Integer	The time (in milliseconds) an existing connection from client to server is kept alive to be used for further invocations. The default is 300000. (*)
IT_CONNECT_TABLE_SIZE_DEFAULT	Integer	The initial size of the connection table. This is resized automatically. This defaults to 100. (*)
IT_DETECT_APPLET_SANDBOX	Boolean	If set to true, always try to detect whether the ORB is being used in an applet. If the applet sandbox is detected, do not perform operations that cause a SecurityException, such as accessing system properties. The default value is true. (*)
IT_DEFAULT_IIOP_VERSION	String	This controls the IIOP version embedded in IORs produced in Orbix Java servers. It indicates what versions of IIOP the target supports, and also the version of messages sent by a client (as long as it is less than or equal to that of the target). Set to 10 (IIOP version 1.0) by default. You must set this to 11 in servers to allow clients to use IIOP fragmentation.
IT_DII_COPY_ARGS	Boolean	Whether the DII should copy invocation arguments. Set this to false to optimize stub marshalling for large messages. This defaults to false. (*)

Table 2: *Orbix Java-Specific Configuration Variables*

Variable	Type	Description
IT_DSI_COPY_ARGS	Boolean	Whether the DSI should copy invocation arguments. The default value is false. (*)
IT_HTTP_TUNNEL_HOST	String	The TCP/IP hostname used by a client to contact a Wonderwall IIOP proxy for HTTP tunnelling.
IT_HTTP_TUNNEL_PORT	Integer	The TCP/IP port used by a client to contact a Wonderwall IIOP proxy for HTTP tunnelling. This defaults to 0.
IT_HTTP_TUNNEL_PREFERRED	Boolean	Whether HTTP tunnelling should be used in preference to any other connection mechanism. This defaults to false.
IT_HTTP_TUNNEL_PROTO	String	The HTTP protocol used by a client to contact a Wonderwall IIOP proxy for HTTP tunnelling (usually http).
IT_IIOP_LISTEN_PORT	Integer	A server's well-known port; the port to listen for client invocations using IIOP. The default value is 0. (*)
IT_IIOP_PROXY_HOST	String	The TCP/IP hostname used by a client to contact a Wonderwall IIOP proxy for IIOP proxy connections.
IT_IIOP_PROXY_PORT	Integer	The TCP/IP port used by a client to contact a Wonderwall IIOP proxy for IIOP proxy connections. This has a default value of 0.
IT_IIOP_PROXY_PREFERRED	Boolean	Indicates whether connecting using IIOP proxying via a Wonderwall should be used in preference to any other connection mechanism. This defaults to false.
IT_IMPL_READY_IF_CONNECTED	Boolean	Specifies whether the Orbix Java runtime should inform the daemon that the server is ready by calling impl_is_ready() when the server calls ORB.connect(). This defaults to true.

Table 2: *Orbix Java-Specific Configuration Variables*

Variable	Type	Description
IT_IMPL_IS_READY_TIMEOUT	Integer	When an in-process server is launched, the Java daemon waits to be informed that the server is active before allowing the causative client request to proceed. Refer to the *Orbix Programmer's Guide Java Edition* for further details. It waits a maximum of this amount of time, specified in milliseconds. The default is 30000 milliseconds (30 seconds).
IT_INITIAL_REFERENCES	String	A list of IORs for initial service objects, as returned by the ORB operation list_initial_references(). It is specified in a "name value name value..." format. For example, "NameService IOR:[IOR_for_naming_service] TradingService IOR:[IOR_for_Trader]".
IT_IORS_USE_DNS	Boolean	Indicates whether IIOP object references use DNS hostnames or IP addresses. A true value here indicates that they should use DNS hostnames. This defaults to false. (*)
IT_JAVA_COMPILER	String	The path to the Java compiler executable. Used by the owjavac tool when building the Orbix Java demos.
IT_JVM_SYSTEM_PROPERTY_SWITCH	String	This allows the Java daemon to be run on different JVMs. It facilitates the different switches that different Java Interpreters support to pass system properties to the JVM. The default is -D for the JDK. You should set this to /d: for Microsoft's JView.
IT_KEEP_ALIVE_FORWARDER_CONN	Boolean	Whether the connection from the client to the Orbix Java daemon should be kept alive after a bind() call. The default is true. (*)
IT_LISTENER_PRIORITY	Integer	The priority of the server-side connection-listener thread. The default value is 5. (*)
IT_LOCAL_DOMAIN	String	The name of the local DNS domain.
IT_LOCAL_HOSTNAME	String	The name of the local host. You do not need to set this normally, but it can be useful if you wish to control the interface on which incoming connections are accepted.

Table 2: *Orbix Java-Specific Configuration Variables*

Variable	Type	Description
IT_MARSHAL_NULLS_OK	Boolean	Allow Java nulls to be used to represent null IDL strings and anys. This variable enables API compatibility with pre-OMG standard versions of Orbix Java and Orbix C++. The default is false.
IT_MULTI_THREADED_SERVER	Boolean	Whether this instance of the Java runtime can contain multiple servers in the one process. This defaults to false. (*)
IT_NAMES_HASH_TABLE_LOAD_FACTOR	Float	Percentage of table elements used before a resize. The default value is 0.5.
IT_NAMES_HASH_TABLE_SIZE	Integer	The initial size for the Naming Service hash table. This value must be a prime number. The default value is 23.
IT_NAMES_REPOSITORY_PATH	String	This represents the default location of the Naming Service repository entries. This is set to the following directory by default: <install dir>/config/NamesRep
IT_NAMES_SERVER	String	The name of the Name Server that is registered with the Implementation Repository.
IT_NAMES_TIMEOUT	Integer	The default timeout, set to the following: -1(IT-INFINITE_TIMEOUT)
IT_NAMES_SERVER_HOST	String	The TCP/IP hostname of the host where the CORBA Naming Service is installed.
IT_NS_IP_ADDR	String	The IP address of the host where the CORBA Naming Service is installed. If this is not set, the IT_NAMES_SERVER_HOST variable is used instead. (*)
IT_NS_PORT	Integer	The TCP/IP port of the host running the CORBA Naming Service. The default value is 1570.
IT_OBJECT_CONNECT_TIMEOUT	Integer	The amount of time an object is available after connect() is called. The default value of -1 means indefinitely. (*)
IT_OBJECT_TABLE_LOAD_FACTOR	Float	The load factor of the server object table. Once this proportion of objects has been registered, it is resized. This has a default of 0.75. (*)

Table 2: *Orbix Java-Specific Configuration Variables*

Variable	Type	Description
IT_OBJECT_TABLE_SIZE	Integer	The initial size of the internal table used to register Orbix Java objects in a server. The default value is 1789. (*)
IT_ORBIXD_IIOP_PORT	Integer	The TCP/IP port number on which the Orbix Java daemon can be contacted when using IIOP. Provided to support legacy daemons requiring a separate port for each protocol. The default is 1570.
IT_ORBIXD_PORT	Integer	The TCP/IP port number on which the Orbix Java daemon should be contacted when using the Orbix protocol. The default is 1570.
IT_READER_PRIORITY	Integer	The priority of the server-side request-reader thread. The default is 3. (*)
IT_REQ_CACHE_SIZE	Integer	The initial size of the internal cache for outgoing requests. The default is 10. (*)
IT_SEND_FRAGMENTS	Boolean	If this is set to true and the target server supports IIOP version 1.1 or higher, messages that exceed IT_BUFFER_SIZE are sent as fragments. This defaults to false.
IT_TRADING_SERVER	String	The server name for the CORBA Trader service. (*)
IT_USE_ALIAS_TYPECODE	Boolean	When set to true creates an alias TypeCode. This defaults to false.
IT_USE_BIDIR_IIOP	Boolean	Whether bidirectional IIOP connections should be used to support callbacks through firewalls. This is set to false by default.
IT_USE_EXTENDED_CAPABILITIES	Boolean	Orbix Java provides built-in support for Netscape's Capabilities API. If this is enabled, connections can be opened to any host using IIOP, Orbix protocol or SSL-IIOP, when a valid Netscape Object Signing certificate is used. This is set to true by default. (*)

Table 2: *Orbix Java-Specific Configuration Variables*

Variable	Type	Description
IT_USE_ORBIX_COMP_OBJREF	Boolean	When this is set to false, the default TypeCode alias is used for object references. This is IDL:CORBA/Object:1.0 When this is set to true, the following TypeCode alias is used for object references: IDL:omg.org/CORBA/Object:1.0 The default is false.
IT_USE_ORB_THREADGROUP	Boolean	When set to true, this causes Orbix Java to place any threads it creates into an "ORB threadgroup", a top-level thread-group. This allows ORB threads to be separated from application threads, and is especially useful in Netscape-signed applets. In the JVM, multiple instances of the same applet sharing the same ORB object can interfere with each others operation. This is set to true by default. (*)
config	String	The configuration file to use. By default, the first configuration file found in the classpath, or the first found in the CODEBASE directory for applets is used.
pingDuringBind	Boolean	Whether a client should try to ping the server during a bind() call. This is set to true by default. (*)
setDiagnostics	Integer	Specifies the Orbix Java diagnostics level output to stdout. You should enter a value in the range 0–255. The default value is 1.
useDefaults	Boolean	If this is set to true, Orbix Java does not output a warning if the configuration file cannot be found.
IT_USE_TRUE_PROCESS_PID	Boolean	Specifies whether an Orbix server will use a JNI library to figure out the true PID of itself. The default value is false.
IT_KEYOBJECTTABLE_USINGPORT	Boolean	Whether Orbix Java should take the hostname and port into consideration when adding servants into the runtime object table. If this is set to false, Orbix uses the object key only. The default value is false.
IT_CALLBACK_PORT_BASE	Integer	Sets the base port to start assigning port numbers for callbacks. Setting to 0 will let the kernel assign the callback ports. The default value is 0.

Table 2: *Orbix Java-Specific Configuration Variables*

Variable	Type	Description
`IT_CALLBACK_PORT_RANGE`	Integer	Sets the port range to start assigning port numbers for callbacks. The default value is `1`.
`IT_ENABLE_IPV6`	Boolean	Enable IPv6 communication. This enable both IPv4 and IPv6 communication. The default value is `false`.

Note: The entries in Orbix configuration files are scoped with a prefix; for example, `Common{...}` or `OrbixWeb{...}`.

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

Orbix Java Daemon Options

Orbixd Options

The Orbix Java 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 pathname is specified, the file is placed in a directory relative to that from which the daemon is launched.
`-i` *filename*	Outputs the daemon's interoperable object reference (IOR) to the specified file. Unless an absolute pathname is specified, the file is placed in a directory relative to that from which the daemon is launched.
`-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` *seconds*	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` *seconds*	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.
`-?`	Displays the switches to `orbixd`.

Orbixdj Options

The Orbix Java daemon process, `orbixdj`, takes the following options:

`-inProcess`	By default, the Java daemon activates servers in a separate process. This is termed *out-of-process* activation. If this switch is set, the Java daemon starts servers in a separate thread. This is termed *in-process* activation.
`-textConsole`	By default, the Java daemon launches a GUI console. Adding this switch causes the Java daemon to use the invoking terminal as the console.
`-noProcessRedirect`	By default, the `stdout` and `stderr` streams of servers activated in a separate process are redirected to the Java daemon console. Specifying this switch causes the output streams to be hidden.
`-u`	Allows the use of unregistered persistently-launched servers.
`-V`	Prints a detailed description of the configuration parameters used by the Java daemon on start-up. The Java daemon then exits.
`-v`	Causes the Java daemon to print a summary of the configuration it runs with. The Java daemon then exits.
`-?`	Displays the switches to `orbixdj`.

Orbix Java Command-Line Utilities

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

Utility Summary

The following table shows the available command-line utilities:

Purpose	Utility
Server Registration	`putitj`, `rmitj`
Listing Server Information	`lsitj`, `psitj`, `catitj`
Process Management	`pingitj`, `killitj`
Implementation Repository Directories	`mkdiritj`, `rmdiritj`
Security	`chownitj`, `chmoditj`
Interface Repository Management	`putidl`, `readifr`, `rmidl`
Configuration Information	`dumpconfig`

Table 0.1: *Orbix Java Command-Line Utilities*

This appendix describes each command-line utility in alphabetical order.

Note: To get help on any utility, enter the utility name followed by the `-?` or the `-help` switch. For example, `putitj -?`.

catitj

The `catitj` utility outputs full information about a given Implementation Repository entry.

Syntax

`catitj [-v] [-h` *`host`***`]`** *`server_name`*

Options

`-v` Outputs the utility version information.

`-h` *`host`* Outputs information about an entry on a specific machine.

chmoditj

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

Syntax

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

Options

`-v`	Outputs the utility version information.
`-h` *host*	Modify an entry on a specific host.
`-a`	Specify that a user or group is to be added to an access control list (ACL) for a directory of servers.
`i+`	Add a user or group to the invoke ACL.
`i-`	Remove a user or group from the invoke ACL.
`l+`	Add a user or group to the launch ACL.
`l-`	Remove a user or group from the launch ACL.

By default, only the owner of an Implementation Repository entry can launch or invoke the registered server. However, launch and invoke 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 ACL.

chownitj

The `chownitj` utility makes changes to the ownership of Implementation Repository entries and directories.

Syntax

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

Options

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

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 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 command. Spaces must exist between an option and its argument, and on either side of the `+` or `-` that follows a directory.

Orbix Java 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 utility version information.

killitj

The `killitj` utility kills (stops) a running server process.

Syntax

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

Options

`-v`	Outputs the utility version information.
`-h` *host*	Kills a server on a specific machine.
`-m`	Specifies a marker value to identify a specific object, or set of objects, to which the `killitj` utility applies.

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

The `killitj` utility uses the `SIGTERM` signal. This utility does not remove the entry from the Implementation Repository.

lsitj

The `lsitj` utility lists entries in an Implementation Repository directory.

Syntax

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

Options

`-v`	Outputs the utility version information.
`-h` *host*	Lists entries on a specific host.
`-R`	Recursively lists all subdirectories and entries.

mkdiritj

The `mkdiritj` utility creates a new registration directory.

Syntax

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

Options

`-v`	Outputs the utility version information.
`-h` *host*	Creates a new directory on a specific host.

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

pingitj

The `pingitj` utility tries to contact an Orbix Java daemon to determine if it is running.

Syntax

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

Options

`-v`	Outputs the utility version information.
`-h` *host*	Pings a specific host machine.

psitj

The `psitj` utility outputs a list of server processes known to an Orbix Java daemon.

Syntax

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

Options

`-v`	Outputs the utility version information.
`-h` *host*	Lists server processes on the specified host.

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; for example, `*`.
`Code`	The data encoder used; for example, `cdr`.

Comms	The communications protocol used; for example, tcp.
Port	The port number used by the communications system.
Status	This can be auto, manual or inactive.
Per-Client?	Indicates whether the server is a per-client server.
OS-pid	The operating system process.

putidl

The putidl utility allows you to add a set of IDL definitions to the Interface Repository. This utility 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 utility to succeed.

Syntax

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

Options

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

putitj

The putitj utility creates an entry in the Implementation Repository that represents how Orbix Java can start a server.

Note: The availability of a given putitj switch depends on which Orbix Java daemon is used orbixd or orbixdj. Switches labelled orbixd are not currently supported by the Java daemon orbixdj.

Syntax

putitj [-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***][-l] [-persistent]**
[-nservers | -n *number_of_servers* **]**
serverName **[--** *command_line_parameters* **]**

Options

Executing `putitj` 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 hostname on which to execute the `putitj` command. By default, this utility is executed on the local host.
`-per-client` *(orbixd)*	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` *(orbixd)*	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` *(orbixd)*	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 *command_line*.
`-java`	The `-java` switch indicates that the specified server should be launched via the Java interpreter. You can truncate this switch to `-j`.
`-classpath` *full classpath*	You can only use this switch in conjunction with the `-java` switch. Specifies a full class path to be passed to the Java interpreter when the server is launched. Overrides the default value `IT_DEFAULT_CLASSPATH` in `common.cfg`.
`-addpath` *partial classpath*	You can only use this switch in conjunction with the `-java` switch. Specifies a partial class path to be appended to the default value `IT_DEFAULT_CLASSPATH` when the Orbix Java daemon attempts to launch the server.
`-oc` *ORB_class*	Passes `-Dorg.omg.CORBA.ORBClass=`*ORB_class* to the Java interpreter. You should use this switch with the `-os` switch. For Orbix Java servers, the parameter to this switch should be as follows: `IE.Iona.OrbixWeb.CORBA.ORB.` 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 Orbix Java servers the parameter to this switch should be `IE.Iona.OrbixWeb.CORBA.singletonORB`. This string must be passed to the Java interpreter before the server 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 Orbix Java servers being executed by JDK1.2.

`-l`	Allows you to register pre-Orbix 2.3 servers using the `putitj` command.
`-per method` *(orbixd)*	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* *(orbixd)*	Specifies a well-known port number for a server so that Orbix Java, 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).

`--` *parameters*	Allows the addition of extra command-line parameters to be passed to a server.
	All parameters specified after the `--` switch are ignored by the `putitj` utility and passed to the daemon as the launch command. For example,
	`putitj -j testServer`
	`-- -DOrbixWeb.setDiagnostics=255`
	`packageName.className`

The following options apply to the shared mode:

`-nservers` *number_of_servers* *(orbixd)*	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`. You can truncate the `-nservers` switch to `-n`.
`-persistent` *(orbixd)*	Specifies that the server can only be launched persistently (that is, manually). The server is never automatically launched by Orbix Java. If the `-u` option is passed to the Orbix Java 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 `putitj` applies. Marker names specified using `putitj` cannot contain white space.

The following option applies to the per-method mode:

`-method` *method* *(orbixd)*	Specifies a method name to identify a specific method, or set of methods, to which the `putitj` applies.

Server Activation Modes

Activation modes control how servers are implemented when they become processes of the underlying operating system. The primary activation modes are as follows:

Shared	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. This is the default activation mode.
Unshared	In unshared mode, individual objects are registered with the Implementation Repository, and a process is launched for each object.

Per-Method 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:

- 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. This means 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 are independent from shared, unshared and per-method modes. You can combine these activation modes in an arbitrary manner; for example, you can combine per-client with shared, unshared or with per-method.
- 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 `-`.

Lastly, because each of the characters `*?!^-[]\` is special, in the sense that it is interpreted by the pattern matching algorithm; each 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` utility allows you to view an IDL definition stored in the Interface Repository. This utility 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 utility to succeed.

Syntax

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

Options

`-?`	Displays the allowed options for this command.
`-v`	Outputs the utility version information.
`-h` *host*	Indicates the host at which the Interface Repository server is available.
`-d`	Displays all derived types of an IDL interface.

rmdiritj

The `rmdiritj` utility removes an Implementation Repository registration directory.

Syntax

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

Options

`-v`	Outputs the utility version information.
`-h` *host*	Indicates the host from which the directory is deleted.
`-R`	Recursively deletes the directory, and all the Implementation Repository entries and subdirectories within it.

The `rmdiritj` utility returns an error if it is called without the `-R` option on a registration directory that is not empty.

rmidl

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

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

Syntax

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

Options

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

rmitj

Removes an Implementation Repository entry or modifies an entry.

Syntax

`rmitj [-v] [-h` *host*`]`
`[-marker` *marker* **`| -method`** *method*`]` *server_name*

Options

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

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

You can use pattern matching for markers and methods as described in the `putitj` utility reference on page 79.

System Exceptions

The following tables shows the system exceptions defined by CORBA, and the system exceptions that are specific to Orbix Java.

System Exceptions Defined by CORBA

Table 3: *CORBA System Exceptions*

Exception	Description
BAD_CONTEXT	Error processing context object.
BAD_INV_ORDER	Routine invocations out of order.
BAD_OPERATION	Invalid operation.
BAD_PARAM	An invalid parameter was passed.
Bounds	Bounds exception.
BAD_TYPECODE	Bad TypeCode.
COMM_FAILURE	Communication failure.
DATA_CONVERSION	Data conversion error.
IMP_LIMIT	Violated implementation limit.
INITIALIZE	ORB initialization failure.
INTERNAL	ORB internal error.
INTF_REPOS	Error accessing Interface Repository.
INV_IDENT	Invalid identifier syntax.
INV_FLAG	Invalid flag was specified.
INV_OBJREF	Invalid object reference.
MARSHAL	Request marshalling error.
NO_MEMORY	Dynamic memory allocation failure.
NO_PERMISSION	No permission for attempted operation.
NO_IMPLEMENT	Operation implementation unavailable.
NO_RESOURCES	Insufficient resources for request.
NO_RESPONSE	Response to request not yet available.
OBJ_ADAPTOR	Failure detected by object adaptor.
PERSIST_STORE	Persistent storage failure.
TRANSACTION	Transaction exception.
TRANSIENT	Transient failure—reissue request.

Table 3: *CORBA System Exceptions*

Exception	Description
UNKNOWN	The unknown exception.

System Exceptions Specific to Orbix Java

Table 4: *Orbix Java-Specific System Exceptions*

Orbix Java Exception	Description
FILTER_SUPPRESS	Suppress exception raised in per-object pre-filter.

Index

K

L

M

N

V

X