MICRO FOCUS

Orbix Mainframe 6.3.1

CORBA Administrator's Guide

Micro Focus
The Lawn
22-30 Old Bath Road
Newbury, Berkshire RG14 1QN
UK
https://www.microfocus.com

2021-03-18

Contents

Part 2 Managing an Orbix Environment

List of Figures

List of Tables

Preface

Introduction

Orbix is a software environment for building and integrating distributed object-oriented applications. Orbix provides a full implementation of the Common Object Request Broker Architecture (CORBA) from the Object Management Group (OMG). Orbix is compliant with version 2.6 of the OMG'S CORBA specification. This guide explains how to configure and manage the components of an Orbix environment.

Audience

This guide is aimed at administrators managing Orbix environments, and programmers developing Orbix applications.

Note: Not all features mentioned in this guide are available on z/OS.

Organization

This guide is divided into the following parts:

- Introduction introduces the Orbix environment, and the basic concepts required to understand how it works.
- Managing an Orbix Environment explains how to manage each component of an Orbix environment. It provides task-based information and examples.
- Command Reference provides a comprehensive reference for all Orbix configuration variables and administration commands.
- Appendices explain how to use Orbix components as Windows NT services. They also provide reference information for initialization parameters and environment variables.

Related documentation

Orbix documentation also includes the following related books:

- *Management User's Guide*
- *Deployment Guide*
- *CORBA Programmer's Guide*
- *CORBA Programmer's Reference*
- *CORBA Code Generation Toolkit Guide*

Additional resources

The Knowledge Base contains helpful articles, written by experts, about Orbix Mainframe, and other products:

https://community.microfocus.com/t5/Orbix/ct-p/Orbix

If you need help with Orbix Mainframe or any other products, contact technical support:

https://www.microfocus.com/en-us/support/

Document conventions

This guide uses the following typographical conventions:

`Constant width`

Constant width font in normal text represents commands, portions of code and literal names of items (such as classes, functions, and variables). For example, constant width text might refer to the `itadmin orbname create` command.

Constant width paragraphs represent information displayed on the screen or code examples. For example the following paragraph displays output from the `itadmin orbname list` command:

```
ifr
naming
production.test.testmgr
production.server
```

Italic

Italic words in normal text represent emphasis and new terms (for example, *location domains*).

`Code italic`

Italic words or characters in code and commands represent variable values you must supply; for example, process names in your *`particular`* system:

`itadmin process create` *`process-name`*

Code bold — Code bold font is used to represent values that you must enter at the command line. This is often used in conjunction with constant width font to distinguish between command line input and output. For example:

```
itadmin process list
ifr
naming
my_app
```

The following keying conventions are observed:

No prompt	When a command's format is the same for multiple platforms, a prompt is not 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 or Windows command prompt.
...	Horizontal ellipses in format and syntax descriptions indicate that material has been eliminated to simplify a discussion.
[]	Italicized 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. Individual items can be enclosed in {} (braces) in format and syntax descriptions.

Part 1

Introduction

In this part

This part contains the following chapters:

CHAPTER 1

The Orbix Environment

Orbix is a network software environment that enables programmers to develop and run distributed applications.

Overview

This chapter introduces the main components of an Orbix environment, explains how they interact, and gives an overview of Orbix administration.

In this chapter

This chapter contains the following sections:

Basic CORBA Model

Overview

An Orbix environment is a networked system that makes distributed applications function as if they are running on one machine in a single process space. Orbix relies on several kinds of information, stored in various components in the environment. When the environment is established, programs and Orbix services can automatically store their information in the appropriate components.

To establish and use a proper Orbix environment, administrators and programmers need to know how the Orbix components interact, so that applications can find and use them correctly. This chapter starts with a sample application that requires a minimal Orbix environment. Gradually, more services are added.

The basic model for CORBA applications uses an object request broker, or *ORB*. An ORB handles the transfer of messages from a client program to an object located on a remote network host. The ORB hides the underlying complexity of network communications from the programmer. In the CORBA model, programmers create standard software objects whose member methods can be invoked by client programs located anywhere in the network. A program that contains instances of CORBA objects is known as a *server*.

When a client invokes a member function on a CORBA object, the ORB intercepts the function call. As shown in Figure 1, the ORB redirects the function call across the network to the target object. The ORB then collects results from the function call and returns these to the client.

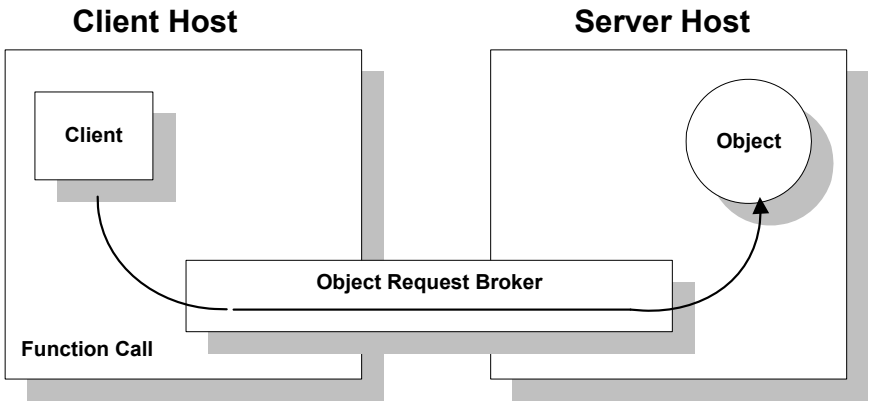

Figure 1: *Basic CORBA Model*

Simple Orbix Application

Overview

A simple Orbix application might contain a client and a server along with one or more objects (see Figure 2). In this model, the client obtains information about the object it seeks, using *object references.* An object reference uniquely identifies a local or remote object instance.

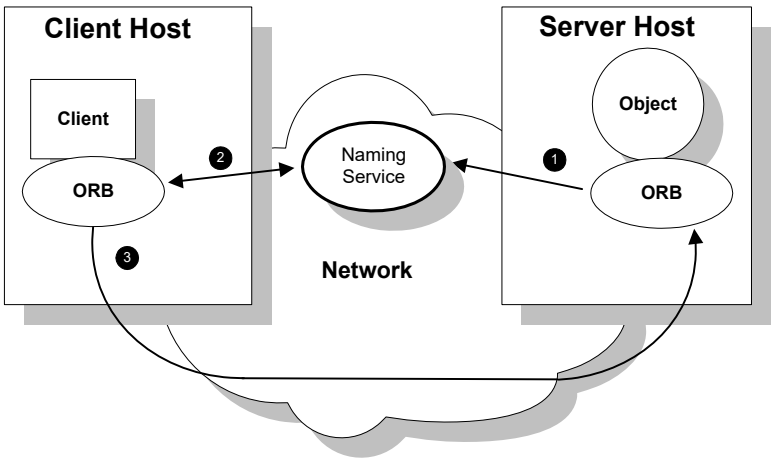

Figure 2: *Overview of a Simple Orbix Application*

How an ORB enables remote invocation

Figure 2 shows how an ORB enables a client to invoke on a remote object:

1. When a server starts, it creates one or more objects and publishes their object references in a *naming service*. A naming service uses simple names to make object references accessible to prospective clients. Servers can also publish object references in a file or a URL.
2. The client program looks up the object reference by name in the naming service. The naming service returns the server's object reference.
3. The client ORB uses the object reference to pass a request to the server object

Portable Object Adapter

Overview

For simplicity, Figure 2 on page 4 omits details that all applications require. For example, Orbix applications use a portable object adapter, or *POA*, to manage access to server objects. A POA maps object references to their concrete implementations on the server, or *servants*. Given a client request for an object, a POA can invoke the referenced object locally.

POA functionality

A POA can divide large sets of objects into smaller, more manageable subsets; it can also group related objects together. For example, in a ticketing application, one POA might handle reservation objects, while another POA handles payment objects.

Figure 3 shows how the POA connects a client to a target object. In this instance, the server has two POAs that each manage a different set of objects.

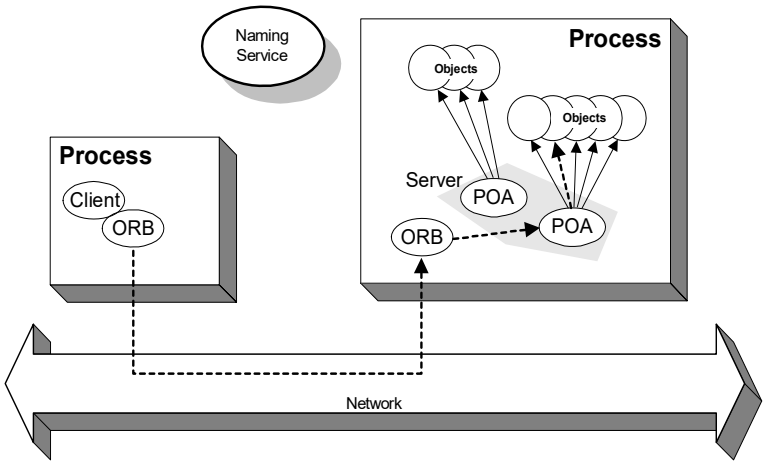

Figure 3: *A POA's Role in Client–Object Communication*

POA names

Servers differentiate between several POAs by assigning them unique names within the application. The object reference published by the server contains the complete or *fully qualified POA name* (*FQPN*) and the object's ID. The client request embeds the POA name and object ID taken from the published object reference. The server then uses the POA name to invoke the correct POA. The POA uses the object ID to invoke the desired object, if it exists on the server.

Limitations of a simple application

This simple model uses a naming service to pass object references to clients. It has some limitations and does not support all the needs of enterprise-level applications. For example, naming services are often not designed to handle frequent updates. They are designed to store relatively stable information that is not expected to change very often. If a process stops and restarts frequently, a new object reference must be published with each restart. In production environments where many servers start and stop frequently, this can overwork a naming service. Enterprise applications also have other needs that are not met by this simple model—for example, on-demand activation, and centralized administration. These needs are met in a broader Orbix environment, as described in the next section.

Broader Orbix Environment

Overview

Along with the naming service, Orbix offers a number of features that are required by many distributed applications, for flexibility, scalability, and ease of use. These include:

- *Location domains* enable a server and its objects to move to a new process or host, and to be activated on demand.
- *Configuration domains* let you organize ORBs into independently manageable groups. This brings scalability and ease of use to the largest environments.
- The *interface repository* allows clients to discover and use additional objects in the environment—even if clients do not know about these objects at compile time.
- The *event service* allows applications to send events that can be received by multiple objects.

In this section

This section discusses the following topics:

Managing Object Availability

Overview

A system with many servers cannot afford the overhead of manually assigned fixed port numbers, for several reasons:

- Over time, hardware upgrades, machine failures, or site reconfiguration require you to move servers to different hosts.
- To optimize resource usage, rarely used servers only start when they are needed, and otherwise are kept inactive.
- To provide fault tolerance and high availability for critical objects, they can be run within redundant copies of a server. In case of server overload or failure, clients can transparently reconnect to another server

Orbix location domains provide all of these benefits, without requiring explicit programming.

Transient and persistent objects

A server makes itself available to clients by publishing interoperable object references, or *IOR*s. An IOR contains an object's identity and address. This address can be of two types, depending on whether the object is transient or persistent:

- The IORs of transient objects always contain the server host machine's address. A client that invokes on this object sends requests directly to the server. If the server stops running, the IORs of its transient objects are no longer valid, and attempts to invoke on these objects raise the `OBJECT_NOT_EXIST` exception.
- The IORs of persistent objects are exported from their server with the address of the domain's *locator daemon*. This daemon is associated with a database, or *implementation repository*, which dynamically maps persistent objects to their server's actual address.

Invocations on persistent objects

When a client invokes on a persistent object, Orbix locates the object as follows:

1. When a client initially invokes on the object, the client ORB sends the invocation to the locator daemon.
2. The locator daemon searches the implementation repository for the actual address of a server that runs this object in the implementation repository. The locator daemon returns this address to the client.
3. The client connects to the returned server address and directs this and all subsequent requests for this object to that address.

All of this work is transparent to the client. The client never needs to contact the locator daemon explicitly to obtain the server's location.

Locator daemon benefits

Using the locator daemon provides two benefits:

- By interposing the locator daemon between client and server, a location domain isolates the client from changes in the server address. If the server changes location—for example, it restarts on a different host, or moves to another port— the IORs for persistent objects remain valid. The locator daemon supplies the server's new address to clients.
- Because clients contact the locator daemon first when they initially invoke on an object, the locator daemon can launch the server on behalf of the client. Thus, servers can remain dormant until needed, thereby optimizing use of system resources.

Components of an Orbix location domain

An Orbix location domain consists of two components: a locator daemon and a node daemon:

locator daemon: A CORBA service that acts as the control center for the entire location domain. The locator daemon has two roles:

- Manage the configuration information used to find, validate, and activate servers running in the location domain.
- Act as the contact point for clients trying to invoke on servers in the domain.

node daemon: Acts as the control point for a single host machine in the system. Every machine that runs a server must run a node daemon. The node daemon starts, monitors, and manages servers on its machine. The locator daemon relies on node daemons to start processes and tell it when new processes are available.

Scaling Orbix Environments with Configuration Domains

Overview

Small environments with a few applications and their ORBs can be easy to administer manually: you simply log on to systems where the ORBs run and adjust configuration files as needed. However, adding more ORBs can substantially increase administrative overhead. With configuration domains, you can scale an Orbix environment and minimize overhead.

Grouping related applications

Related application ORBs usually have similar requirements. A configuration domain defines a set of common configuration settings, which specify available services and control ORB behavior. For example, these settings define libraries to load at runtime, and initial object references to services.

File- and repository-based configurations

Configuration domain data can be maintained in two ways:

- As a set of files distributed among domain hosts.
- In a centralized configuration repository.

Each ORB gets its configuration data from a domain, regardless of how it is implemented. Orbix environments can have multiple configuration domains organized by application, by geography, by department, or by some other appropriate criteria. You can divide large environments into smaller, independently manageable Orbix environments.

Simple configuration domain and location domain

Figure 4 shows a simple configuration, where all ORBs are configured by the same domain. Such a configuration is typical of small environments. In fact, many environments begin with this configuration and grow from there.

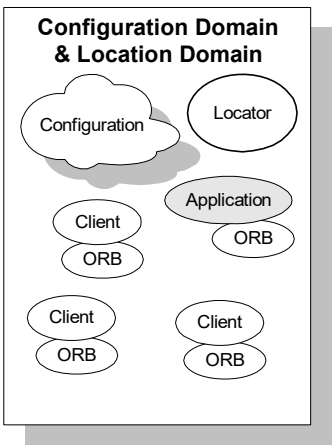

Figure 4: *Simple Configuration Domain and Location Domain*

Multiple configuration and location domains

Figure 5 shows an environment with multiple configuration domains. This environment can be useful in a organization that must segregate user groups. For example, separate configurations can be used for production and finance departments, each with different security requirements. In this environment, all clients and servers use the same locator daemon; thus, the two configuration domains are encompassed by a single location domain.

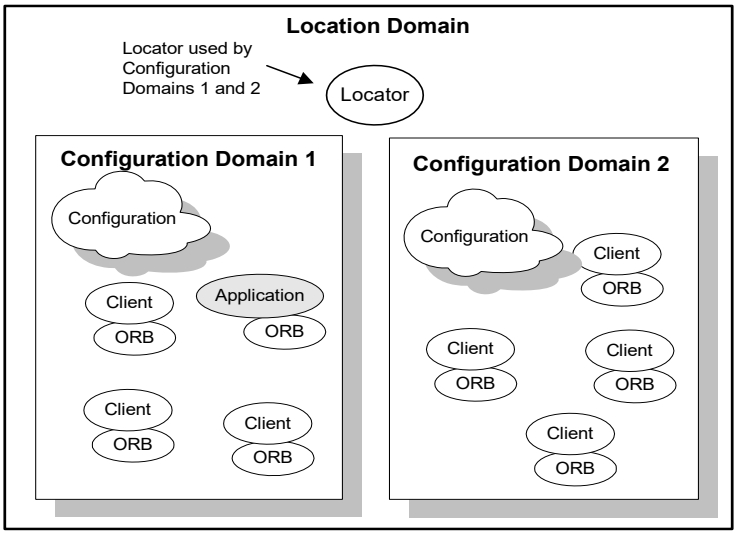

Figure 5: *Multiple Configuration Domains*

Using Dynamic Orbix Applications

Overview

Within the CORBA model, client programs can invoke on remote objects, even if those objects are written in a different programming language and run on a different operating system. CORBA's Interface Definition Language (*IDL*) makes this possible. IDL is a declarative language that lets you define interfaces that are independent of any particular programming language and operating system.

Orbix includes a CORBA IDL compiler, which compiles interface definitions along with the client and server code. A client application compiled in this way contains internal information about server objects. Clients use this information to invoke on objects.

This model restricts clients to using only those interfaces that are known when the application is compiled. Adding new features to clients requires programmers to create new IDL files that describe the new interfaces and to recompile clients along with the new IDL files.

Orbix provides an interface repository, which enables clients to call operations on IDL interfaces that are unknown at compile time. The interface repository (IFR) provides centralized persistent storage of IDL interfaces. Orbix programs can query the interface repository at runtime, to obtain information about IDL definitions.

Managing an interface repository

Administrators and programmers can use interface repository management commands to add, remove, and browse interface definitions in the repository. Interfaces and types that are already defined in a system do not need to be implemented separately in every application. They can be invoked at runtime through the interface repository. For more details on managing an interface repository, see Chapter 7.

Orbix Administration

Overview

Orbix services, such as the naming service, and Orbix components, such as the configuration repository, must be configured to work together with applications. Applications themselves also have administration needs.

This section identifies the different areas of administration. It explains the conditions in the environment and in applications that affect the kind of administration you are likely to encounter. Orbix itself usually requires very little administration when it is set up and running properly. Applications should be easy to manage when designed with management needs in mind.

Administration tasks

Orbix administration tasks include the following:

- Managing an Orbix environment
- Application deployment and management
- Troubleshooting

Managing an Orbix environment

This involves starting up Orbix services, or adding, moving, and removing Orbix components. For example, adding an interface repository to a configuration domain, or modifying configuration settings (for example, initial references to Orbix services). Examples of location domain management tasks include starting up the locator daemon and adding a node daemon. See Part 2 of this manual for more information.

Application deployment and management

An application gets its configuration from configuration domains, and finds persistent objects through the locator daemon. Both the configuration and location domains must be modified to account for application requirements. For more information, see Chapter 3.

Troubleshooting

You can set up Orbix logging in order to collect system-related information, such as significant events, and warnings about unusual or fatal errors. For more information, see Chapter 13.

Administration tools

The Orbix `itadmin` command interface lets you control all aspects of Orbix administration. Administration commands can be executed from any host. For detailed reference information about Orbix administration commands, see Part 4 of this manual.

CHAPTER 2

Selecting an Orbix Environment Model

This chapter shows different ways in which Orbix can be configured in a network environment.

In this chapter

This chapter contains the following sections:

Orbix Development Environment Models

Overview

Business applications must be capable of scaling to meet enterprise level needs. Such applications often extend beyond departments, and even beyond corporate boundaries. Orbix domain and service infrastructures offer a framework for building and running applications that range from small, department-level applications to full-scale enterprise applications with multiple servers and hundreds or thousands of clients.

This chapter offers an overview of Orbix environment models that can handle one or many applications. It also explains Orbix configuration mechanisms, and how to scale an Orbix environment to support more applications, more users, and a wider geographical area. For detailed information on how to set up your Orbix environment, see the *Orbix Deployment Guide*.

Orbix development environments

Orbix development environments are used for creating or modifying Orbix applications. A minimal Orbix development environment consists of the Orbix libraries and the IDL compiler, along with any prerequisite C++ or Java files and development tools.

Application testing requires deployment of Orbix runtime services, such as the configuration repository and locator daemon, naming service, and interface repository.

In environments with multiple developers, each developer must install the Orbix development environment, and the necessary C++ or Java tools. Runtime services can either be installed in each development environment, or distributed among various hosts and accessed remotely.

In this section

This section discusses the following topics:

Independent Development Environments

Overview

This section discusses some typical models of Orbix development (and testing) environments. Actual development environments might contain any one or a blend of these models.

Testing and deployment environment

Figure 6 shows a simple environment that can support application development and testing.

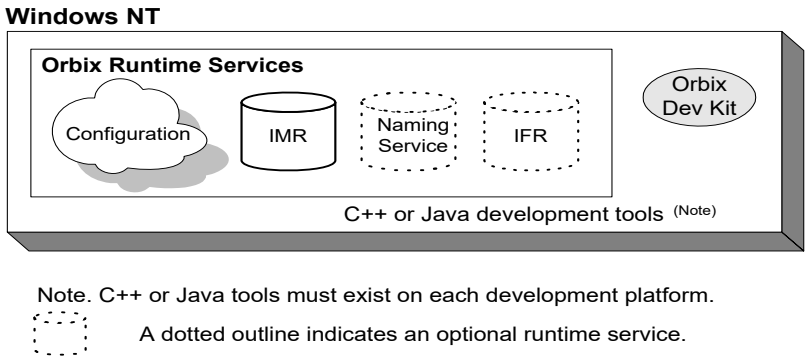

Figure 6: *An Independent Development and Test Environment*

To test an application, it must first be deployed. This involves populating the necessary Orbix repositories (for example, the configuration domain, location domain, and naming service), with appropriate Orbix application data.

This private environment is useful for testing applications on a local scale before introducing them to an environment distributed across a network. Figure 6 shows this environment on Windows NT, but it can be established on any supported platform.

Multiple private environments

Figure 7 is a variant of the model shown in Figure 6 on page 19. In this model, multiple private environments are established on a single multi-user machine. Each of these private environments can be used to create, deploy, and test applications.

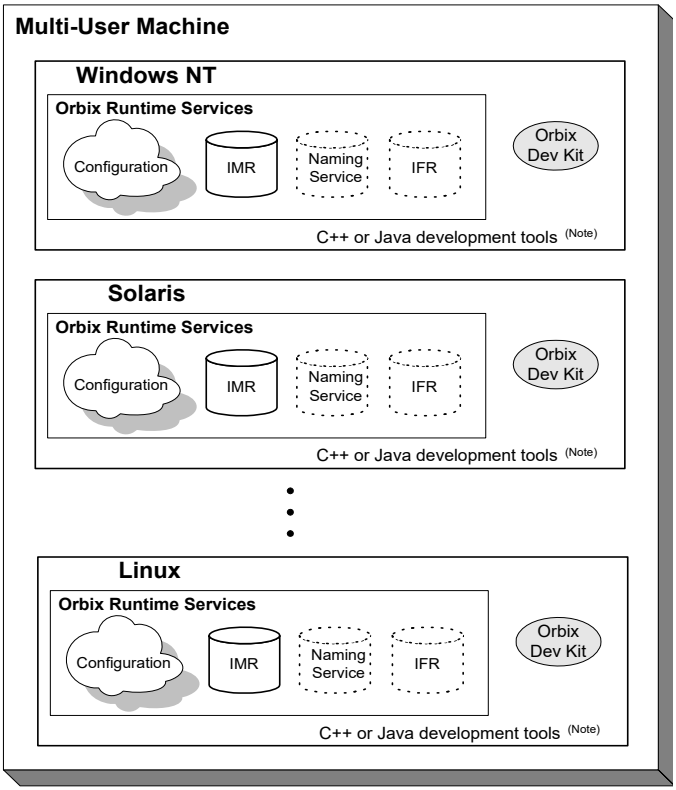

Figure 7: *Multiple Independent Development and Test Environments*

Setting up independent environments

To establish independent development and test environments, first ensure that the appropriate C++ or Java libraries are present. You should then install Orbix on the desired platforms. For information on what C++ or Java libraries are required, and instructions on how to install Orbix, see the *Orbix Installation Guide*.

For information on how to configure and deploy Orbix runtime services in your environment (for example, a locator daemon), see the *Orbix Deployment Guide*.

Distributed Development and Test Environments

Overview

Figure 8 on page 22 illustrates a runtime test environment shared by multiple development platforms. This scenario more closely models the distributed environments in which applications are likely to run. Most applications should be tested in an environment like this before they are deployed into a production environment.

To establish this environment, install the Orbix runtime services in your environment. Ensure that the appropriate C++ or Java libraries are present on your development platforms. Then install the Orbix developer's kit on each platform. For information on how to configure and deploy Orbix runtime services such as the interface repository in your environment, see *Orbix Deployment Guide*.

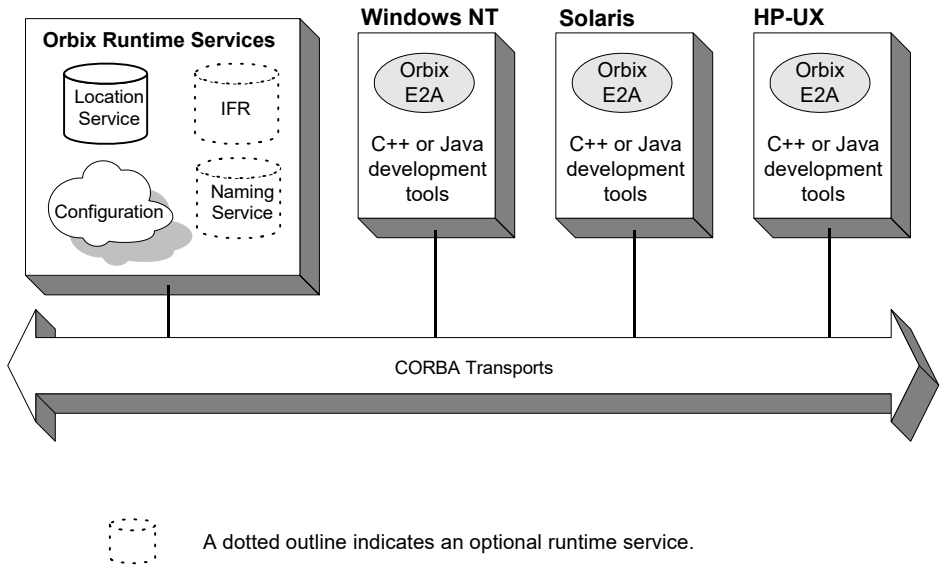

Figure 8: *A Distributed Development and Test Environment*

Configuration Models

Overview

Orbix provides two configuration mechanisms:

- Local file-based configuration
- Configuration repository

For information on managing Orbix configuration domains, see Chapter 3.

Local file-based configuration

A local configuration model is suitable for environments with a small number of clients and servers, or when configuration rarely changes. The local configuration mechanism supplied by Orbix uses local configuration files. Figure 9 on page 24 shows an example Orbix environment where the configuration is implemented in local files on client and server machines.

The Orbix components in Figure 9 on page 24 consist of Orbix management tools, the locator daemon, and configuration files that store the configuration of the Orbix components. When Orbix is installed, it stores its configuration in the same configuration file, but in a separate configuration scope. Application clients store their configurations in files on their host machines. Application clients and servers also include necessary Orbix runtime components, but for simplicity these are not shown in Figure 9 on page 24.

This simple model is easy to implement and might be appropriate for small applications with just a few clients. Keeping these separate files properly updated can become difficult as applications grow or more servers or clients are added.

You can minimize administrative overhead by using a centralized configuration file, which is served to many ORBs using NFS, Windows Networking, or a similar network service. A centralized file is easier to maintain than many local files, because only one file must be kept updated.

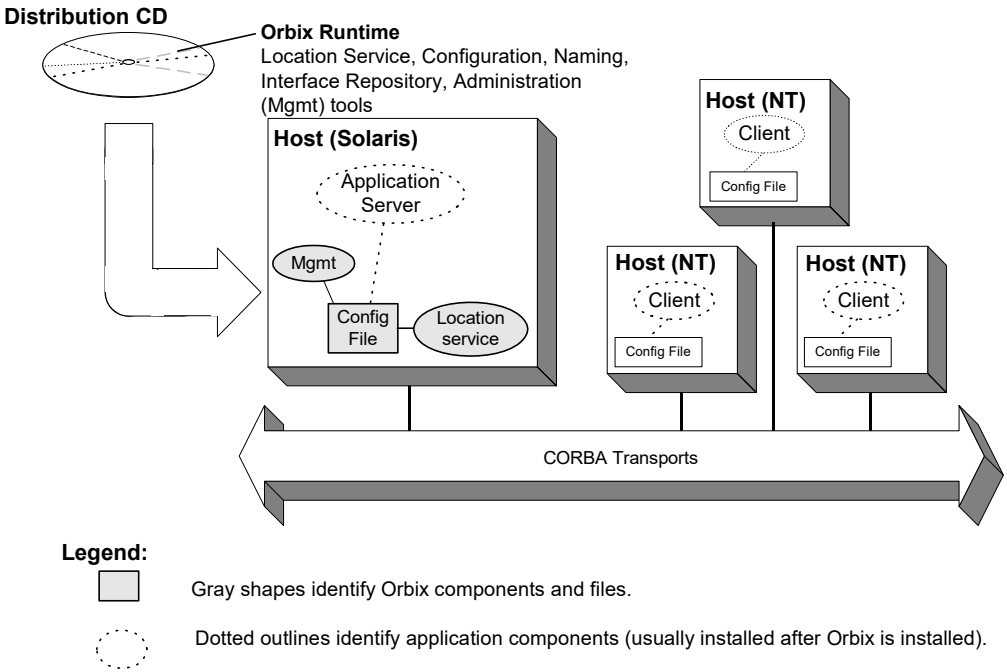

Figure 9: *Orbix Environment with Local Configuration*

Configuration repository

A centralized configuration model is suitable for environments with a potentially large number of clients and servers, or when configuration is likely to change. The Orbix configuration repository provides a centralized database for all configuration information.

The Orbix components in Figure 10 on page 25 consist of the Orbix management tools, the locator daemon, and a configuration repository. The configuration repository stores the configuration for all Orbix components. When servers and clients are installed, they store their configuration in separate configuration scopes in the configuration repository. Application clients and servers also include their own Orbix runtime components, but these are not shown.

This model is highly scalable because more applications can be added to more hosts in the environment, without greatly increasing administration tasks. When a configuration value changes, it must be changed in one place only. In this model, the host running Orbix, the configuration repository, and locator daemon must be highly reliable and always available to all clients and servers.

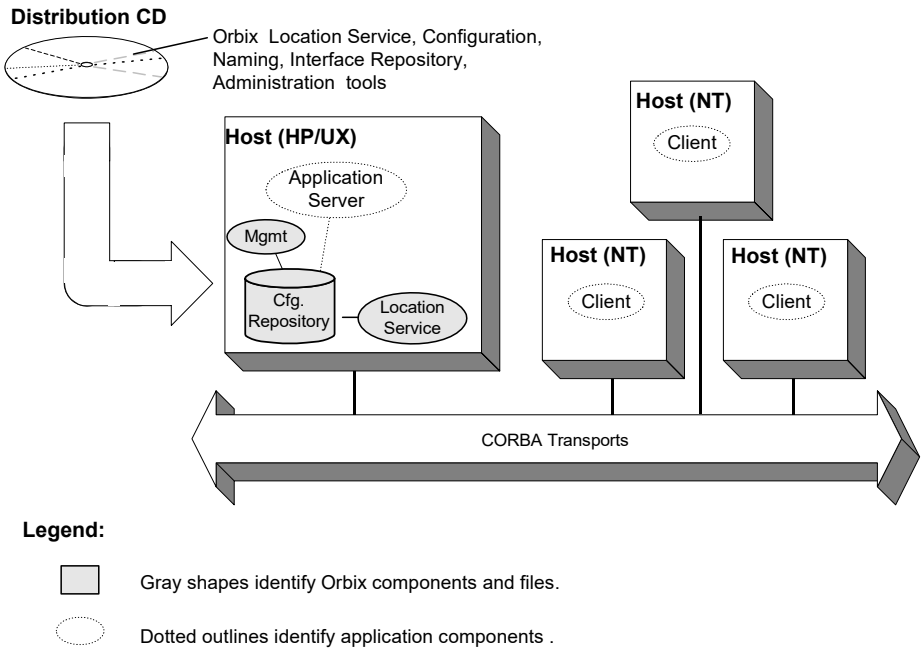

Figure 10: *Orbix Environment with Centralized Configuration*

Getting the Most from Your Orbix Environment

Overview

As you add more or larger applications to your Orbix environment, scalability becomes more crucial. This section discusses some Orbix features that support scalability, and shows how to use them. The following topics are discussed:

- "Using Capabilities of Well-Designed Orbix Applications" on page 27
- "Using the Right Data Storage Mechanism" on page 29

Moving other Orbix services (for example, a naming service), or moving servers also requires some administration to ensure continuation of these services. However, handling these changes is relatively simple and does not involve much administration.

Using Capabilities of Well-Designed Orbix Applications

Orbix optimizations

Like a major highway, Orbix is designed to handle a lot of traffic. For example, when Orbix clients seek their configuration from a centralized configuration mechanism, they compare the version of the locally cached configuration to the version of the live configuration. If versions match, the client uses the cached version. Not reading the entire configuration from the central repository saves time and network bandwidth. Many other programmatic techniques are used throughout Orbix to make it efficient. On the administrative side, proper domain management keeps applications and their clients in an orderly, efficient, and scalable framework.

For such reasons, most applications and environments will not come close to any limitations imposed by Orbix. It is more likely that other network or host-related limitations will get in the way first. Nevertheless, extremely large applications, or large environments with huge numbers of applications and users, are special cases and there are guidelines for keeping such applications and their environments running smoothly.

Special cases

For example, imagine a very large database application with thousands of POAs registered with the locator daemon. If a server restarts, programmatic re-registering of POA state information with the locator daemon can take some time, and even slow down other applications that are using the locator daemon. In such cases, programmers should use the Orbix dynamic activation capability to avoid an unnecessary server-side bottleneck. With dynamic activation, POAs are registered during application deployment. POA state information is handled only if an object is invoked, and only for the POA that is hosting the object.

Looking now at the client side of very large applications, imagine a locator daemon with thousands of registered POAs (for example, an airline ticketing application) handling thousands of client requests per minute. Programmatic optimizations (for example, efficient use of threads, proper organization of the application's POA system or load balancing) help to minimize bottlenecks here. Administrators can take additional steps, such as active connection management, to optimize performance.

Other issues

Other application design issues include multi-threading, how to partition objects across POAs, how to partition POAs across servers, and what POA policies would be best to use under certain circumstances). For more information, see the *CORBA Programmer's Guide*.

Using the Right Data Storage Mechanism

Overview

Orbix provides standard storage mechanisms for storing persistent data used by Orbix and by applications. Access to these standard mechanisms uses the CORBA persistent state service. This service allows alternative storage mechanisms to be used within an environment for storing data for configuration, location, and the naming service. If your applications encounter limitations imposed by a specific storage mechanism, consider moving to an industrial strength database (for example, Oracle or Sybase) at the backend.

Information about implementing alternative storage mechanisms is outside the scope of this guide. Consult your Orbix vendor for more information.

Getting the Most from Orbix Configuration

Overview

This section answers some basic questions administrators might have about using:

- Separate Orbix environments
- Multiple configuration domains

Separate Orbix environments

Companies can use separate Orbix environments to insulate development, test, and production environments from each other. While you can use separate configuration scopes for this, having separate sets of Orbix services reduces the risk of development and test efforts interfering with production-level Orbix services.

Multiple configuration domains

Development environments might use separate configuration domains to isolate development and test efforts from one another. Security policies might also require multiple configuration domains within a single customer environment. For example, separate organizations in a company might have different administrators with different network security credentials.

Geographic separation or network latency issues might also drive a decision to have separate configuration domains.

Part 2

Managing an Orbix Environment

In this part

This part contains the following chapters:

CHAPTER 3

Managing Orbix Configuration

All Orbix clients and servers, including Orbix services such as the locator daemon or naming service, belong to a configuration domain that supplies their configuration settings.

Orbix identifies a client or server by the name of its ORB, which maps to a *configuration scope*. This scope contains configuration variables and their settings, which control the ORB's behavior. Configuration domains can be either based on a centralized configuration repository, or on configuration files that are distributed among all application hosts. Both configuration types operate according to the principles described in this chapter.

Note: For detailed information on how to set up an Orbix environment, see the *Orbix Deployment Guide*.

In this chapter

This chapter contains the following sections:

How an ORB Gets its Configuration

Overview

Every ORB runs within a configuration domain, which contains variable settings that determine the ORB's runtime behavior. Figure 12 summarizes how an initializing ORB obtains its configuration information in a repository-based system, where services are distributed among various hosts.

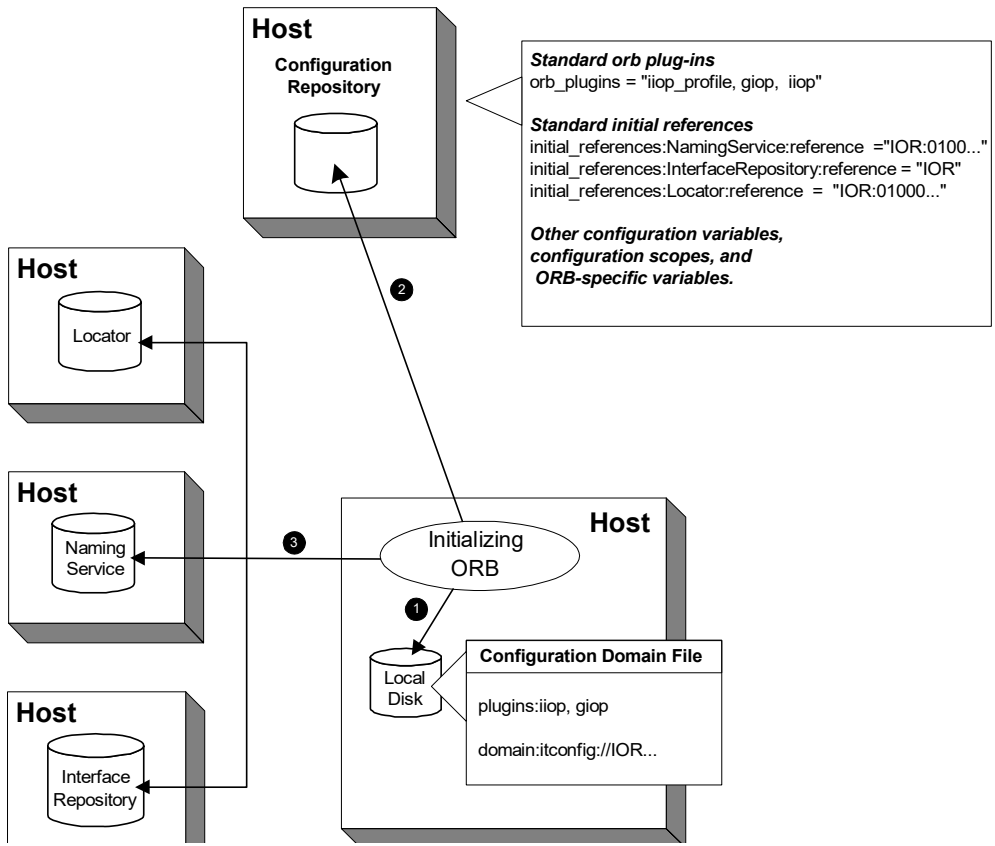

Figure 11: *How an Orbix Application Obtains its Configurations*

1. The initializing ORB reads the local configuration file, which is used to contact the configuration repository.

   Note: In repository-based configuration domains, the local configuration file contains a `domain` configuration variable, which is set to the repository's IOR. For example:

   ```
   domain = "itconfig://00034f293b922...00d3";
   ```

   In a file-based configuration, the *`domain-name`*`.cfg` file does not contain a `domain` variable; instead, the local configuration file itself contains all configuration data.

2. The ORB reads configuration data from the configuration repository, and obtains settings that apply to its unique name. This establishes the normal plug-ins and locates other CORBA services in the domain.
3. The fully initialized ORB communicates directly with the services defined for its environment.

Configuration steps

An initializing ORB obtains its configuration in two steps:

1. Locates its configuration domain.
2. Obtains its configuration settings.

The next two sections describe these steps.

Locating the Configuration Domain

An ORB locates its configuration domain as described in the following language-specific sections.

C++ applications

In C++ applications, the ORB obtains the domain name from one of the following, in descending order of precedence:

1. The `-ORBconfig_domain` command-line parameter
2. The `IT_CONFIG_DOMAIN` environment variable
3. `default-domain.cfg`

The domain is located in one of the following, in descending order of precedence:

1. The path set in either the `-ORBconfig_domains_dir` command line parameter or the `IT_CONFIG_DOMAINS_DIR`.environment variable.
2. The `domains` subdirectory to the path set in either the `-ORBconfig_dir` command-line parameter or the `IT_CONFIG_DIR`.environment variable.
3. The default configuration directory:
   UNIX

```
/etc/opt/microfocus
```

   Windows

```
%IT_PRODUCT_DIR%\etc
```

Java applications

In Java applications, the ORB obtains the domain name from one of the following, in descending order of precedence:

1. The `-ORBconfig_domain` command-line parameter.
2. The `ORBconfig_domain` Java property.
3. `default-domain.cfg`.

The domain is located in one of the following, in descending order of precedence:

1. The path set in either the `-ORBconfig_domains_dir` command-line parameter or the `ORBconfig_domains_dir` Java property.
2. The `domains` subdirectory to the path set in either the `-ORBconfig_dir` command-line parameter or the `ORBconfig_dir` Java property.
3. All directories specified in the classpath.

Note: Java properties can be set for an initializing ORB in two ways, in descending order of precedence:

- As system properties.
- In the `iona.properties` properties file. See "Java properties" on page 411 for information on how an ORB locates this file.

Obtaining an ORB's Configuration

Overview

All ORBs in a configuration domain share the same data source—either a configuration file or a repository. Configuration data consists of variables that determine ORB behavior. These are typically organized into a hierarchy of scopes, whose fully-qualified names map directly to ORB names. By organizing configuration variables into various scopes, you can provide different settings for individual ORBs, or common settings for groups of ORBs.

Configuration scopes apply to a subset of ORBs or a specific ORB in an environment. Orbix services such as the naming service have their own configuration scopes. Orbix services scopes are automatically created when you configure those services into a new domain.

Applications can have their own configuration scopes and even specific parts of applications (specific ORBs) can have ORB-specific scopes.

Scope organization

Figure 12 shows how a configuration domain might be organized into several scopes:

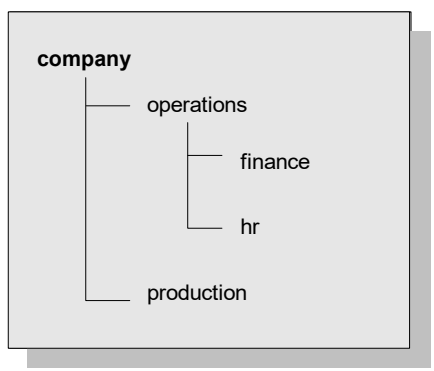

Figure 12: *Hierarchy of Configuration Scopes*

Five scopes are defined:

- `company`
- `company.production`

- company.operations
- company.operations.finance
- company.operations.hr

Given these scopes, and the following ORB names:

```
company.operations.finance.ORB001
company.operations.finance.ORB002
company.operations.finance.ORB003
company.operations.finance.ORB004
```

All ORBs whose names are prefixed with company.operations.finance obtain their configuration information from the company.operations.finance configuration scope.

Variables can also be set at a configuration's root scope—that is, they are set outside all defined scopes. Root scope variables apply to all ORBs that run in the configuration domain.

Scope name syntax

An initializing ORB must be supplied the fully qualified name of its configuration scope. This name contains the immediate scope name and the names of all parent scopes, delimited by a period (.). For example:

```
company.operations.hr
```

ORB name mapping

An initializing ORB maps to a configuration scope through its ORB name. For example, if an initializing ORB is supplied with a command-line -ORBname argument of company.operations, it uses all variable settings in that scope, and the parent company and root scopes. Settings at narrower scopes such as company.operations.finance, and settings in unrelated scopes such as company.production, are unknown to this ORB and so have no effect on its behavior.

If an initializing ORB doesn't find a scope that matches its name, it continues its search up the scope tree. For example, given the hierarchy shown earlier, ORB name company.operations.finance.payroll will fail to find a scope that matches. An ORB with that name next tries the parent scope company.operations.finance. In this case, ORB and scope names match and the ORB uses that scope. If no matching scope is found, the ORB takes its configuration from the root scope.

Defining configuration scopes

After you create a configuration domain, you can modify it to create the desired scopes:

- A file-based configuration can be edited directly with any text editor, or with `itadmin` commands `scope create` and `scope remove`.
- A repository-based configuration can only be modified with `itadmin` commands `scope create` and `scope remove`.

File-based configuration

In a file-based configuration, scopes are defined as follows:

```
scope-name
{
   variable settings
   ...
   nested-scope-name
   {
      variable settings
      ...
   }
}
```

For example, the following file-based Orbix configuration information defines the hierarchy of scopes shown in Figure 12 on page 38:

```
company
{
   # company-wide settings
   operations
   {
      # Settings common to both finance and hr

      finance
      {
         # finance-specific settings
      }
      hr
      {
         # hr-specific settings
      }

   } # close operations scope
   production
   {
      # production settings
   }

} # close company scope
```

itadmin commands

You can create the same scopes with `itadmin` commands, as follows:

```
itadmin scope create company
itadmin scope create company.production
itadmin scope create company.operations
itadmin scope create company.operations.finance
itadmin scope create company.operations.hr
```

Precedence of variable settings

Configuration variables set in narrower configuration scopes override variable settings in wider scopes. For example, the `company.operations.orb_plugins` variable overrides `company.orb_plugins`. Thus, the plug-ins specified at the `company` scope apply to all ORBs in that scope, except those ORBs that belong specifically to the `company.operations` scope and its child scopes, `hr` and `finance`. Example 1 shows how a file-based configuration might implement settings for the various configurations shown in Figure 12 on page 38:

Example 1: *File-Based Configuration*

```
1  company
   {
      # company-wide settings

      # Standard ORB plug-ins
       orb_plugins =
          ["local_log_stream", "iiop_profile", "giop", "iiop"];

      # Standard initial references.
      initial_references:RootPOA:plugin = "poa";
      initial_references:ConfigRepository:reference
                                      = "IOR:010000002000...00900";
      initial_references:InterfaceRepository:reference
                                      = "IOR:010000002000...00900";
     # Standard IIOP configuration
         policies:iiop:buffer_sizes_policy:max_buffer_size = -1
2     operations
      {
         # Settings common to both finance and hr

         # limit binding attempts
         max_binding_iterations = "3";
3        finance
         {
            # finance-specific settings

            # set 5-second timeout on invocations
            policies:relative_binding_exclusive_request_timeout =
                                                              "5000"
         }
```

Example 1: *File-Based Configuration*

```
      hr
      {
         # hr-specific settings

         # set 15-second timeout on invocations
         policies:relative_binding_exclusive_request_timeout =
                                                          "15000"
      }

   } # close operations scope
   production
   {
      # production settings
      policies:iiop:buffer_sizes_policy:max_buffer_size =
        "4096";

   }

} # close company scope
```

1. The `company` scope sets the following variables for all ORBs within its scope:
   - `orb_plugins` specifies the plug-ins available to all ORBs.
   - Sets initial references for several servers.
   - Sets an unlimited maximum buffer size for the IIOP transport.
2. ORBs in the `operations` scope limit all invocations to three rebind attempts.
3. All ORBs in the `finance` scope set invocation timeouts to 5 seconds.
4. All ORBs in the `hr` scope set invocation timeouts to 15 seconds.
5. The `production` scope overrides the `company`-scope setting on `policies:iiop:buffer_sizes_policy:max_buffer_size`, and limits maximum buffer sizes to `4096`.

Sharing scopes

All ORBs in a configuration domain must have unique names. To share settings among different ORBs, define a common configuration scope for them. For example, given two ORBs with common configuration settings, a file-based configuration might define their scopes as follows:

```
common {
   # common settings here
   # ...
   server1 {
      #unique settings to server1
   }
   server2 {
      #unique settings to server2
      ...
   }
} # close common scope
```

Thus, the two ORBs—`common.server1` and `common.server2`—share `common` scope settings.

If an ORB has no settings that are unique to it, you can omit defining a unique scope for it. For example, if `common.server2` has no unique settings, you might modify the previous configuration as follows:

```
common {
   # common settings here
   # ...
   server1 {
      #unique settings to server1
   }
} # close common scope
```

When the `common.server2` ORB initializes, it fails to find a scope that matches its fully qualified names. Therefore, it searches up the configuration scope tree for a matching name, and takes its settings from the parent scope, `common`.

Configuration Variables and Namespaces

Variable components

Configuration variables determine an ORB's behavior, and are organized into namespaces. For example, a configuration might contain the following entry:

```
initial_references:IT_Locator:reference ="IOR:010000...0900";
```

This variable consists of three components:

- The `initial_references:IT_Locator` namespace.
- The variable name `reference`.
- A string value.

Namespaces

Configuration namespaces are separated by a colon (`:`). Configuration namespaces group related variables together—in the previous example, initial references. Orbix defines namespaces for its own variables. You can define your own variables within these namespaces, or create your own namespaces.

Data types

Each configuration variable has an associated data type that determines the variable's value. When creating configuration variables, you must specify the variable type.

Data types can be categorized into two types:

- Primitive types
- Constructed types

Primitive types

Three primitive types, `boolean`, `double`, and `long`, correspond to IDL types of the same name. See the *CORBA Programmer's Guide* for more information.

Constructed types

Orbix supports two constructed types: `string` and `ConfigList` (a sequence of strings).

A `string` type is an IDL string whose character set is limited to the character set supported by the underlying configuration domain type. For example, a configuration domain based on ASCII configuration files could only support ASCII characters, while a configuration domain based on a remote configuration repository might be able to perform character set conversion.

Variables of the string type also support string composition. A composed string variable is a combination of literal values and references to other string variables. When the value is retrieved, the configuration system replaces the variable references with their values, forming a single complete string.

The `ConfigList` type is simply a sequence of `string` types. For example:

```
orb_plugins = ["local_log_stream", "iiop_profile",
   "giop","iiop"];
```

Setting configuration variables

`itadmin` provides two commands for setting configuration domain variables:

- `itadmin variable create` creates a variable or namespace in the configuration domain.
- `itadmin variable modify` changes the value of a variable or namespace in a configuration domain.

In a file-based domain, you can use these commands, or you can edit the configuration file manually. In a file-based configuration, all variable values must be enclosed in quotes (`""`) and terminated by a semi-colon (`;`).

Managing Configuration Domains

Configuration management generally consists of the tasks outlined in Table 1.

Table 1: *Configuration Domain Management Tasks*

Perform this task...	By running...
Start the configuration repository	One of the following: `start_domain-name_services` script starts the configuration repository and other domain services. `itconfig_rep run` starts the configuration repository only.
Stop the configuration repository	`itadmin config stop`
View configuration repository contents	`itadmin config dump`
List all replicas of the configuration repository	`itadmin config list_servers`
Convert from a file to a configuration repository	`itadmin file_to_cfr.tcl`
Create scope	`itadmin scope create`
List scopes	`itadmin scope list`
View scope contents	`itadmin scope show`
Create namespace	`itadmin namespace create`
List namespaces	`itadmin namespace list`
View namespace contents	`itadmin namespace show`
Remove namespace	`itadmin namespace remove`
Create variable	`itadmin variable create`
View variable	`itadmin variable show`

Table 1: *Configuration Domain Management Tasks*

Perform this task...	By running...
Modify variable	`itadmin variable modify`
Remove variable	`itadmin variable remove`

Troubleshooting configuration domains

By default, `itadmin` manages the same configuration that it uses to initialize itself. This can be problematic if you need to run `itadmin` in order to repair a configuration repository that is unable to run. In this case, you can run `itadmin` in another configuration domain by supplying the following command-line parameters (or the equivalent environment variable or Java property):

`-ORBdomain_name`	Specifies the configuration for `itadmin`. This is typically a temporary file-based configuration created for this purpose only.
`-ORBadmin_domain_name`	Specifies the configuration domain repository to modify.
`-ORBadmin_config_domains_dir`	Specifies the directory in which to find the the administered configuration. This parameter is required only if the configuration's location is different from the default domain's directory.

For example, the following `itadmin` command runs the `itadmin` tool in the `temp-domain` domain, and adds the `orb_plugins` variable to the repository of the `acme-products` domain:

```
itadmin -ORBdomain_name temp-domain
        -ORBadmin_domain_name acme-products
        variable create -type list
        -value iiop_profile,giop,iiop orb_plugins
```

CHAPTER 4

Managing Persistent CORBA Servers

Location and activation data for persistent CORBA servers are maintained by the locator daemon in the implementation repository.

In this chapter

This chapter explains how to register and manage server information in a location domain. It contains the following sections:

Introduction

Overview

CORBA servers that export persistent objects must be registered with a locator daemon using its implementation repository. Servers that are registered with the same locator daemon comprise a *location domain*. Through the implementation repository, a locator daemon can locate persistent objects on any server in its domain. A server can also be configured for automatic activation, if necessary, through a *node daemon* that runs on each domain host.

Management tasks

After you register persistent servers in an implementation repository, servers and clients use this repository transparently. A configured location domain typically requires very little outside management. However, occasional circumstances might require you to manage a location domain. For example:

- The locator daemon stops and needs to be restarted, or runtime parameters need to be updated.
- An application is installed, moved, or removed, and application data needs to be updated.
- Activation parameters need to be changed—for example, the command line arguments passed into a server.

itadmin commands

`itadmin` commands lets you update and view data in the implementation repository. You can issue these commands manually from the command line or the `itadmin` command shell, or automatically through an application setup script. You can execute these commands from any host that belongs to the location domain.

Registering Persistent Servers

CORBA persistent servers

A persistent CORBA server is one whose ORB contains persistent POAs. All persistent POAs must be registered in the implementation repository of that server's location domain. When the server initializes, the following occurs:

1. The server's ORB creates communication endpoints for its persistent POAs, where POA managers listen for incoming object requests.
2. The ORB sends POA endpoint addresses to the locator daemon, which registers them in the implementation repository against the corresponding entry.
3. The locator daemon returns its own address to the server's ORB. Persistent POAs that run in this ORB embed that address in all persistent object references.

Because a persistent object's IOR initially contains the locator daemon's address, the locator daemon receives the initial invocation and looks up the object's actual location in the implementation repository. It then returns this address back to the client, which sends this and later invocations on the object directly to the server.

By relying on the locator daemon to resolve their location, persistent objects and their servers can exist anywhere in the location domain. Furthermore, an implementation repository can register server processes for on-demand activation and for per-client activation.

Persistent server registration process

In general, registration of a persistent server is a three-step process:

1. "Register the server process for on-demand activation".
2. "Register the ORB" that runs in that process.
3. "Register POAs" that run in the ORB.

This section shows how to use `itadmin` commands to perform these tasks. You can enter these commands either on the command line, or using a script.

Per-client activation is a special case of on-demand activation that provides a one-to-one mapping between clients and server processes. See "Per-client activation" on page 54 for more details.

Register the server process for on-demand activation

`itadmin process create` lets you register a process with a location domain for on-demand activation. When a locator daemon receives an invocation for an object whose server process is inactive, it contacts the node daemon that is registered for that process, which activates the process.

The following example registers the `my_app` server process with the `oregon` node daemon:

```
itadmin process create
      -node_daemon iona_services.node_daemon.oregon
      -pathname "d:/bin/myapp.exe"
      -startupmode on_demand
      -args "training.persistent.my_server
             -ORBname my_app.server_orb" my_app
```

In this example, the `process create` command takes the following parameters:

`-node_daemon`	Specifies the node daemon that resides on the process's host. This node daemon is responsible for starting the process.
`-startupmode`	When set to `on_demand`, this specifies that the node daemon restarts the server process when requested.
`-args`	Specifies command-line arguments. Use the `-args` argument to specify the ORB name and (for Java executables) the Java class name. You can also use this argument to set the Java class path.

For more about these and other parameters, see `process create`.

Register the ORB

After you register a server process, associate it with the name of the ORB that it initializes, using `itadmin orbname create`. This name must be the same as `-ORBname` argument that you supply the server during startup. For example, the following command associates the registered process, `my_app`, with the `my_app.server_orb` ORB:

```
itadmin orbname create -process my_app my_app.server_orb
```

The ORB name must be unique in the location domain; otherwise an error is returned.

Note: If you change an ORB name to make it unique in the location domain, also be sure to change the ORB name that is specified for the server. If an ORB-specific scope has been established in the configuration domain, also change the configuration scope name.

Register POAs

After you register a server process and its ORB, register all persistent POAs and their ancestors—whether persistent or transient—using `itadmin poa create`. Persistent POAs must be registered with the ORB name (or in the case of replicated POAs, ORB names) in which they run. For example, the following command registers the `banking_service/account/checking` persistent POA and its immediate ancestors `banking_service/checking` and `banking_service` with the `my_app.server_orb` ORB:

```
itadmin poa create -orbname my_app.server_orb \
   banking_service
itadmin poa create \
   banking_service/account -transient
itadmin poa create -orbname my_app.server_orb \
   banking_service/account/checking
```

All POA names within a location domain must be unique. For more information about avoiding name conflicts, see "Ensuring Unique POA Names" on page 69.

Transient POAs

A transient POA does not require state information in the implementation repository. However, you must register its POA name in the implementation repository if it is in the path of any persistent POAs below it. In the previous example, the `banking_service/account` transient POA is registered as the parent of the `banking_service/account/checking` persistent POA.

POA replicas

Orbix implements server replication at the POA level. To create POA replicas, specify the ORB names in which they run using the `-replicas` argument. For more details, refer to "Building a Replicated Server" on page 89.

Per-client activation

You can register a process for per-client activation using the `itadmin process create` command. In this case, instead of multiple clients sharing the same server, a new process is created for each client. When the locator daemon receives an invocation for an object whose server process is registered as `per_client`, it creates a new ORB name and process, and contacts the registered node daemon to launch the server.

The following example registers the `my_app` server process with the `oregon` node daemon for per-client activation:

```
itadmin process create
 -node_daemon iona_services.node_daemon.oregon
 -pathname "d:/bin/myapp.exe"
 -startupmode per_client
 -args "training.persistent.my_server
        -ORBname %o" my_app
```

In this example, the `process create` command takes the following parameters:

`-node_daemon`	Specifies the node daemon that resides on the process's host. This node daemon is responsible for starting the process.
`-startupmode`	When set to `per_client`, specifies that the locator creates a new process and ORB name for each client invoking on objects in the associated persistent POA, and requests the node daemon to start the process.
`-args`	Specifies the command-line arguments. Because the locator generates the ORB name, any string matching `%o` in the process's argument list is replaced with the name of the new ORB. Similarly, any string matching `%p` is replaced with the name of the process created by the locator.

To ensure that multiple servers containing the same object can co-exist, the locator creates a new ORB name and a new process for each client. The new ORB name is created by appending an *id* string to the registered ORB name, where *id* is an integer value. In this example, the created ORB names might be `my_app.12` and `my_app.3`. This naming scheme ensures that configuration variables can be shared between the server processes. New process names are created in a similar manner. When a server process has terminated, the locator can reuse the ORB name and process name.

WARNING: The locator or node daemons do not terminate the server process when the server's associated client terminates. It is the application's responsibility to terminate the server process by, for example:

- adding a shutdown operation that is invoked by the client before the client terminates;
- using the leasing plug-in to detect when the client has completed;
- making the server to terminate after a certain amount of time has elapsed without any invocation.

The persistent POA associated with a per-client activated process must support dynamic addition of replicas. This support is automatically enabled when creating a POA whose associated process's startup-mode is per-client. See `poa create` and `poa modify` for more details.

For more information about these and other parameters, see `process create`.

Server Environment Settings

Overview

When a registered server process starts, it is subject to its current environment.

In this section

The following sections discuss:

Windows Environment Settings

Creation flag settings

The following creation flag settings apply:

DETACHED_PROCESS for console processes, denies the newly created process access to the console of the parent process.

CREATE_NEW_PROCESS_GROUP identifies the created process as the root process of a new process group. The process group includes all processes that are descendants of this root process.

CREATE_DEFAULT_ERROR_MODE specifies that the created process does not inherit the error mode of the calling process.

NORMAL_PRIORITY_CLASS indicates a normal process with no special scheduling needs.

Handle inheritance

Open handles are not inherited from the node daemon.

Security

The new process's handle and thread handle each get a default security descriptor.

UNIX Environment Settings

File access permissions

You can set user and group IDs for new processes using the `-user` and `-group` arguments to `itadmin process create`. Before setting user or group IDs for the target process, ensure that the following applies on the host where the target process resides:

- The specified user exists in the user database.
- The specified group exists in the group database.
- The specified group matches the primary group of the specified user in the user database.

If the specified group does not match the primary group in the users database, the specified user must be a member of the specified group in the group database.

Note: If you cannot edit the `/etc/group` file, specify the user's primary group. This allows the server to operate normally, even if the `/etc/group` file is not well maintained.

Before a server starts, the file access privilege of the activated process is lowered if the node daemon is the superuser. If the node daemon is not the superuser, the activated process has the same privileges as the node daemon.

Check whether newly activated target processes have `set-uid/set-gid` permissions. These allow the server to change the effective user and group IDs, enabling a possible breach of security.

The user and group ID settings affect the working directory settings (if directory paths are created) and the open standard file-descriptor processing.

File creation permissions

The file mode creation mask is set by supplying the `-umask` argument to `itadmin process create`. By default, the `umask` is `022` and the actual creation mode is `755` (`rwxr-xr-x`).

The `umask` setting affects the current directory setting (if directory paths are created) and the open standard file-descriptor processing.

Open file descriptors	The activated server has only standard input, output, and error open for both reading and writing, and is connected to `/dev/null` instead of to a terminal.
Resource limits	Resource limits are inherited from the node daemon.
Session leader	The activated server creates a new session and becomes leader of the session and of a new process group. It has no controlling terminal.
Signal disposition	All valid signals between `1` and `NSIG-1` are set to their default dispositions for the activated server.

Managing a Location Domain

Management tasks

Location domain management generally consists of the following tasks:

- Managing server processes.
- Managing the locator daemon.
- Managing node daemons.
- Listing location domain data.
- Modifying a location domain.
- Ensuring that all POA names within a domain are unique.

Managing Server Processes

Starting and stopping registered server processes

Server processes that are registered for on-demand activation do not require any manual intervention. You only need to explicitly start and stop processes that are not set for on-demand activation.

To manually start a registered target server process on a host where a node daemon resides, use the `itadmin process start` command. For example:

```
itadmin process start my_app
```

To stop a registered target server process on the host where the node daemon resides, use the `itadmin process stop` command. For example:

```
itadmin process stop my_app
```

Securing server processes

You can specify that the node daemon can launch processes only from a list of secure directories, in one of two ways:

- Set the `itnode_daemon run`'s `-ORBsecure_directories` parameter.
- Set the `secure_directories` configuration variable.

Both specify a list of secure directories in which the node daemon can launch processes. When the node daemon attempts to launch a registered process, it checks its pathname against the `secure_directories` list. If a match is found, the process is activated; otherwise, the node daemon returns a `StartProcessFailed` exception to the client.

Moving manually launched processes

A process that is not registered to be launched on demand can be moved to a new host by stopping it on its current host, and restarting it on the new host.

This behavior can be disabled by setting the following configuration variable to `false`, and restarting the locator:

`plugins:locator:allow_node_daemon_change`

Attempting to move a process that is already active or is registered to be launched on demand results in an error.

Managing the Locator Daemon

Overview

A locator daemon enables clients to locate servers in a network environment. Normally, a locator daemon runs as root on UNIX, or with administrator privileges on Windows NT. To start and stop a locator daemon, you must be logged on as UNIX root or with Windows NT administrator privileges.

This section assumes that Orbix has been installed and configured to run within your network environment. For more on configuring and deploying Orbix, see *Orbix Deployment Guide*.

Starting a locator daemon

To start a locator daemon:

1. On the machine where the locator daemon runs, log on as root or NT administrator.
2. Open a terminal or command window.
3. Enter `itlocator run`

   By default, this runs the locator daemon in the foreground.
4. Complete the appropriate actions for your platform as specified below.

   Windows

   Leave the command window open while the locator is running.

   UNIX

   Leave the terminal window open or use operating system commands to run the process in the background.

Note: In a configuration repository domain, the configuration repository must be running before starting the locator daemon.

Stopping a locator daemon

To stop a locator daemon, use the `itadmin locator stop` command. This command has the following syntax:

```
itadmin locator stop locator-name
```

Stopping all daemons and monitored processes

To stop the locator, all registered node daemons, and monitored processes running in the location domain, use the `-alldomain` argument:

```
itadmin locator stop -alldomain locator-name
```

Restarting a locator daemon

If a locator daemon is stopped and restarted while server processes are active, it recovers information about the active processes when it starts up again. The locator daemon validates that server processes, ORBs and POAs that were active when it was shutdown are still responding. If these server processes are no longer running, the locator daemon can detect this.

Managing Node Daemons

Overview

In an Orbix location domain, the node daemon is responsible for activating and managing server processes. Every host running an server must also run a node daemon. The node daemon performs the following tasks:

- Starts processes on demand.
- Monitors all child processes of registered server processes, and informs the locator daemon about any events relating to these child processes—in particular, when a child process terminates. This enables the locator daemon to remove the outdated dynamic process state information from the implementation repository, and to restart the process if necessary.
- Monitors all services via heartbeating. If a manually started service crashes, the node daemon detects this and returns all requests routed to this server with the appropriate exception.
- Acts as the contact point for servers starting on this machine. When an server starts on a machine, it contacts the locally running node daemon to announce its presence. The node daemon informs the locator daemon of the new server's presence.

Target server processes that are manually started do not need to register their process information with the locator daemon. Even when process information is not registered with the locator daemon, these processes should behave normally with respect to other location domain capabilities (for example, object location).

However, if you enter process information for a manually started server, you can still use manual starting by setting its automatic start-up mode to disabled. You might wish to store this information, to keep a record of all processes installed in the location domain.

Starting a node daemon

To start a node daemon, log on to the host where you want to run the daemon and enter `itnode_daemon run`.

By default, at startup, the node daemon attempts to contact the CORBA servers that it managed during the previous time it ran. If the node daemon was managing a large number of CORBA servers, this can take up to several minutes, and delay the node daemon from starting up.

In certain circumstances—for example, restarting after a reboot—it is not necessary for the node daemon to contact running CORBA servers. This is because it can be guaranteed that those servers are not running. You can use the following configuration variable to turn off this default behavior:

```
plugins:node_daemon:recover_processes="false";
```

This enables the node daemon to complete its initialization more quickly. You should set this variable in the node daemon's configuration scope.

Running multiple node daemons on a single host

One node daemon can control multiple server processes; and normally one node daemon runs on a given host. Sometimes an application might require a separate node daemon (for example, to launch servers as different users). In this case, you can run multiple node daemons on a single host. For example, one node daemon might run as root, and another as a different user with fewer privileges.

Multiple node daemons on the same host must have different names, which should reflect their application name in some way.

To configure multiple node daemons, perform the following steps:

1. In the default `node_daemon` configuration scope, create a sub-scope (for example, `node_daemon.engineering`).
2. Provide a value for the node daemon name configuration variable. For example:

   ```
   itadmin variable create -scope node_daemon.engineering
   -type string -value "eng_node_daemon"
   plugins:node_daemon:name
   ```

3. Run the node daemon in the new scope, using the `-ORBname` argument:

   For example, the following commands start two node daemons on the same host:

   ```
   itnode_daemon
   itnode_daemon -ORBname node_daemon.engineering
   ```

Stopping a node daemon

To terminate a node daemon, use `itadmin node_daemon stop`. This command also stops all the server processes that the node daemon monitors. For example, the following command stops the node daemon on `alaska`:

```
itadmin node_daemon stop alaska
```

Viewing a node daemon's processes

Before you stop a node daemon, you might want to list all the active processes that it currently monitors. To do so, run `itadmin process list -active`. For example, this command lists the active processes for the node daemon on `alaska`:

```
itadmin process list -active -node_daemon alaska
   my_server_process
```

Listing Location Domain Data

With `itadmin` commands, you can list the names and attributes of registered entries in the implementation repository.

Table 2: *Commands that List Location Domain Data*

Command	Action
`process list`	Lists the names of all target processes registered in the location domain.
`process show`	Lists the attributes of server processes registered with the locator daemon.
`orbname list`	Lists all ORB names in the location domain.
`orbname show`	Lists the attributes of ORB names registered with the locator daemon.
`poa list`	Lists the names of all POAs in the location domain.
`poa show`	Lists the attributes of all registered POA names.

Modifying a Location Domain

Overview

With `itadmin` commands, you can modify and remove registered processes, ORB names, and POA names from the implementation repository. For detailed information, see Chapter 23 on page 299.

Modifying entries

The `itadmin` commands listed in Table 3 modify entries for processes, ORB names, and POA names that are registered with a location domain.

Table 3: *Commands that Modify a Location Domain*

Command	Action
`process modify`	Modifies the specified process entry.
`orbname modify`	Associates an ORB name with the specified process name.
`poa modify`	Modifies the specified POA name.

Removing entries

You can remove any entry from the implementation repository, whether the target object is running or not. The `itadmin` commands listed in Table 4 remove entries for processes, ORB names, and POA names that are registered with a location domain.

Table 4: *Commands that Remove Location Domain Components*

Command	Action
`process remove`	Removes a process entry.
`orbname remove`	Removes an ORB name from the location domain. If there is an active ORB entry for the ORB name in the locator's active ORB table, this is also removed.
`poa remove`	Removes the entry for the specified POA and its descendants from the location domain. By default, all active entries for the POA and its descendants are also removed.

Ensuring Unique POA Names

Overview

The locator daemon finds persistent objects by looking up their POA names in the implementation repository. Consequently, POA names must be unique in a location domain.

If you use a repository-based configuration, the implementation repository prevents name duplication and raises the following error:

```
ERROR: Unable to add an implementation repository entry for the
POA: EntryAlreadyExists
```

If different Orbix applications use the same POA names, you can avoid name conflicts by setting `plugins:poa:root_name`. The `root_name` variable names the application's root POA, which is otherwise unnamed. By setting this variable for each application's ORB to a unique string, you can ensure unique names for all POAs.

Procedure

The following procedure shows how to register a root POA's name within a location domain, and use it with all descendant persistent POAs:

1. To define a root POA name for a server, create a `plugins:poa:root_name` configuration variable in the server ORB's configuration scope:

   ```
   itadmin variable create
      -scope production.test.servers.server001 -type string
      -value "my_app" plugins:poa:root_name
   ```

   When the server initializes, it reads its root POA name and applies this to all its POA names.

2. Register the root POA's name in the implementation repository:

   ```
   itadmin poa create -transient my_app
   ```

3. When you register persistent POAs for this server in the implementation repository, prefix their names (and the names of all ancestor POAs) with the root POA's prefix. The following commands register two persistent POAs:

```
itadmin poa create -transient my_app/poa1
itadmin poa create -orbname
   production.test.servers.server001 my_app/poa1/poa2
itadmin poa create -orbname
   production.test.servers.server001 my_app/poa1/poa2/poa3
```

Using Direct Persistence

Using direct persistence enables Orbix to bypass the locator daemon when resolving persistent object references or contacting Orbix services.

In this section

This section discusses the following topics:

CORBA Applications

In general, a CORBA applications rely on the location daemon to resolve persistent object references. Alternatively, you might want to avoid the overhead that is incurred by relying on the location daemon. In this case, you can set up a server that generates direct persistent object references—that is, object references whose IORs contain a well-known address for the server process. This section includes:

- "Requirements".
- "Example".
- "Setting direct persistence in configuration only".

Requirements

Two requirements apply:

- The server that generates the object references must set its POA policies to `PERSISTENT`, `DIRECT_PERSISTENCE`. The POA must also have a `WELL_KNOWN_ADDRESSING_POLICY` whose value is set to *`prefix`* (see the *CORBA Programmer's Guide*).
- The configuration must contain a well-known address configuration variable, with the following syntax:

  `address-prefix``::`*`transport`*`:addr_list=[ `*`address-spec [,...]`*` ]`

  where *`address-spec`* has the following syntax:

  "*`[+]host-spec`*`:`*`port-spec`*"

  The plus (+) prefix is optional, and only applies to replicated servers, where multiple addresses might be available for the same object reference (see "Direct Persistence and Replica Failover" on page 86).

Note: These requirements involve setting direct persistence programatically. As an alternative for C++ servers, see also "Setting direct persistence in configuration only".

Example

For example, you might create a well-known address configuration variable in scope `MyConfigApp` as follows:

```
MyConfigApp {
    ...
    my_server:iiop:addr_list=["host.com:1075"];
    ...
}
```

Given this configuration, a POA created in the `MyConfigApp` ORB can have its `PolicyList` set so it generates persistent object references that use direct persistence, as follows:

C++

```
CORBA::PolicyList policies;
policy.length(4);
CORBA::Any persistence_mode_policy;
CORBA::Any well_known_addressing_policy;
persistence_mode_policy_value <<=
   IT_PortableServer::DIRECT_PERSISTENCE;
well_known_addressing_policy_value <<=
   CORBA::Any::from_string("wka", IT_TRUE);

policy[0] = poa->create_lifespan_policy
                (PortableServer::PERSISTENT);
policy[1] = poa->create_id_assignment_policy
                (PortableServer::USER_ID);
policy[2] = orb->create_policy
                (IT_PortableServer::PERSISTENCE_MODE_POLICY_ID,
                 persistence_mode_policy);
policy[3] = orb->create_policy
                (IT_CORBA::WELL_KNOWN_ADDRESSING_POLICY_ID,
                 well_known_addressing_policy);
```

Java

```
import com.iona.corba.*;
import com.iona.IT_CORBA.*;
import com.iona.IT_PortableServer.*;

// Set up IONA policies
org.omg.CORBA.Any persistent_mode_policy_value =
   global_orb.create_any();
org.omg.CORBA.Any well_known_addressing_policy_value =
   global_orb.create_any();
PersistenceModePolicyValueHelper.insert(
    persistent_mode_policy_value,
    PersistenceModePolicyValue.DIRECT_PERSISTENCE);
well_known_addressing_policy_value.insert_string("wka");

org.omg.CORBA.Policy[] policies=new Policy[]
{
  root_poa.create_lifespan_policy(
        LifespanPolicyValue.PERSISTENT),
  root_poa.create_id_assignment_policy(
        IdAssignmentPolicyValue.USER_ID),
  global_orb.create_policy(
        PERSISTENCE_MODE_POLICY_ID.value,
        persistence_mode_policy_value),
  global_orb.create_policy(
        WELL_KNOWN_ADDRESSING_POLICY_ID.value,
        well_known_addressing_policy_value),
};
...
```

Setting direct persistence in configuration only

Orbix has two configuration variables that enable POAs to use direct persistence and well-known addressing, if the policies have not been set programatically. Both variables specify the policy for individual POAs by specifying the fully qualified POA name for each POA. They take the form of `poa:`*`fqpn`*`:`*`variable-name`* (*`fqpn`* is frequently used POA name). For example, to set the well-known address for a POA whose fully qualified POA name is `darleen` you would set the variable `poa:darleeen:well_known_address`.

poa:*`fqpn`***:direct_persistent** specifies if a POA runs using direct persistence. If this is set to `true` the POA generates IORs using the well-known address that is specified in the `well_known_address` variable. Defaults to `false`.

poa:*FQPN*:well_known_address specifies the address used to generate IORs for the associated POA when that POA's `direct_persistent` variable is set to `true`.

For example, by default, the `simple_persistent` demo creates an indirect persistent POA called `simple_persistent`. If you want to run this server using direct persistence, and well known addressing, add the following to your configuration:

```
simple_orb {
     poa:simple_persistent:direct_persistent = "true";
     poa:simple_persistent:well_known_address = "simple_server";
     simple_server:iiop:port = "5555";
 };
```

All object references created by the `simple_persistent` POA will now be direct persistent containing the well known IIOP address of port `5555`.

Obviously, if your POA name was different the configuration variables would need to be modified. The scheme used is the following:

```
poa:FQPN:direct_persistent=<BOOL>;
poa:FQPN:well_known_address=<address_prefix>;
AddressPrefix:iiop:port=<LONG>;
```

FQPN is the fully qualified POA name. This introduces the restriction that your POA name can only contain printable characters, and may not contain white space.

AddressPrefix is the string that gets passed to the well-known addressing POA policy. Specify the actual port used using the variable *AddressPrefix*`:iiop:port`. You can also use `iiop_tls` instead of `iiop`.

Note: This functionality is currently only implemented in the C++ ORB. If you are using the Java ORB, you must set the direct persistence and well known addressing policies programmatically.

Orbix Services

In general, Orbix uses the locator daemon to resolve the initial reference for each of the services. Alternatively, you might want to avoid the overhead that is incurred by relying on the location daemon. In this case, you would configure the service to run in direct persistence mode.

Technical details

When a service runs in direct persistence mode it listens on a fixed host and port number. This information is embedded into the IOR that the service exports as an initial reference.

When a CORBA client asks for the service's initial reference, it receives the IOR containing the host and port information for the service. The client uses the embedded information to directly contact the service, bypassing the locator and node daemon normally used by Orbix services.

Performance issues

While direct persistence reduces the overhead of using the locator and node daemons, it also has a cost in terms of fault tolerance and flexibility. When running in direct persistence mode a service cannot be started on demand and it must always listen on the configured host and port number.

Configuration variables

To configure a service to run in direct persistence mode, three configuration variables need to be modified:

`plugins:`*`ServiceName`***`:direct_persistence`** Indicates whether the service uses direct or indirect persistence. The default value is `FALSE`, which indicates indirect persistence.

plugins:*`ServiceName`***:iiop:port** Specifies the port number that the service will listen on. If security is installed, then a TLS port is also required.

initial_references:*`ServiceReferenceString`***:reference** specifies the IOR of the service.

If the service is clustered, `plugins:`*`ServiceName`*`:iiop:host` must also be set.

Configuring direct persistence

To configure a service to run in direct persistence mode complete the following steps:

1. If the service is running, shut it down.
2. Set `plugins:`*`ServiceName`*`:direct_persistence` to `TRUE` within the service's configuration scope.
3. Within the same configuration scope, set `plugins:`*`ServiceName`*`:iiop:port` to some open port number.
4. Prepare the service. This causes the service to generate a new IOR for itself. The new IOR will be printed to the console. Save it for use in the next step.
5. Within the same configuration scope as used in steps **2** and **3**, replace the value of `initial_references:`*`ServiceReferenceString`*`:reference` with the IOR returned in step **4**.
6. Restart the service.

CHAPTER 5

Configuring Scalable Applications

Enterprise-scale systems, which are distributed across multiple hosts, networks, and applications, must be designed to handle a wide variety of contingencies.

For example, mechanical or electrical malfunctions can cause host machines to stop working. A network can be cut apart by an excavator that accidentally slices through phone lines. Operating systems can encounter fatal errors and fail to reboot. Compiler or programming errors can cause software applications to crash.

Poor design can also cause problems. For example, you might run multiple copies of a web server to handle higher levels of browser activity. However, if you run all copies on the same underpowered host machine, you may reduce, rather than increase, system performance and scalability. Running all web servers on the same host also makes the entire web site dependent on that machine—if it fails, it brings down the entire site.

In general, a distributed enterprise system must facilitate reliability and availability. Otherwise, users and applications are liable to encounter service bottlenecks and outages.

In this chapter

This chapter contains the following sections:

Further information

See Chapter 11 for information on additional features that are designed to enhance scalability and performance (for example, Java new I/O and shared memory).

Fault Tolerance and Replicated Servers

Overview

Reliable and available CORBA applications require an ORB that supports fault tolerance—that is, an ORB that avoids any single point of failure in a distributed application. With the enterprise edition of Orbix, you can protect your system from single points of failure through *replicated servers*.

A replicated server is comprised of multiple instances, or *replicas*, of the same server; together, these act as a single logical server. Clients invoke requests on the replicated server, and Orbix routes the requests to one of the member replicas. The actual routing to a replica is transparent to the client.

Benefits

Orbix replicated servers provide the following benefits:

Client transparency: Client applications can invoke requests on replicated servers without requiring any changes.

Transparent failover: If one replica in a replicated server fails, Orbix automatically redirects clients to another replica, without the clients' knowledge.

Dynamic management: You can modify a replicated server by adding or removing replicas at runtime, without affecting client applications or other replicas.

Replicated infrastructure: Critical services such as the locator daemon, configuration repository, and naming service are configured as replicated servers. This ensures that they are always available.

Load balancing: Client invocations can be routed to different replicas within a replicated server, thus balancing the client load across all, and improving system performance. Orbix provides out-of-the-box round robin and random load-balancing strategies. The Orbix load-balancing framework is pluggable, so you can easily implement your own strategies.

About Replicated Servers

Overview

Orbix replicates servers with the same infrastructure that supports persistent CORBA objects—that is, objects that are maintained in POAs with a lifetime policy of `PERSISTENT`. Orbix locates persistent objects using the locator daemon, which maintains their addresses on a physical server (see "Managing Object Availability" on page 8). A client that invokes on a persistent object for the first time sends its request to the locator daemon, which redirects the request to the server's current host and port. Thus, a client invoking on these objects is insulated from any knowledge of their actual location.

Orbix uses the locator daemon to support replicated servers. If a persistent object is instantiated on a replicated server, its references contain the address of the locator daemon. The locator daemon is responsible for redirecting client requests on that object to one of the server's replicas.

POA replicas

Object persistence is always set by POA policies. Therefore, Orbix implements replication through registration of multiple instances, or *replicas*, of a POA, in a location domain's implementation repository. This provides the necessary level of granularity without adding significant administrative overhead. POA replicas ensure continuous access to persistent objects; and the Orbix infrastructure is required only to monitor POA activity, which it does in any case.

Deployment of a replicated server

For example, you might want to deploy a replicated server that implements the replicated POA `ozzy` on hosts `zep`, `floyd`, and `cream`. To do this, complete the following steps:

Note: The following procedure assumes that a locator daemon and a naming service are already deployed.

1. Register replicas of POA `ozzy` in the location domain's implementation repository. At runtime, each server sends the replica's actual address to the domain's locator daemon. For details on registering POA replicas, see "Example 1: Building a Replicated Server to Start on Demand" on page 90.
2. Make persistent object references in a replicated server available to prospective clients—typically, by advertising object references through the CORBA naming service.
3. Ensure that the node daemon activates servers on the initial client request. Otherwise, you must manually activate those servers.

Replicated server startup

When the servers start up, the following occurs:

1. Each server's ORB creates communication endpoints for its persistent POAs, where POA managers listen for incoming object requests.
2. The ORB sends POA endpoint addresses to the locator daemon, which registers them in the implementation repository against the corresponding POA entry. If a persistent POA is replicated across multiple servers, each replica's address is registered against the corresponding replica entry. Thus, the locator daemon can maintain multiple addresses for the same POA.
3. The locator daemon returns its own address to each ORB. Persistent POAs that run in this ORB embed that address in all persistent object references.

Invocations on replicated persistent objects

When a client invokes on a persistent object in the replicated server, the following occurs:

1. The client ORB sends a locate request to the object reference's communication endpoint, which is the locator daemon.
2. When the locator daemon receives the locate request, it searches the implementation repository for the target object's POA. In this case, it finds that the `ozzy` POA is replicated across three servers that run on `zep`, `floyd`, and `cream`.

3. The locator daemon uses the load-balancing algorithm that is associated with the `ozzy` POA to determine which POA replica should handle the request—for example, the replica on `zep`.
4. The locator daemon obtains the address to the `ozzy` POA on `zep`, and returns a *direct object reference* that contains this address to the requesting client's ORB.
5. The client's ORB sends another locate request for the object, this time with the direct object reference, to `zep`. The replica confirms the object's existence with an `object-here` reply.
6. When the client ORB receives the `object-here` reply, it resends the client's request to the object instantiated in the `ozzy` replica on `zep`.

Except for the original invocation, all steps in this process are transparent to the client. Thus, clients can invoke on a server in exactly the same way, whether it exists alone or as a replica within a replicated server.

Automatic Replica Failover

Replica Failure

If a replica becomes unavailable—for example, because of machine or network failure—another replica enables clients to access the same objects as follows:

1. As soon as a direct object reference fails, the client ORB retrieves the object's original IOR, and sends a locate request to the locator daemon.
2. The locator daemon reapplies the load balancing algorithm for the target POA against the remaining viable replicas, to determine which one should handle requests on this object. It then returns a direct object reference to the client for the chosen replica.
3. All client invocations on the object, including the forwarded one, are handled by the new replica.

Replica restoration

If a failed replica is restored, it can transparently rejoin the replicated server by reregistering its address with the locator daemon. The locator daemon reassociates that replica with the name of the replicated POA in its database, thus making that replica available for subsequent client requests.

Restarting on a different host

A replica must be restarted on the host with which it is registered. If the failed replica needs to be restarted on a different host, you must modify the replicas registration using the following command:

```
itadmin process modify -node_daemon <new-node-daemon> <process>
```

Because persistent object references are addressed initially to the locator daemon, it is always safe to remove replicas from a replicated server and add new ones at runtime, without affecting client invocations.

Direct Persistence and Replica Failover

Overview

The failover mechanism described thus far relies upon the locator daemon to forward persistent object references from a failed replica to another replica that is still active. However, you can also create a persistent POA that circumvents the overhead of a locator daemon. This POA publishes persistent object references that embed a well-known address—that is, the address where the POA listens for incoming requests.

Requirements

To ensure failover in a replicated POA with direct persistence, the following requirements apply:

- The well-known address list that each replica obtains from its configuration must specify all addresses for each replica, including its own. Thus, the object references published by each replica must list the addresses of all replicas.
- The well-known address list for a given replica must always single out one address as its listening address. In the IORs that it generates, all other addresses are for publication only.

When a client request uses a direct object reference, it is directed to the first replica address in the list. If that replica is not available, it tries the next replica in the list, and so on, until it finds an available replica.

Example configuration

For example, given replicas that are instantiated on `host1` and `host2`, you can create the following configuration for each replica as follows:

```
MyConfigApp {
    ...
    wka_1:iiop:addr_list=["host1.com:1075", "+host2.com:2075"];
    wka_2:iiop:addr_list="+host1.com:1075", "host2.com:2075"];
    ...
}
```

The plus (+) prefix indicates that an address is for publication only in the IOR; a non-prefixed address is for publication and listening. Each POA replica obtains a different listening address as follows:

- The replica on `host1` specifies well-known address prefix `wka_1`, so it listens on the non-prefixed address `host1.com:1075`.
- The replica on `host2` specifies well-known address prefix `wka_2`, so it listens on the non-prefixed address `host2.com:2075`.

Note: For full details of all configuration required for direct persistence and well-know addressing, see "Setting direct persistence in configuration only" on page 74.

Example server code

The server code shown earlier is modified on each host as follows:

C++

```
// on host1:
// ...
CORBA::Any well_known_addressing_policy_value;
well_known_addressing_policy_value <<=
    CORBA::Any::from_string("wka_1", IT_TRUE);

// ...

policies[3] = orb->create_policy(
    IT_CORBA::WELL_KNOWN_ADDRESSING_POLICY_ID,
    well_known_addressing_policy_value );


// on host2:
// ...
CORBA::Any well_known_addressing_policy_value;
well_known_addressing_policy_value <<=
    CORBA::Any::from_string("wka_2", IT_TRUE);

// ...

policies[3] = orb->create_policy(
    IT_CORBA::WELL_KNOWN_ADDRESSING_POLICY_ID,
    well_known_addressing_policy_value );
```

Java

```
//on host1:
// ...
PersistenceModePolicyValueHelper.insert(
    persistent_mode_policy_value,
    PersistenceModePolicyValue.DIRECT_PERSISTENCE);
well_known_addressing_policy_value.insert_string(
    "wka_1");
// ...


//on host2:
// ...
PersistenceModePolicyValueHelper.insert(
    persistent_mode_policy_value,
    PersistenceModePolicyValue.DIRECT_PERSISTENCE);
well_known_addressing_policy_value.insert_string(
    "wka_2");
// ...
```

The object references for both replicas contain the same address list. Thus, requests on these IORs are first directed to `host1` address. If the replica on `host1` is unavailable, the request is redirected to the address on `host2`.

Building a Replicated Server

Overview

The following sections walk you through the process of building a replicated server, including the ability to load balance clients across multiple servers, activate multiple servers in response to a single client request, and dynamically change replicas in a replicated server.

Sample code

These examples are based on several demos in the Orbix `demos\corba\enterprise\clustering` directory. These demos consist of a simple client and server. The server program exports a single object, `SimpleClusteredObject`, which has the following interface:

```
module Clustering
{
    interface SimpleClusteredObject
    {
         string
         server_name();
    };
};
```

`SimpleClusteredObject` has a single operation, `server_name()`, which returns the name of the server as passed on the server command line. This is used to demonstrate the Orbix load-balancing features. Each server that runs the simple object is passed a different server name on the command line. Clients that connect to the object get and display the server name, thereby showing the server that they have been connected to.

Example 1: Building a Replicated Server to Start on Demand

The following example shows how to register a replicated server for on-demand activation in a location domain.

1. Build the application. For example:

```
$ cd C:\Program Files\Micro Focus\Orbix\asp\6.3\demos\corba\
   enterprise\clustering
$ nmake -e all
```

2. Start an `itadmin` session, and use the `process create` command to create an entry in the implementation repository for each replica in a replicated server:

```
$ itadmin
% process create \
    -pathname
      /opt/microfocus/asp/version/demos/enterprise/ \
      clustering/cxx_server/server \
      -node_daemon daemon_name \
      -startupmode on_demand \
      -args "--ORBname demos.clustering.server_1 server_1" \
      demos.clustering.server_process_1
%
% process create \
      # same arguments as before \
        ... \
      -args "--ORBname demos.clustering.server_2 server_2"\
      demos.clustering.server_process_2
%
% process create \
      ... same arguments as before \
      -args "--ORBname demos.clustering.server_3 server_3" \
      demos.clustering.server_process_3
%
```

These `process create` commands create entries for three servers to start on demand. This command requires the following arguments:

- The path name for the server executable.
- The name of the node daemon to start the server.

> **Note:** The server must always be started on the same host as its associated node daemon. Otherwise, you will receive a `PROCESS_IN_DIFFERENT_NODE_DAEMON` exception.

- A list of command line arguments passed to the server using the `-args` argument. These include a unique ORB name that is associated with each server replica.

3. Call `orbname create` to associate an ORB name with each server instance. The `-process` argument associates the new ORB name with the corresponding process name created in step **3**. The process name must be the same one that specified the new ORB name:

```
% orbname create \
    -process demos.clustering.server_process_1 \
     demos.clustering.server_1
% orbname create
    -process demos.clustering.server_process_2 \
     demos.clustering.server_2
% orbname create \
    -process demos.clustering.server_process_3 \
     demos.clustering.server_3
```

4. Call `poa create` to create a replicated POA, supplying two arguments:
   - The `-replicas` argument replicates the POA `ClusterDemo` on the three ORB names created in step **3**.
   - The `-load_balancer` argument specifies the load-balancing strategy to associate with the replicated POA; this tells the locator daemon how to route requests to the POA replicas. In this case, the `random` strategy is specified, which routes requests randomly among the POA's available replicas.

```
$ itadmin
% poa create -replicas demos.clustering.server_1, \
    demos.clustering.server_2, demos.clustering.server_3 \
    -load_balancer random ClusterDemo
```

5. Run the servers.

   Each server is passed an `-ORBname` parameter to identify the server. This parameter is passed to `ORB_init()`, which passes it on to the locator to identify the server when it creates the POA. Each of the servers must also be passed a server name parameter (for example, `server_1`), which is returned to the client to identify the server.

   The following shows how you might run these servers.

```
$ # cd $IT_PRODUCT_DIR/asp/version/demo/clustering
$ ./server -ORBname demos.clustering.server_1 server_1
    ../object.ior &
$ ./server -ORBname demos.clustering.server_2 server_2 &
$ ./server -ORBname demos.clustering.server_3 server_3 &
```

6. Run the client against the server.

   The client output shows how the locator randomly selects a server for each client that is running, load balancing the clients across the set of servers. If you kill one of the servers, the locator continues to forward clients to the remaining two servers, choosing between them at random.

Example 2: Updating a Replicated Server

Orbix replication is implemented so that you can add new servers on-the-fly without shutting down your system. The following commands add a server replica to the set already registered in the `clustering` demo:

Example 2: *Commands for Updating a Replicated Server*

```
1  process create \
       -pathname $server_name \
       -node_daemon $daemon_name \
       -startupmode on_demand \
       -args "--ORBname demos.clustering.server_4 server_4" \
    demos.clustering.server_process_4
2  orbname create
       -process demos.clustering.server_process_4
    demos.clustering.server_4
3  poa modify \
       -replicas \
          demos.clustering.server_1, \
          demos.clustering.server_2, \
          demos.clustering.server_3, \
          demos.clustering.server_4 \
    ClusterDemo
```

1. `process create` registers a new location domain process, `demos.clustering.server_process_4`.
2. `orbname create` associates a new ORB name, `demos.clustering.server_4`, with the new process.
3. `poa modify` redefines the `ClusterDemo` POA, specifying a fourth POA replica to run in the `demos.clustering.server_4` ORB.

After following these steps, run the clients against the server again. As before, the client output shows how the locator randomly selects a server for each client that is running, and eventually prints out the name of the fourth server.

Example 3: Dynamically Changing the Load Balancing Algorithm

Orbix enables you to dynamically change the load-balancing algorithm used for a replicated POA. Orbix supports the following load-balancing algorithms:

`round_robin`	The locator uses a round-robin algorithm to select from the list of active servers—that is, the first client is sent to the first server, the second client to the second server, and so on
`random`	The locator randomly selects an active server to handle the client.

For example, you can change the load-balancing algorithm used by the `clustering` demo by issuing the following `itadmin poa modify` command:

```
$ itadmin poa modify -load_balancer round_robin ClusterDemo
```

You can verify this by running several clients. The names of the servers now print out in the order in which they were started.

Per-Request Load Balancing

By default the locator load balancing is performed on a per-client ORB basis therefore once a binding to a replica has been established all requests from that ORB use the initial binding. You can choose to select an option that will allow load balancing to occur within the ORB, on a per-request basis.

To activate per-request load balancing set the policy in the configuration file as follows:

```
policies:per_request_lb = "true"
```

Replicating Orbix Services

Overview

Clients that use replicated Orbix services, such as the locator, are automatically routed to the first available server. If a server fails, clients are transparently rerouted to another server. Orbix services are normally replicated across a number of hosts, but it is also possible to replicate services on the same host.

The following Orbix services can be replicated:

- Locator daemon.
- Naming service.
- Configuration repository (CFR).
- Security service.

Figure 13 shows an example of a replicated naming service. This shows updates being pushed across from the master naming service to the slave naming service.

Replicating locator daemon and naming service

Continuous availability is especially important for the locator daemon and naming service. Replicating these services ensures that:

- Clients can always access persistent servers.
- New persistent servers can be activated on demand.
- `itadmin` commands that read the implementation repository always work (for example, `itadmin poa list`, and `itadmin process show`).
- Clients can always obtain object references from the naming service.

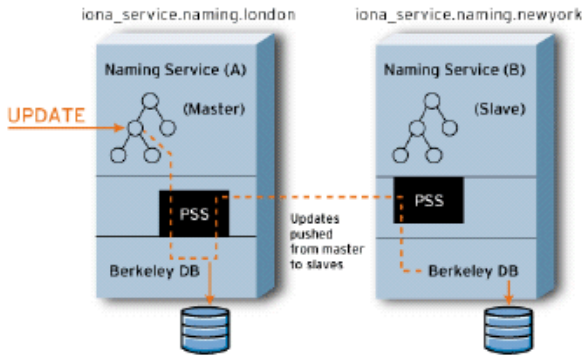

Figure 13: *Replicated Naming Service*

CFR-based versus file-based replication domains

Orbix services can be replicated in both CFR-based domains and in configuration file-based domains.

In a CFR-based domain, it is recommended that the CFR service is replicated, in addition to any other replicated services (for example, the security service). This ensures that all clients and servers can continue to run in the event of a failure.

Replicating the security service

In a secure domain, replicating the security service is important to ensure that all services are accessible even in the event of a host failure.

To replicate the security service, use the Orbix Configuration GUI tool to specify a replica host, like with other services (see the *Orbix Deployment Guide*). The generated configuration will contain the all relevant CORBA clustering information. However, with the security service, you must also edit your `is2.properties` file, and create a `cluster.properties` file. For details on these files, see the *Orbix Security Guide*.

Master and slave replicas

The locator daemon, naming service, and configuration repository use the persistent state service (PSS) to replicate their state. The PSS uses a master-slave model where a single replica is designated the master, and can process both read and write operations. All other replicas are slaves and can only process read operations. For more details, see "Master-Slave Replication" on page 98.

Note: All replicas in a PSS-based replicated service must be run on identical operating systems.

Adding and removing replicas

New server replicas can be added dynamically into a running system, and existing replicas can also be removed. For more details, see the *Orbix Deployment Guide*.

Master-Slave Replication

Overview

In PSS master-slave replication, one replica is designated as the master, and the remaining replicas are designated as slaves. Only the master can perform both read and write access, while slave replicas provide read-only access. In addition, only the master can process any read operation that is part of a distributed transaction.

If a slave replica receives a write or a read request in a distributed transaction, this request is either delegated to the master, or rejected if there is no master available. If the master fails, the remaining slaves hold an election to determine the new master. The automatic promotion of a slave to master is transparent to clients. This section includes the following:

- "Startup of master-slave services".
- "Master election protocol".
- "Setting replica priorities".
- "Setting master heartbeats".
- "Setting a refresh master interval".
- "Relaxing majority rule".
- "Replica administration".

Startup of master-slave services

When a group of replicated services has been deployed, all services are started as slaves. A majority of a service's replicas must have started before an election to select the master replica can take place.

This means, for example, in a replica group with four replicas (including the master), that at least three replicas must be running before an election can take place and write requests are possible.

Having a majority of replicas running ensures that a network partition can not result in duplicate masters. It also guarantees that previously committed updates are not lost.

Master election protocol

When the master is unavailable, an election protocol is used to determine the new master. If a majority of replicas are running, the slave that is most up-to-date with updates from the master is elected as the new master. If there is a tie, a priority system is used to elect the master. If there is still a tie, a random selection is made.

To support the automatic promotion of a slave, the minimum number of replicas in a group is three (one master and two slaves). For more details, see "Relaxing majority rule".

Setting replica priorities

You can configure the priority of a replica in elections using the following configuration variable:

```
plugins:pss_db:envs:env-name:replica_priority = "1";
```

The default value is `1`. Higher values mean a higher priority, and a priority of `0` means that slave is not to be promoted. For more details, see `plugins:pss_db:envs:env-name` in the *Orbix Configuration Reference*.

By default, the first replica deployed is given a higher priority than the remaining replicas. This increases the likelihood that the first replica runs as master when the services are started. This avoids unnecessary delegation for write operations.

Replica priorities are more likely to be honoured if services are shutdown cleanly (using the `stop_`*domain_name*`_services` command).

Setting master heartbeats

Slave replicas monitor the health of the master using periodic heartbeat messages. This enables a slave to be promoted in a timely manner. You can configure the interval between these heartbeats using the following configuration variable:

```
plugins:pss_db:envs:env-name:master_heartbeat_interval= "10";
```

The Orbix Configuration tool (`itconfigure`) sets the variable for each service to 30 seconds. For example, the setting for the locator daemon is:

```
plugins:pss_db:envs:it_locator:master_heartbeat_interval = "30";
```

For more details, see `plugins:pss_db:envs:env-name` in the *Orbix Configuration Reference*.

If is it necessary to disable heartbeats, you can set this variable to `0` (for example, to reduce network traffic). Disabling heartbeats means that the election of a new master normally occurs only when a slave attempts to delegate a request to the failed master.

Setting a refresh master interval

Each of the replicated Orbix services that use PSS replication enable you to configure the amount of time that a slave replica waits for a new master to be elected:

```
plugins:naming:refresh_master_interval
plugins:locator:refresh_master_interval
plugins:config_rep:refresh_master_interval
```

This interval specifies the maximum number of seconds that a write request is blocked at a slave while waiting for a master to be elected. For example, to set a time limit on the naming service to 30 seconds:

```
plugins:naming:refresh_master_interval = "30";
```

For more details, see the following sections in the *Orbix Configuration Reference*:

```
plugins:naming
plugins:locator
plugins:config_rep
plugins:pss_db:envs:env-name
```

Relaxing majority rule

To promote a slave, a majority of replicas must be running. This means that in a replica group with two replicas (one master and one slave), the slave can never be promoted. As a special case, it is possible to allow the slave to be promoted. You can do this by setting the following variable to `true`:

```
plugins:pss_db:envs:env-name:allow_minority_master = "true";
```

For more details, see `plugins:pss_db:envs:env-name` in the *Orbix Configuration Reference*.

Note: Setting `allow_minority_master` to `true` means that it is possible for duplicate masters to exist if there is a network partition. It also means that updates may be lost if services are started in different orders. To minimize the possibility of this, perform the following steps:

1. Only set the `allow_minority_master` variable to true on one replica (the one most likely to be the slave).
2. The replica with this variable set to true should always be started second.
3. If the master fails, and the slave is promoted, the previous master must be restarted only when the new master is running.

Replica administration

The `itadmin` tool provides several commands to examine the state of replicated services:

```
itadmin ns list_servers
itadmin ns show_server
itadmin locator list_servers
itadmin locator show
itadmin config list_servers
itadmin config show_server
itadmin pss_db list_replicas
itadmin pss_db show
```

For more details on these `itadmin` commands, see the following:

- "Naming Service" on page 339.
- "Location Domain" on page 299.
- "Configuration Domain" on page 271.
- "Persistent State Service" on page 375.

In addition, for details on administration of PSS databases, see "Managing Orbix Service Databases" on page 149.

Active Connection Management

Overview

Orbix active connection management lets servers scale up to large numbers of clients without encountering connection limits. Using active connection management, Orbix recycles least recently used connections as new connections are required.

You can control active connection management in Orbix with configuration variables, that specify the maximum number of incoming and outgoing client–server connections. Two settings are available for both client-side and server-side connections:

- A hard limit specifies the number of connections beyond which no new connections are permitted.
- A soft limit specifies the number of connections at which Orbix begins closing connections.

Setting incoming server-side connections

To limit the number of incoming server-side connections, set the following configuration variables:

plugins:iiop:incoming_connections:hard_limit specifies the maximum number of incoming (server-side) connections permitted to IIOP. IIOP does not accept new connections above this limit. This variable defaults to `-1` (disabled).

plugins:iiop:incoming_connections:soft_limit specifies the number of connections at which IIOP starts closing incoming (server-side) connections. This variable defaults to `-1` (disabled).

For example, the following file-based configuration entry sets a server's hard connection limit to `1024`:

```
plugins:iiop:incoming_connections:hard_limit=1024;
```

The following `itadmin` command sets this variable:

```
itadmin variable create -type long -value 1024
      plugins:iiop:incoming_connections:hard_limit
```

Setting outgoing client-side connections

To limit the number of outgoing client-side connections, set the following configuration variables:

plugins:iiop:outgoing_connections:hard_limit specifies the maximum number of outgoing (client-side) connections permitted to IIOP. IIOP does not allow new outgoing connections above this limit. This variable defaults to `-1` (disabled).

plugins:iiop:outgoing_connections:soft_limit specifies the number of connections at which IIOP starts closing outgoing (client-side) connections. This variable defaults to `-1` (disabled).

For example, the following file-based configuration entry sets a hard limit for outgoing connections to `1024`:

```
plugins:iiop:outgoing_connections:hard_limit=1024;
```

The following `itadmin` command sets this variable:

```
itadmin variable create -type long -value 1024
      plugins:iiop:outgoing_connections:hard_limit
```

Setting Buffer Sizes

Overview

If the IIOP buffer size within an ORB is configured to a sufficiently large number, fragmentation is not required by the ORB and does not occur. This section describes how to set the buffer size in the C++ and Java CORBA ORBs.

C++ configuration

`policies:<protocol-name>:buffer_sizes_policy:default_buffer_size`

This variable is used as the initial size for the buffer and also as the increment size if the buffer is too small.

For example, when sending a message of 60,000 bytes (including GIOP header overhead, remember depending on the types used by GIOP, this overhead may be large), if the `default_buffer_size value` is set to `10000`, the buffer is initially 10,000 bytes. The C++ ORB then sends out 6 message fragments of 10,000 bytes each. If the `default_buffer_size` value is set to `64000`, only one unfragmented message is sent out.

Java configuration

`policies:<protocol-name>:buffer_sizes_policy:default_buffer_size`

This variable is used as the initial size for the buffer unless it is less than the system defined minimum buffer size.

`policies:<protocol-name>:buffer_sizes_policy:max_buffer_size`

This value is used as the initial size for the buffer if smaller than `default_buffer_size`. For example, when sending a message with an overall size of 60,000 bytes, if the lower of the `buffer_size` values mentioned above is set to `10000`, the buffer is initially 10,000 bytes. The Java ORB then sends out 6 message fragments of 10,000 bytes each. If the lower of the `buffer_size` values mentioned above is set to `64000`, only one unfragmented message is sent out.

Note: These configuration settings apply to secure or non-secure IIOP, depending on whether the `iiop` or `iiop_tls` scope is used. For alignment purposes, buffer size values should be a multiple of 8 (i.e. 32,000 or 64,000).

Data fragmentation

For a CORBA ORB to be considered compliant with the OMG GIOP 1.1 specification, the ORB implementation must support data fragmentation.

Some CORBA ORB implementations do not support data fragmentation but claim GIOP 1.1 compliance. Orbix ORBs support fragmentation and are fully compliant with the GIOP 1.1 specification.

CHAPTER 6

Managing the Naming Service

The naming service lets you associate abstract names with CORBA objects in your applications, enabling clients to locate your objects.

The interoperable naming service is a standard CORBA service, defined in the Interoperable Naming Specification. The naming service allows you to associate abstract names with CORBA objects, and enables clients to find those objects by looking up the corresponding names. This service is both very simple and very useful. Most CORBA applications make some use of the naming service. Locating a particular object is a common requirement in distributed systems and the naming service provides a simple, standard way to do this. The naming service is installed by default as part of every Orbix installation.

In addition to naming service functionality, Orbix also provides naming-based load balancing, using *object groups*. An object group is a collection of objects that can increase or decrease in size dynamically. When a bound object is an object group, clients can resolve object names in a naming graph, and transparently obtain references to different objects.

In this chapter

This chapter contains the following sections:

Naming Service Administration

Overview

The naming service maintains hierarchical associations of names and object references. An association between a name and an object is called a *binding*. A client or server that holds a CORBA object reference *binds* a name to the object by contacting the naming service. To obtain a reference to the object, a client requests the naming service to look up the object associated with a specified name. This is known as *resolving* the object name. The naming service provides interfaces, defined in IDL, that enables clients and servers to bind to and resolve names to object references.

The naming service has an administrative interface and a programming interface. These enable administrators and programmers to create new bindings, resolve names, and delete existing bindings. For information about the programming interface to the naming service, see the *CORBA Programmer's Guide.*

Typical administration tasks

While most naming service operations are performed by programs, administrative tasks include:

- Controlling the naming service (for example, starting and stopping the naming service).
- Viewing naming information (for example, bindings between names and objects).
- Adding or modifying naming information that has not been properly maintained by programs. For instance, you might need to remove outdated information left behind by programs that have been moved or removed from the environment.

You can perform these tasks administratively with `itadmin` commands. This is especially useful when testing applications that use the naming service. You can use `itadmin` commands to create, delete, and examine name bindings in the naming service.

Name formats and naming graphs

Naming service names adhere to the CORBA naming service format for string names. You can associate names with two types of objects: a *naming context* or an *application object*. A naming context is an object in the naming service within which you can resolve the names of application objects.

Naming contexts are organized into a *naming graph*. This can form a naming hierarchy, much like that of a filing system. Using this analogy, a name bound to a naming context would correspond to a directory and a name bound to an application object would correspond to a file.

The full name of an object, including all the associated naming contexts, is known as a *compound name*. The first component of a compound name gives the name of a naming context, in which the second component is accessed. This process continues until the last component of the compound name has been reached.

A compound name in the CORBA naming service can take two forms:

- An IDL sequence of name components
- A human-readable `StringName` in the Interoperable Naming Service (INS) string name format

Naming Service Commands

`itadmin` provides commands for browsing and managing naming service information. Many naming service commands take a *`path`* argument. This specifies the path to the context or object on which the command is performed.

Note: Many of these commands take object references as command-line arguments. These object references are expected in the string format returned from `CORBA::ORB::object_to_string()`. By default, this string format represents an interoperable object reference (IOR).

For reference information about these `itadmin` commands, see "Naming Service" on page 339. The rest of this chapter uses `itadmin` commands to build an example naming graph and populate it with name bindings.

Controlling the Naming Service

Starting the naming service

You must start up the naming service on the machine where it runs. To start the naming service:

1. Log in as `root` on UNIX, or as `administrator` on Windows NT.
2. Open a terminal or command window.
3. Enter `itnaming run`
4. Do the following depending on your platform:

   Windows

   Leave the command window open.

   UNIX

   Leave the terminal window open, or push the process into the background and close the window.

Stopping the naming service

`itadmin ns stop` stops the naming service.

Building a Naming Graph

Overview

A naming context is an object in the naming service that can contain the names of application objects. Naming contexts are organized into a hierarchical naming graph. This section uses `itadmin` commands to build the naming graph shown in Figure 14.

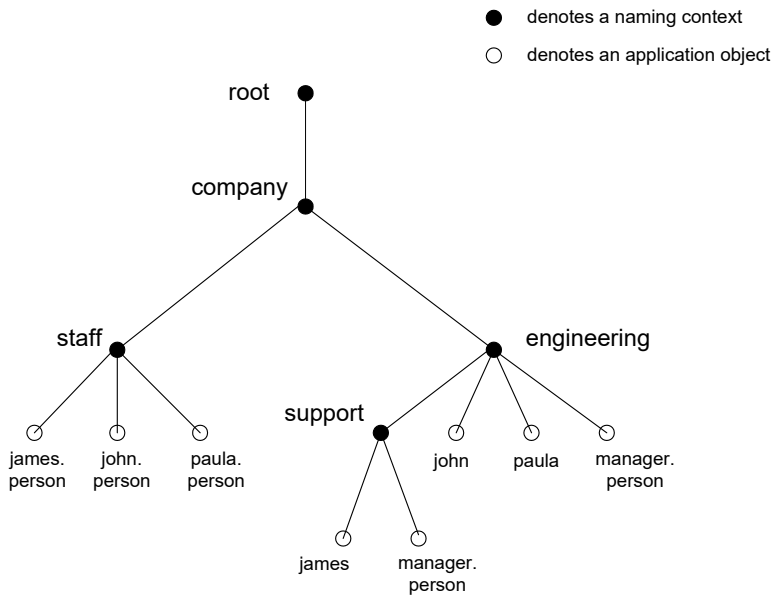

Figure 14: *Naming Context Graph*

Names are given in the INS string name format `id.kind` (for example, `john.person`). The `kind` component can be empty (for example, `john`). The combination of `id` and `kind` fields must unambiguously specify the name.

In this section

Using the example naming graph in Figure 14, this section explains the following tasks:

- Creating Naming Contexts.
- Creating Name Bindings.
- Listing name bindings.
- Finding object references by name.
- Removing name bindings.
- Rebinding a name to an object or naming context

Creating Naming Contexts

`itadmin ns newnc` provides the simplest way to create a naming context. This command takes an optional *path* argument, which takes the form of an INS string name. For example, the following command creates a new context that is bound to a simple name with an `id` of `company`, and an empty `kind` value:

```
itadmin ns newnc company
```

The following example creates a new naming context that is bound to the name `company/engineering`; the context `company` must already exist.

```
itadmin ns newnc company/engineering
```

The following example creates a new context that is bound to the name `company/engineering/support`; the context `company/engineering` must already exist.

```
itadmin ns newnc company/engineering/support
```

Creating an unbound naming context

You can also use `itadmin ns newnc` to create an unbound context. If the *path* argument is not specified, `itadmin ns newnc` prints the IOR to standard out. For example:

```
itadmin ns newnc
"IOR:000000000002356702b4944c3a6f6d672e6f7267...."
```

On UNIX, to bind the context created with `ns newnc`, use the `ns bind -context` command, as follows:

```
itadmin ns bind -c -path company/staff 'itadmin ns newnc'
```

This binds the new context to the name `company/staff`.

Creating Name Bindings

To bind a name to an object, use `itadmin ns bind -object`. Given the naming context graph shown in Figure 14 on page 113, this section assumes the application objects are associated with the following object reference strings:

```
james     IOR:0000000037e276f47a4b94874c64648e949...
john      IOR:0000028e276f47a40b9248474c64646F3E5...
paula     IOR:00000000569a2e8034b94874d6583f09e24...
```

You can bind these objects to appropriate names within the `company/staff` naming context, as follows:

```
itadmin ns bind -o -path company/staff/james.person
"IOR:0000000037e276f47a4b94874c64648e949..."

itadmin ns bind -o -path company/staff/john.person
"IOR:0000028e276f47a40b9248474c64646F3E5..."

itadmin ns bind -o -path company/staff/paula.person
"IOR:00000000569a2e8034b94874d6583f09e24..."
```

These commands assign a `kind` of `person` in the final component of each employee name.

`itadmin ns bind` takes an IOR from the command line. For example, on UNIX, if you have Paula's IOR in a file named `paula.ior`, you can bind it, as follows:

```
itadmin ns bind -o -path company/staff/paula.person 'cat
   paula.ior'
```

To build the naming graph further, create additional bindings that are based on the departments that employees are assigned to. The following example takes IORs from files printed to standard input.

```
itadmin ns bind -o -path
   company/engineering/support/james.person 'cat james.ior'

itadmin ns bind -o -path company/engineering/john.person 'cat
   john.ior'

itadmin ns bind -o -path company/engineering/paula.person 'cat
   paula.ior'
```

To enable an application to find the manager of a department easily, add the following bindings:

```
itadmin ns bind -o -path company/engineering/manager.person 'cat
   paula.ior'

itadmin ns bind -o -path
   company/engineering/support/manager.person 'cat paula.ior'
```

The following names now resolve to the same object:

```
company/staff/paula.person
company/engineering/paula.person
company/engineering/manager.person
company/engineering/support/manager.person
```

The naming contexts and name bindings created by this sequence of commands builds the complete naming graph shown in Figure 14 on page 113.

Maintaining a Naming Graph

Maintenance commands

After you create a naming graph, it is likely you will need to periodically modify its contents—for example, remove bindings, or to change the bindings for an object reference. Table 5 describes the `itadmin` commands that you can use to maintain naming contexts and bindings.

Table 5: *Naming Graph Maintenance Commands*

Command	Task
`ns list`	List all bindings in a naming context
`ns resolve`	Print the object reference for the application object or naming context to which a name is bound.
`ns unbind`	Unbind the binding for an object reference.
`ns remove`	Unbind and destroy a name binding.

Note: `unbind` and `remove` can be disabled by setting `plugins:naming:destructive_methods_allowed` to `false`.

Rebinding a name to an object or naming context

To change the binding for an object reference, perform the following steps:

1. Use `itadmin ns resolve to` obtain the object reference bound to the current path and write it to a file:

   ```
   itadmin ns resolve path > file
   ```

   The *`path`* argument takes the form of a string name.
2. Call `itadmin ns unbind` to unbind the current path:

   ```
   itadmin ns unbind path
   ```

3. Call `itadmin ns bind` to bind the saved object reference to the new path. For example, on UNIX:

   ```
   itadmin ns bind -c newpath 'cat file'
   ```

Managing Object Groups

Overview

An *object group* is a naming service object that provides transparent naming-based load balancing for clients. An object group contains application objects, and can increase or decrease in size dynamically when member objects are added or removed.

An object group object can be bound to a path in a naming graph like any other object. Each object group contains a pool of member objects associated with it. When a client resolves the path that an object group is bound to, the naming service returns one of the member objects according to the group's *selection policy.*

Creating an object group

You can create an object group using the `itadmin` commands in the following steps:

1. Create the object group using `itadmin nsog create` and specify the desired selection algorithm (see "Selection algorithms" on page 119).
2. Add application objects to the newly created object group using `itadmin nsog add_member` on it.
3. Bind an existing naming context to the object group using `itadmin nsog bind`.

When you create the object group, you must supply a group identifier. This identifier is a string value that is unique among other object groups.

Similarly, when you add a member to the object group, you must supply a reference to the object and a corresponding member identifier. The member identifier is a string value that must be unique within the object group.

Selection algorithms

Each object group has a selection algorithm that is set when the object group is created. This algorithm is applied when a client resolves the name associated with the object group. Three selection algorithms are supported:

- Round-robin
- Random
- Active load balancing

The naming service directs client requests to objects according to the group's selection algorithm.

Active load balancing

In an object group that uses active load balancing, each object group member is assigned a load value. The naming service satisfies client `resolve()` invocations by returning references to members with the lowest load values.

Default load values can be set administratively using the configuration variable `plugins:naming:lb_default_initial_load`. Thereafter, load values should be updated programmatically by periodically calling `ObjectGroup::update_member_load()`. `itadmin` provides an equivalent command, `nsog update_member_load`, in cases where manual intervention is required.

You should also set or modify member timeouts using `itadmin nsog set_member_timeout`, or programmatically using `ObjectGroup::set_member_timeout()`. You can configure default timeout values by updating `plugins:naming:lb_default_load_timeout`. If a member's load value is not updated within its timeout interval, its object reference becomes unavailable to client `resolve()` invocations. This typically happens because the object itself or an associated process is no longer running, and therefore cannot update the object's load value.

A member reference can be made available again to client `resolve()` invocations by resetting its load value using `ObjectGroup::update_member_load()` or `itadmin nsog update_member_load`. In general, an object's timeout should be set to an interval greater than the frequency of load value updates.

Commands

"Object Groups" on page 344 describes the `itadmin` commands that you can use to create and administer object groups.

Deploying Naming Service Replicas on z/OS

Overview

To deploy Naming Service replicas on z/OS, perform the following steps:

Performing initial setup

Before deploying Naming Service replicas, ensure that the first deploy job in *orbixhlq*.JCLLIB(DEPLOY1) has been run (where *orbixhlq* represents the high-level qualifier for your Orbix Mainframe installation). Additionally, submit the following JCL to ensure that the locator and node daemon are running:

- *orbixhlq*.JCLLIB(LOCATOR)
- *orbixhlq*.JCLLIB(NODEDAEM)

Updating configuration to accommodate replicas

By default, configuration information is stored in *orbixhlq*.DOMAINS(FILEDOMA). However, the member name for a user's configuration domain might be user-defined. The configuration domain contains a scope for the Naming Service, called `naming`. This scope must be updated, as follows, to accommodate replicas:

1. Ensure that a comment character (that is, `#`) precedes the `plugins:pss_db:envs:it_naming_store:replica_priority` configuration item. This will be defined instead in new replica scopes.
2. Ensure that a comment character (that is, `#`) precedes the `plugins:pss_db:envs:it_naming_store:db_home` configuration item. This will be defined instead in new replica scopes.
3. Add the configuration item `plugins:pss_db:envs:it_naming_store:allow_minority_master = "true";`.
4. Add a scope for the first Naming Service.

The following is an example of these updates. In this example, the first Naming Service has a scope called `replica1`:

```
…
naming
{
    event_log:filters = ["IT_NAMING=*",
                         "IT_PSS_DB=WARN+ERROR+FATAL"];

    configuration:hostname = "%{LOCAL_HOSTNAME}";

    plugins:naming:allow_nil = "true";
    plugins:naming:lb_default_load_timeout = "1000000000";
    plugins:naming:lb_default_initial_load = "0.0";

    plugins:pss_db:envs:it_naming_store:master_heartbeat_interval\
            = "30";
#   plugins:pss_db:envs:it_naming_store:replica_priority = "2";

#   plugins:pss_db:envs:it_naming_store:db_home
#           = "%{LOCAL_HFS_ROOT}/filedomain/dbs/naming";

    #
    # Settings for well known addressRun JCLLIB(NSLIST) - see 3 naming contextsing:
    # (mandatory is direct_persistence is enabled)
    #
    # LOCAL_NAMING_PORT =  5004;
    # plugins:naming_cluster:iiop_addr_list =
    #              ["+%{LOCAL_HOSTNAME}:%{LOCAL_NAMING_PORT}"];
    # plugins:naming:iiop:port = "%{LOCAL_NAMING_PORT}";
    # plugins:naming:iiop:host = "%{LOCAL_HOSTNAME}";

    plugins:naming:direct_persistence = "false";


    policies:iiop:server_address_mode_policy:local_hostname
            = "%{LOCAL_HOSTNAME}";

    plugins:orb:is_managed = "false";

    plugins:it_mgmt:managed_server_id:name
           = "iona_services.naming";

    plugins:pss_db:envs:it_naming_store:allow_minority_master = "true";
```

```
    replica1
    {
        plugins::pss_db:envs:it_naming_store:replica_priority = "2";

        plugins:pss_db:envs:it_naming_store:db_home
        = "%{LOCAL_HFS_ROOT}/filedomain/dbs/naming_replica1";
        plugins:orb:is_managed = "false";
        plugins:it_mgmt:managed_server_id:name
               = "iona_services.naming.replica1";
    };
    …
};
```

Deploying the first Naming Service

Follow these steps to deploy the first Naming Service:

1. Create an `NSARGS1` member in the *`orbixhlq`*`.CONFIG` PDS. The *`orbixhlq`*`.CONFIG(NSARGS1)` member should typically consist of the following:

   ```
   -ORBdomain_name DEFAULT@
   -ORBname iona_services.naming.replica1
   ```

   Note: If your configuration domain name is a user-defined name other than `DEFAULT@`, ensure that the correct name is specified. Also, ensure that the value specified for `-ORBname` references the scope that is defined in the configuration for the first Naming Service.

2. Create a JCL member called `DEPNS1` in the *`orbixhlq`*`.JCLLIB` PDS. The `DEPNS1` JCL should be as follows (where *`orbixhlq`* represents the high-level qualifier for your Orbix Mainframe installation):

   ```
   //DEPNS1   JOB   (),
   //         CLASS=A,
   //         MSGCLASS=X,
   //         MSGLEVEL=(1,1),
   //         NOTIFY=&SYSUID,
   //         REGION=0M,
   //         TIME=1440,
   //         COND=(0,NE)
   //*
   //         JCLLIB ORDER=(orbixlq.PROCLIB)
   //         INCLUDE MEMBER=(ORXVARS)
   //*
   ```

```
//*************************************************************
//* JCL to deploy first naming service
//* Requires locator and node_daemon to be running
//*************************************************************
//*
//* Make the following changes before running this JCL
//*
//* 1.  If you ran DEPLOY1 (or DEPLOYT) to configure in a domain
//*     other than the default, please ensure that dataset
//*     &ORBIXCFG(ORBARGS) has the domain name used by DEPLOY1
//*     (or DEPLOYT).
//*
//*************************************************************
//*
//* Prepare the Naming Service
//*
//PREPNAM   EXEC PROC=ORXG,
//          PROGRAM=ORXNAMIN,
//         PPARM='prepare -publish_to_file=DD:ITCONFIG(IORNAM) '
//ORBARGS   DD DSN=&ORBIXCFG(NSARGS1),DISP=SHR
//*
//* Update configuration domain with naming service IOR
//*
//ITCFG1    EXEC ORXADMIN
//SYSIN  DD *
    variable modify \
      -type string \
      -value  --from_file:6 //DD:ITCONFIG(IORNAM) \
    NS_1
/*
//ORBARGS  DD DSN=&ORBIXCFG(ORBARGS),DISP=SHR
//*
//* Create a named key for the Naming Service
//*
//ITCFG2   EXEC ORXADMIN
//SYSIN DD *
    variable modify \
      -type string \
      -value  --from_file:3 //DD:ITCONFIG(IORNAM) \
    LOCAL_NAMING_REFERENCE
/*
//ORBARGS  DD DSN=&ORBIXCFG(ORBARGS),DISP=SHR
//*
//ITCFG3   EXEC ORXADMIN
//SYSIN DD *
```

```
    named_key create \
      -key NameService \
      --from_file:6 //DD:ITCONFIG(IORNAM)
/*
//ORBARGS  DD DSN=&ORBIXCFG(ORBARGS),DISP=SHR
```

3. Run the *orbixhlq*.JCLLIB(DEPNS1) JCL to deploy the first Naming Service.
4. Create a JCL member called MNSPOAS1 in the *orbixhlq*.JCLLIB PDS. The MNSPOAS1 JCL should be as follows (where *orbixhlq* represents the high-level qualifier for your Orbix Mainframe installation):

```
//REP1     JOB   (),
//         CLASS=A,
//         MSGCLASS=X,
//         MSGLEVEL=(1,1),
//         NOTIFY=&SYSUID,
//         REGION=0M,
//         TIME=1440,
//         COND=(0,NE)
//*
//         JCLLIB ORDER=(orbixlq.PROCLIB)
//         INCLUDE MEMBER=(ORXVARS)
//*
//* Make the following changes before running this JCL:
//* 1.  If you ran DEPLOY1 (or DEPLOYT) to configure in a domain
//*     other than the default, please ensure that dataset
//*     HLQ.ORBIX63.CONFIG(ORBARGS) has the domain
//*     name used by DEPLOY1 (or DEPLOYT).
//*
//MFARELD EXEC ORXADMIN
//SYSIN DD *
 poa modify -allowdynreplicas  yes IT_NamingContextExt
 poa modify -allowdynreplicas  yes IT_ObjectGroupFactory
 poa modify -allowdynreplicas  yes IT_ObjectGroup
 poa modify -allowdynreplicas yes \
     IT_NamingServiceAdmin_iona_services.naming.replica1
 poa modify -allowdynreplicas yes \
     IT_MasterNamingContextExt_iona_services.naming.replica1
 poa modify -allowdynreplicas yes \
     IT_MasterObjectGroupFactory_iona_services.naming.replica1
 poa modify -allowdynreplicas yes \
     IT_MasterObjectGroup_iona_services.naming.replica1
/*
//ORBARGS  DD DSN=&ORBIXCFG(ORBARGS),DISP=SHR
```

Note: The itadmin commands in the preceding example make reference to the first Naming Service scope, which is `replica1` in this case. These commands must match the Naming Service scope that is defined in the configuration.

5. Run the *`orbixhlq`*`.JCLLIB(MNSPOAS1)` JCL to allow for naming service replicas.
6. Update the configuration in *`orbixhlq`*`.DOMAINS(FILEDOMA)` to make use of the first Naming Service, as follows:

```
#IT_NameServiceReplicas =
#  ["iona_services.naming=%{LOCAL_NAMING_REPLICA_REFERENCE}"];
IT_NameServiceReplicas =
[
     "iona_services.naming.replica1=%{NS_1}
];
```

As shown in the preceding example:

- Ensure that the existing `IT_NameServiceReplicas` configuration item is preceded by a comment character.
- Ensure that the specified ORBname matches the name of the configuration scope for the first Naming Service. In this example, the name is `iona_services.naming.replica1`.
- Ensure that the specified configuration value, represented by `NS_1` in this case, matches the value used in *`orbixhlq`*`.JCLLIB(DEPNS1)` to deploy the first Naming Service. The `ITCFG1` step of the `DEPNS1` JCL updates this configuration variable.

7. Create a JCL member called `NSREP1` in the *`orbixhlq`*`.JCLLIB` PDS. The `NSREP1` JCL should be as follows (where *`orbixhlq`* represents the high-level qualifier for your Orbix Mainframe installation):

```
//NS1      JOB   (),
//         CLASS=A,
//         MSGCLASS=X,
//         MSGLEVEL=(1,1),
//         NOTIFY=&SYSUID,
//         REGION=0M,
//         TIME=1440,
//         COND=(0,NE)
//*
//         JCLLIB ORDER=(orbixlq.PROCLIB)
//         INCLUDE MEMBER=(ORXVARS)
//*
//* Run the Orbix Naming service
//*
//* Make the following changes before running this JCL:
//*
//* 1.  Change SET DOMAIN='DEFAULT@' to your configuration
//*     domain name.
//*
//           SET DOMAIN='DEFAULT@'
//*
//GO  EXEC PROC=ORXG,
//    PROGRAM=ORXNAMIN,
//    PPARM='run -ORBname iona_services.naming.replica1'
//ITDOMAIN DD DSN=&ORBIXCFG(&DOMAIN),DISP=SHR
```

Note: Ensure that the specified ORBname matches the name of the configuration scope for the first Naming Service. In the preceding example, the name is `iona_services.naming.replica1`.

8. Run the *`orbixhlq`*`.JCLLIB(NSREP1)` JCL to start the first Naming Service.

Deploying additional replicas

Follow these steps for each additional replica you want to deploy:

Note: In this example, the replica is called `replica2`.

1. In *orbixhlq*.DOMAINS(FILEDOMA), add a sub-scope for the Naming Service replica within the `naming` scope, as follows:

```
replica1
…
replica2
{
    plugins:pss_db:envs:it_naming_store:replica_priority = "3";
    plugins:pss_db:envs:it_naming_store:prevent_unilateral_promotion = "true";

    plugins:pss_db:envs:it_naming_store:db_home
    = "%{LOCAL_HFS_ROOT}/filedomain/dbs/naming_replica2";

    plugins:orb:is_managed = "false";
    plugins:it_mgmt:managed_server_id:name = "iona_services.naming.replica2";
};
…
```

   In this case, set the first three configuration items as appropriate for the replica you want to deploy.

2. Create a JCL member called `NSARGS2` in the *orbixhlq*.CONFIG PDS. The `NSARGS2` JCL should be as follows:

```
-ORBdomain_name DEFAULT@
-ORBname iona_services.naming.replica2
```

   Note: Ensure that the specified ORBname matches the name of the configuration scope for the Naming Service replica. In the preceding example, the name is `iona_services.naming.replica2`.

3. Create a JCL member called `DEPNS2` in the *`orbixhlq`*`.JCLLIB` PDS. The `DEPNS2` JCL should be as follows:

```
//DEPNS2   JOB   (),
//         CLASS=A,
//         MSGCLASS=X,
//         MSGLEVEL=(1,1),
//         NOTIFY=&SYSUID,
//         REGION=0M,
//         TIME=1440,
//         COND=(0,NE)
//*
//         JCLLIB ORDER=(orbixlq.PROCLIB)
//         INCLUDE MEMBER=(ORXVARS)
//*
//*****************************************************************
//* JCL to deploy first naming service
//* Requires locator and node_daemon to be running
//*****************************************************************
//*
//* Make the following changes before running this JCL
//*
//* 1.  If you ran DEPLOY1 (or DEPLOYT) to configure in a domain
//*     other than the default, please ensure that dataset
//*     &ORBIXCFG(ORBARGS) has the domain name used by DEPLOY1
//*     (or DEPLOYT).
//*
//*****************************************************************
//*
//* Prepare the Naming Service
//*
//PREPNAM   EXEC PROC=ORXG,
//          PROGRAM=ORXNAMIN,
//          PPARM='prepare -publish_to_file=DD:ITCONFIG(IORNAM) '
//ORBARGS   DD DSN=&ORBIXCFG(NSARGS2),DISP=SHR
//*
//* Update configuration domain with naming service IOR
//*
//ITCFG1    EXEC ORXADMIN
//SYSIN  DD *
    variable modify \
      -type string \
      -value  --from_file:6 //DD:ITCONFIG(IORNAM) \
    NS_2
/*
//ORBARGS  DD DSN=&ORBIXCFG(ORBARGS),DISP=SHR
```

4. Run the *orbixhlq*.JCLLIB(DEPNS2) JCL to deploy the naming service replica.
5. Create a JCL member called MNSPOAS2 in the *orbixhlq*.JCLLIB PDS. The MNSPOAS2 JCL should be as follows (where *orbixhlq* represents the high-level qualifier for your Orbix Mainframe installation):

```
//REP2     JOB   (),
//         CLASS=A,
//         MSGCLASS=X,
//         MSGLEVEL=(1,1),
//         NOTIFY=&SYSUID,
//         REGION=0M,
//         TIME=1440,
//         COND=(0,NE)

//*
//         JCLLIB ORDER=(orbixlq.PROCLIB)
//         INCLUDE MEMBER=(ORXVARS)
//*
//* Make the following changes before running this JCL:
//*
//* 1.  If you ran DEPLOY1 (or DEPLOYT) to configure in a domain
//*     other than the default, please ensure that dataset
//*     HLQ.ORBIX63.CONFIG(ORBARGS) has the domain
//*     name used by DEPLOY1 (or DEPLOYT).
//*
//MFARELD EXEC ORXADMIN
//SYSIN DD *
 poa modify -allowdynreplicas yes \
     IT_NamingServiceAdmin_iona_services.naming.replica2
 poa modify -allowdynreplicas yes \
     IT_MasterNamingContextExt_iona_services.naming.replica2
 poa modify -allowdynreplicas yes \
     IT_MasterObjectGroupFactory_iona_services.naming.replica2
 poa modify -allowdynreplicas yes \
     IT_MasterObjectGroup_iona_services.naming.replica2
/*
//ORBARGS  DD DSN=&ORBIXCFG(ORBARGS),DISP=SHR
```

6. Run the *orbixhlq*.JCLLIB(MNSPOAS2) JCL to allow for the naming service replica.

Performing post-deployment setup

Follow these steps to perform post-deployment setup:

1. After all naming service replicas have been deployed, stop the Naming Service.
2. Edit *orbixhlq*.DOMAINS(FILEDOMA) to update the IT_NameServiceReplicas configuration item as follows:

```
ITNameServiceReplicas =
[
    "iona_services.naming.replica1=%{NS_1}",
    "iona_services.naming.replica2=%{NS_2}",
    "iona_services.naming.replica3=%{NS_3}"
];
```

Note: The preceding example assumes that the first Naming Service and two replicas were deployed.

Running the replicas

Create a separate JCL member in the *orbixhlq*.JCLLIB PDS for each naming service replica. In each case, the JCL should be similar to the following:

```
//NS1 JOB (),
// CLASS=A,
// MSGCLASS=X,
// MSGLEVEL=(1,1),
// NOTIFY=&SYSUID,
// REGION=0M,
// TIME=1440,
//*
// JCLLIB ORDER(orbixhlq.PROCLIB)
// INCLUDE MEMBER=(ORXVARS)
//*
//* Run the Orbix Naming Service
//*
//* Make the following changes before running this JCL:
//*
//* 1. Change SET DOMAIN='DEFAULT@' to your configuration
//* domain name.
//*
// SET DOMAIN='DEFAULT@'
```

```
//*
//GO EXEC PROC=ORXG,
// PROGRAM=ORXNAMIN,
// PPARM='run -ORBname iona_services.naming.replica1'
//ITDOMAIN DD DSN=&ORBIXCFG(&DOMAIN),DISP=SHR
```

Ensure that the JCL for each replica specifies the correct ORBname for that replica. For example, the preceding example specifies an ORBname of `iona_services.naming.replica1`. Then submit each JCL member to run each of the naming service replicas.

CHAPTER 7

Managing an Interface Repository

An interface repository stores information about IDL definitions, and enables clients to retrieve this information at runtime. This chapter explains how to manage the contents of an interface repository.

In this chapter

This chapter contains the following sections:

Interface Repository

Overview

An interface repository maintains information about the IDL definitions implemented in your system. Given an object reference, a client can use the interface repository at runtime to determine the object's type and all information about that type. Clients can also browse the contents of an interface repository. Programmers can add sets of IDL definitions to an interface repository, using arguments to the IDL compiler command.

Interface repository administration

An interface repository database is centrally located. When Orbix environments have more than one interface repository, they are often organized so that each application or set of related applications uses a common interface repository. When an interface repository has been configured, it requires minimal administrative intervention. Typical tasks include stopping and restarting the interface repository, when necessary, removing outdated definitions, when applications are removed, and troubleshooting, when necessary.

This chapter provides information for administrators on how start and stop the interface repository. It also provides information for programmers on how to add, examine, and remove IDL definitions.

For details on advanced interface repository features, see the *CORBA Programmer's Guide.*

Controlling the Interface Repository Daemon

Overview

The primary interface repository tasks for administrators are starting and stopping the interface repository daemon.

Starting the interface repository daemon

Run the interface repository daemon on the machine where the interface repository runs. To start the interface repository:

1. Log in as root on UNIX, or as administrator on Windows.
2. Open a terminal or command window.
3. Enter `itifr run`.
4. Follow the directions for your platform:

   Windows

   Leave the command window open.

   UNIX

   Leave the terminal window open, or push the process into the background and close the window.

Stopping the interface repository daemon

`itadmin ifr stop` stops the interface repository daemon.

Managing IDL Definitions

Overview

Orbix includes an API that offers applications complete programmatic control over managing and accessing IDL definitions in the interface repository. Occasionally, you might require manual control to list definitions, remove invalid definitions, and so on. This is especially useful during application development and troubleshooting.

The interface repository has a structure that mirrors the natural containment of the IDL types in the repository. Understanding these types and their relationships is key to understanding how to use the interface repository. Refer to the *CORBA Programmer's Guide* for more information.

In this section

This section provides information on using the interface repository to perform the following tasks manually:

For a complete reference of the commands used to manage the interface repository, see "Repository Management" on page 295.

Browsing Interface Repository Contents

Overview

This section shows how to use `itadmin` commands to perform these tasks:

- List the current container
- Display the containment hierarchy
- Navigate to other levels of containment

The `foo.idl` interface provides a simple example of containment, in which interface `Foo` contains a typed`ef` and two operations:

```
// Begin foo.idl

interface Foo {
    typedef long MyLong;
    MyLong op1();
    void op2();
};
```

List the current container

`itadmin ifr list` lists the specified or current container's contents.

```
itadmin ifr list
Foo/
```

Display the containment hierarchy

`itadmin ifr show` displays the entire containment hierarchy, beginning with the current container. For example:

```
itadmin ifr show Foo
    interface Foo
    {
        ::Foo::MyLong
        op1() ;
        typedef long MyLong;
        void
        op2() ;
    };
```

Navigate to other levels of containment

`itadmin ifr cd` lets you navigate to other levels of containment. For example:

```
itadmin ifr cd Foo
itadmin ifr list
op1 MyLong op2
```

Adding IDL Definitions

Overview

Adding IDL definitions to an interface repository makes application objects available to other applications that have access to the same interface repository.

Procedure

You can add IDL definitions to the interface repository with the `idl -R=-v` command, as follows:

1. Go to the directory where the IDL files are located.
2. Enter the following command:

```
idl -R=-v filename
```

Example

The following example shows how to add a simple IDL interface definition to the interface repository with the `IDL` command. The interface definition is:

```
// Begin foo.idl

interface Foo {
    typedef long MyLong;
    MyLong op1();
    void op2();
};
```

The command to add this IDL definition to the interface repository is:

```
$ idl -R=-v foo.idl
Created Alias MyLong.
Created Operation op1.
Created Operation op2.
Created Interface Foo.
$
```

Removing IDL Definitions

Overview

You might wish to remove IDL definitions from the interface repository when they are invalid, or make them unavailable to other applications. To remove an IDL definition, use `itadmin ifr remove` *`scoped-name`*.

Alternatively, to remove the entire contents of the interface repository, use `itadmin ifr destroy_contents`.

Removing an IDL definition

The following example removes the operation `op2` from the `foo.idl` definition:

```
itadmin ifr list
Foo/
itadmin ifr cd Foo
itadmin ifr list
op1 MyLong op2
itadmin ifr remove op2
itadmin ifr list
op1 MyLong
itadmin ifr quit
```

Removing the entire contents of the IFR

To remove the entire contents of the interface repository, use `ifr destroy_contents`. This destroys the entire contents of the interface repository, leaving the repository itself intact.

If you have loaded a very large number of IDL interfaces into the interface repository, and then want destroy the contents of the IFR, you should first increase the value of the following configuration variable:

`plugins:pss_db:envs:ifr_store:lk_max`

This variable specifies the maximum number of locks available to the Berkeley DB. The default is `1000`.

The following example increases this value to `10000`

```
iona_services {
   ...
   ifr {
      ...
      plugins:pss_db:envs:ifr_store:lk_max = "10000";
    };
  };
```

This prevents the IFR from crashing with the following entry in the IFR log file:

```
ERROR: DB del failed; env is ifr_store, db is
   IRObjectPSHomeImpl:1.0, errno is 12 (Not enough space)
```

CHAPTER 8

Managing the Firewall Proxy Service

The Orbix firewall proxy service provides an added layer of security to your CORBA servers by placing a configurable proxy between the server and its clients.

In this chapter

This chapter discusses the following topics:

Orbix Firewall Proxy Service

Overview

The main goal of the firewall proxy service is to enable the firewall administrator to reduce the number of ports that need to be opened to enable access from clients outside the firewall to services inside the firewall. To accomplish this the firewall proxy service creates and registers a proxy for each POA created by a server using the service. The proxies then intercept requests made by clients and forwards the requests on to the appropriate server.

Server registration

Any server using the firewall proxy service will exchange IOR template information with the firewall proxy service during a registration process that is kicked off by the creation of a POA. When a server creates a new POA, the firewall proxy service creates a separate proxy which will forward client requests.

Request forwarding

When a server has registered with the firewall proxy service, it will generate IORs that point clients to proxies managed by the firewall proxy service. When a client invokes a request on one of these IORs, the request is intercepted by the firewall proxy service. The firewall proxy service then uses the stored template information to forward the request to the appropriate server.

Persistence of registrations

The firewall proxy service maintains a persistent store of registration information. When the firewall proxy service initializes, it recreates the bindings for any server that registered with the service during a previous execution. This assures that server registration is persistent across many executions of the firewall proxy service.

Configuring the Firewall Proxy Service

Overview

The firewall proxy service is designed to act as an application level proxy mechanism for servers configured to utilize the service at run time. Configuration from the server's point of view is trivial and only requires that a plug-in be initialized in the ORB.

Configuring a server to use the firewall proxy service

Any server that wishes to use the firewall proxy service needs to include the firewall proxy plug-in to the list of plug-ins that are loaded for the server's ORB. You add the plug-in to the ORB's plug-in list using `itadmin`. The `itadmin` command is:

```
itadmin variable modify -scope ORBName -type list -value
   iiop_profile,giop,iiop,fps orb_plugins
```

Once the firewall proxy plug-in has been added to the ORB's plug-in list and the firewall proxy service is running, the server will automatically register with the firewall proxy service and the service will relay requests on the client's behalf.

For example, you could configure the typetest demo to use the firewall proxy service. To do this complete the following steps:

1. Create a configuration scope for the typetest demo.

```
itadmin scope create typetest
```

2. Add the ORB's plug-in list to the scope.

```
itadmin variable create -scope ORBName -type list -value
   iiop_profile,giop,iiop,fps orb_plugins
```

3. Run the typetest demo server and specify the ORB name.

```
server -ORBname typetest
```

Java libraries

To use Java services, such as trader, with the firewall proxy service, you need to ensure that the firewall proxy service's registration agent's jar file, `fps_agent.jar`, is added to the services `CLASSPATH`.

Managing the number of proxies

By default, the firewall proxy service imposes no restrictions on the number of servers for which it will proxy requests. The maximum is a factor of system resources. However, you can configure the firewall proxy service to employ a least recently used (LRU) eviction algorithm to select which server bindings to remove. The LRU eviction strategy has configurable soft and hard limits that affect its behavior. The soft limit specifies the point at which the firewall proxy service should proactively begin attempting to reclaim resources. The hard limit specifies the point at which new registrations should be rejected.

The limits are controlled by the following configuration variables:

```
fps:proxy_evictor:soft_limit
fps:proxy_evictor:hard_limit
```

Setting the hard limit to zero effectively disables the services resource control features.

Disabling POA registration

If you develop an application containing a number of "outward" facing objects that you want to place behind the firewall proxy service as well as a number of "inward" facing objects that do not need to be placed behind the firewall proxy service, you can use the `INTERDICTION` POA policy.

The `INTERDICTION` policy controls the behavior of the firewall proxy service plug-in, if it is loaded. The `INTERDICTION` policy has two settings:

`ENABLE`	This is the default behavior of the firewall proxy service plug-in. A POA with its `INTERDICTION` policy set to `ENABLE` will be proxified.
`DISABLE`	This setting tells the firewall proxy service plug-in to not proxify the POA. POAs with their `INTERDICTION` policy set to `DISABLE` will not use the firewall proxy service and requests made on objects under its control will come directly from the requesting clients.

The following code samples demonstrate how to set the `INTERDICTION` policy on a POA. In the examples, the policy is set to `DISABLE` which disables the proxification of the POA. For more information on POA policies read the *CORBA Programmer's Guide*.

Java

```
import com.iona.corba.IT_FPS.*;

// Create a PREVENT interdiction policy.
Any interdiction = m_orb.create_any();
InterdictionPolicyValueHelper.insert(interdiction,
   InterdictionPolicyValue.DISABLE);

Policy[] policies = new Policy[1];
polices[0] = m_orb.create_policy(INTERDICTION_POLICY_ID.value,
   interdiction);

// Create and return new POA.
return m_poa.create_POA("no_fps_poa", null, policies);
```

C++

```
#include <orbix/fps.hh>

// Create a PREVENT interdiction policy.
CORBA::Any interdiction;
interdiction <<= IT_FPS::DISABLE;

CORBA::PolicyList policies(1);
policies.length(1);
policies[0] =
   m_orb->create_policy(IT_FPS::INTERDICTION_POLICY_ID,
   interdiction);

 // Create and return new POA.
return m_poa->create_POA("no_fps_poa", 0, policies);
```

Known Restrictions

The current implementation of the firewall proxy service has the following known restrictions:

- There have are problems using the firewall proxy service and POA collocated calls on UNIX platforms. Calls which should be collocated are being routed through the firewall proxy service in a CORBA mediated call and the call being blocked. The work-around is to remove `POA_Coloc` from the `client_binding_list` configuration parameter.
- Transport Layer Security (TLS) is not supported by the firewall proxy service. This means that the firewall proxy service does not work with Micro Focus's IS2 security infastructure or any other systems that use TLS.
- The J2EE portion of your systems cannot be hidden behind a proxy.

CHAPTER 9

Managing Orbix Service Databases

This chapter explains how to manage databases that store persistent data about Orbix services. It explains the Berkeley DB database management system embedded in Orbix.

A number of Orbix services maintain persistent information (for example, the locator daemon, node daemon, naming service, IFR and CFR). By default, these Orbix services use an embedded Berkeley DB database management system. Typically, Berkeley DB requires little or no administration. The default settings are sufficient for most environments. Tasks that you might want to perform include performing checkpoints, and managing backups, recoveries and log files.

In this chapter

This chapter contains the following sections:

Berkeley DB Environment

Overview

A Berkeley DB environment consists of a set of database files and log files. In Orbix, only a single Berkeley DB environment can be used by one process at a time. Multiple processes using the same Berkeley DB environment concurrently can lead to crashes and data corruption. This means that different Orbix services must use different Berkeley DB environments.

This section explains Berkeley DB environment file types and how they should be stored.

Berkeley DB environment files

A Berkeley DB environment consists of two kinds of files:

Data files contain the real persistent data. By default, these files are stored in the `data` subdirectory of the Berkeley DB environment home directory. For example:

install-dir\var*domain-name*\dbs\locator\data

Transaction log files record changes made to the data files using transactions. By default, these files are stored in the `logs` subdirectory of the Berkeley DB environment home directory. For example:

install-dir\var*domain-name*\dbs\locator\logs

All Orbix services use only transactions to update their persistent data.

Transaction log files can be used to recreate the data files (for example, if these files are corrupted or accidently deleted).

Storing environment files

To maximize performance and facilitate recovery, store all the Berkeley DB environment files on a file system that is local to the machine where the Berkeley DB environment is used.

Log files are of more value than data files because data files can be reconstructed from log files (but not vice-versa). Using different disks and disk controllers for the data and the log files further facilitates recovery.

Performing Checkpoints

Overview

The Berkeley DB transaction logs must be checkpointed periodically to force the transfer of updates to the data files, and also to speed up recovery. By default, each Orbix service checkpoints the transaction logs of its Berkeley DB environment every `15` minutes.

Using configuration variables

You can control checkpoint behavior using the following configuration variables:

```
plugins:pss_db:envs:env_name:checkpoint_period
plugins:pss_db:envs:env_name:checkpoint_min_size
```

For example, the following variable sets the checkpoint period for the locator database to `10` minutes.

```
plugins:pss_db:envs:locator:checkpoint_period = 10;
```

For more information, see the section on the `plugins:pss_db` namespace in the *Configuration Reference Guide*.

Using the command line

You can also checkpoint the transaction logs of a Berkeley DB environment using the `itadmin` command. For example:

```
itadmin pss_db checkpoint env-home/env.ior
```

For more information, see "Persistent State Service" on page 375.

Managing Log File Size

Setting log file size

The Berkeley DB transaction logs are not reused. They grow until they reach a specified level. By default, a transaction log file grows until its size reaches 10 MB. Berkeley DB then creates a new transaction log file.

You can control the maximum size of transaction log files using the following configuration variable:

```
plugins:pss_db:envs:env_name:lg_max
```

`lg_max` is measured in bytes and its value must be to the power of 2.

Deleting and archiving old log files

When a transaction log file does not contain any information pertaining to active transactions, it can be archived or deleted by either of the following:

Using configuration settings By default, each Orbix service checks after each periodic checkpoint to see if any transaction log files are no longer used. By default, old log files are then deleted. You can disable the deletion of old log files by setting the following configuration variable to `false`:

`plugins:pss_db:envs:env_name:checkpoint_deletes_old_logs`

Old log files can also be archived (moved to the `old_logs` directory). To archive old log files, set the following variable to `true`:

`plugins:pss_db:envs:env_name:checkpoint_archives_old_logs`

Using itadmin commands You can also delete or archive the old transaction logs of a Berkeley DB environment using `itadmin` commands:

```
itadmin pss_db archive_old_logs env-home/env.ior
itadmin pss_db delete_old_logs env-home/env.ior
```

For more information, see "Persistent State Service" on page 375.

WARNING: Deleting old transaction log files can make recovery from a catastrophic failure impossible. See "Database Recovery for Orbix Services" on page 154.

Troubleshooting Persistent Exceptions

Overview

This section explains what has happened if you received a `PERSIST_STORE` exception from your Orbix service, and how to recover.

PERSIST_STORE exception

When you see an `IDL:omg.org/CORBA/PERSIST_STORE:1.0` error from an Orbix service, it typically means that the service's persistent storage has become corrupted. The exception is usually accompanied with a minor code representing a Persistent State Service (PSS) exception (for example, `IT_PSS_DB`). Such an error is usually caused by some form of corruption in the underlying database. This corruption can be caused by the following:

- There is limited space on the disk for the underlying database files, and thus it is no longer possible to log transactions. If you find this to be the problem, free disk space immediately and restart the service.
- A service has been shutdown ungracefully (without using the `stop_<domain_name>_services` scripts). For example, this could be caused by executing `kill -9` on the service. This can possibly cause corruption on the database due to unfinished transactions.
- You have put your Orbix services databases on an NFS mounted drive, which is either not available, or your machine's NFS client might have a problem.

When the `IDL:omg.org/CORBA/PERSIST_STORE:1.0` error occurs, contact Support with a copy of logs that show the exact exception, and a description of any unusual activity that may have led up to the problem.

How to recover from a PERSIST_STORE error

To recover from the `PERSIST_STORE` error, it is likely you will need to recover the most recent stable state of your underlying database. If precautions are taken beforehand, your system can be brought back to this stable state with minimal downtime. It is important to determine the level of recovery that is acceptable within your production environment.

For example, you may wish to recover all data prior to the system going down. Alternatively, there may not be as much concern for loss of data, and it may be satisfactory to simply get back to a stable state such that the services can be restarted.

Database Recovery for Orbix Services

Overview

Each time you start an Orbix service that uses Berkeley DB, the service performs a *normal recovery*. If the service was stopped in the middle of an update, the transaction is rolled back, and the service persistent data is restored to a consistent state.

In some cases, however, the data files or the log files are missing or corrupted, and normal recovery is not sufficient. Then you must perform a *catastrophic recovery*. This section explains how to back up your data and log files and perform a full or incremental recovery. It includes the following:

- "Full backup".
- "Performing a full backup".
- "Full backup recovery".
- "Incremental backup".
- "Enabling incremental backup".
- "Performing an incremental backup".
- "Performing an incremental recovery".

Full backup

It is important that you archive a stable snapshot of your services database, which can be used in case a recovery is needed. This is referred to as a *full backup* and can be performed by making a backup of the entire `dbs` directory. The purpose of this backup is that if a `PERSIST_STORE` error occurs for any Orbix services, you can replace the corrupted directory with the backup. The services should then start without a problem.

The backup can be made at any time. The only requirement is that the service be in a stable state (can run and function without errors). You can take the backup directly after configuring your domain, or after the system has been running for a while. The backup that you make will determine the snapshot that your system will return to in the case of a recovery. For example, if you have numerous entries into the IMR (registered POAs, ORBs, and so on), you may wish to add these entries before backing up the locator database. This prevents you from having to do the extra re-configuration if you ever need to recover.

Performing a full backup

To do a full backup, perform the following steps:

Note: If you can bring the services down before doing the backup, you can skip the first step. If you have a live system, and are unable to bring down the services, you can do a backup while the services are running.

1. You must first disable the default periodic deletion and/or archival of old log files during the period while you are backing up the database To disable run the following command:

   ```
   itadmin pss_db pre_backup env.ior
   ```

   The `env.ior` represents a handle to the database. Each service should have its corresponding `env.ior` file within the `dbs/<service name>`.
2. Make a backup of the following directories

   ```
   dbs/<service name>/data directory
   dbs/<service name>/ logs directory
   ```

   Store these backups in a safe location. After a successful full backup, you can discard older full backups (if any).
3. Re-enable the default periodic deletion and/or archival of old log files:

   ```
   itadmin pss_db post_backup env.ior
   ```

Full backup recovery

To do a full backup recovery, perform the following steps:

1. Determine which service is failing on startup.
2. Ensure that your Orbix services are stopped.
3. Make a temporary backup the `dbs/<service_name>` directory for the service you wish to recover.
4. Delete the `dbs/<service_name>` directory for the service you wish to recover.
5. Replace the deleted `dbs/<service_name>` directory in your environment with the latest full backup of this directory.
6. Restart the services.

The environment should now be in the state that it was in at the time the last full backup was performed.

Incremental backup

You should determine whether you also need to do regular *incremental backups*. Generally, these are performed in an environment that requires a large amount of additional configuration beyond initial domain creation, or undergoes constant changes to the configuration. For example, it might make sense to do incremental backups of the locator database in an environment where POA and ORB names are being created or modified constantly, and you need to be able to recover to the most recent state possible. Similarly, if the naming service is constantly undergoing changes of objects references, naming contexts, and so on, and any recovery needs to reflect the most recent state of the underlying database. Another candidate would be for a configuration repository where variables are added or modified regularly.

Enabling incremental backup

If you determine that you need to do regular incremental backups, you should perform the following steps first. These steps apply to the locator, but similarly can be applied to naming service, CFR, and so on.

1. To enable incremental backup, you should tell the service not to automatically delete old log files. By default, old log files are automatically deleted when it is determined the log file is no longer being used. To disable this default behavior, set the following configuration variable:

   ```
   plugins:pss_db:envs:it_locator:checkpoint_deletes_old_logs =
      "false"
   ```

   You can easily apply this to other services by changing `it_locator` to another service (for example, `it_naming`).

2. To enable the automatic archival of old log files, set the following configuration variable:

   ```
   plugins:pss_db:envs:it_locator:checkpoint_archives_old_logs
   ```

   This will specify whether old log files are automatically archived to the `old_logs` directory. To archive old log files, set this variable to `true`. This defaults to `false`.

3. To specify where the old log files get archived to, set a value for the following:

   ```
   plugins:pss_db:envs:it_locator:old_logs_dir =
      "<path/to/old_logs>"
   ```

The path is usually set relative to `db_home` directory. You must ensure you have sufficient space in the above directory, and also, in the location specified by:

```
plugins:pss_db:envs:it_locator:db_home
```

Note: It is critical to the stability of your system that you have sufficient space in these locations to hold the database files and transaction logs for the service.

Performing an incremental backup

The following assumes that you have previously performed a complete backup (see "Full backup" on page 154) at least once in your environment. An incremental backup performs a backup of the log files that have changed or have been created since the last full or incremental backup.

On a predetermined schedule (once a day or week), do a incremental backup of each service as follows:

1. Disable the default periodic deletion and/or archival of old log files during the period while you are doing an incremental backup of the database. To disable, run the following command:

   ```
   itadmin pss_db pre_backup env.ior
   ```

   The `env.ior` represents a handle to the database. Each service should have its corresponding `env.ior` file within the `dbs/<service name>`.
2. Make a backup of files (if any) in `<service_name>/old_logs` directory. When you have made the backup, it is then safe to remove the contents of the `<service_name>/old_logs` directory in your production database.
3. Make a backup of the `<service_name>/logs` directory. This contains the most recent (current) transaction log.

Performing an incremental recovery

The following explains the steps needed to recover if data and/or log files have been corrupted. These steps assume you have taken regular incremental backups as described in "Incremental backup" on page 156. Perform the following steps:

1. Determine which service is failing on startup.
2. Ensure that your Orbix services are stopped.

3. Make a temporary backup the `dbs/<service_name>` directory for the service you wish to recover.
4. Delete the `dbs/<service_name>` directory for the service you wish to recover.
5. Replace the deleted `dbs/<service_name>` directory in your environment with the latest full backup of this directory (see "Full backup recovery" on page 155).
6. In the order of oldest to the newest, copy the files from `<service_name/old_logs` and `<service_name>/logs` from each incremental backup. Put the incremental backup versions of the log files in `<service_name/old_logs` and `<service_name>/logs` into the `dbs/<service_name>/logs` directory of your environment.
7. Set the following configuration variable to true:

   ```
   plugins:pss_db:envs:env_name:recover_fatal
   ```
8. Start the Orbix services.
9. Set the following configuration variable to false:

   ```
   plugins:pss_db:envs:env_name:recover_fatal
   ```

The environment should now be in the state it was in when the last archived log file was written. These steps apply to the locator but similarly can be applied to naming service, CFR, and so on.

Further information

For more information, SleepyCat Software provides full details of Berkeley DB administration at http://www.sleepycat.com/docs/.

Replicated Databases

Overview

The Berkeley DB supports replicated databases using the master-slave model with automatic promotion of slaves. The following Orbix services use this functionality to increase their availability:

- Locator daemon
- Naming service
- Configuration repository

Using configuration variables

You can control replicated databases with the following configuration variables:

```
pss_db:envs:env-name:allow_minority_master
pss_db:envs:env-name:always_download
pss_db:envs:env-name:election_backoff_ratio
pss_db:envs:env-name:election_delay
pss_db:envs:env-name:election_init_timeout
pss_db:envs:env-name:init_rep
pss_db:envs:env-name:master_heartbeat_interval
pss_db:envs:env-name:max_elections
pss_db:envs:env-name:replica_priority
```

For more details, see `plugins:pss_db:envs:env-name` in the Orbix Configuration Reference.

Using the command line

You can examine the state of a replicated database and remove replicas using the `itadmin` commands. For example:

```
itadmin pss_db list_replicas env-home/env.ior
```

For more details on these commands, see "Persistent State Service" on page 375.

CHAPTER 10

Configuring Orbix Compression

This chapter explains how to configure the Orbix ZIOP compression plug-in. This can enable significant performance improvements on low bandwidth networks.

In this chapter

This chapter includes the following topics

Introduction

Overview

The Orbix ZIOP compression plug-in provides optional compression/decompression of GIOP messages on the wire. Compressed and uncompressed transports can be mixed together. This can enable significant performance improvements on low bandwidth networks.

These performance improvements depend on the network and the message data. For example, if the requests contain already compressed data, such as `.jpeg` images, there is virtually no compression. However, with repetitive string data, there is good compression.

ZIOP stands for Zipped Inter-ORB Protocol, which is an proprietary Orbix feature. Figure 15 shows a simple overview of ZIOP compression in a client-server environment.

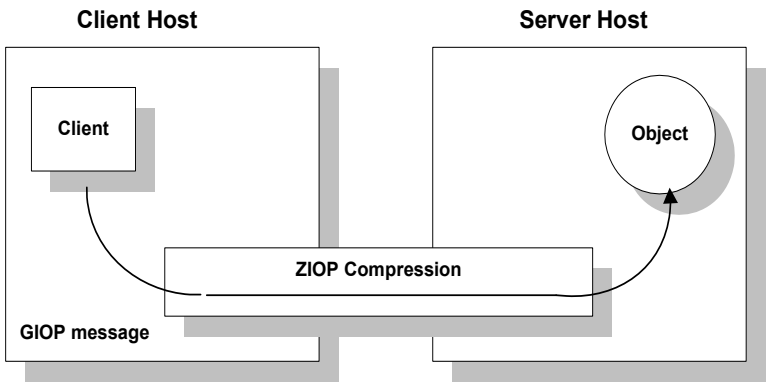

Figure 15: *Overview of ZIOP Compression*

Implementation

Orbix ZIOP compression has been implemented in both C++ and Java and is available on all platforms. The Orbix compression plug-in (`ziop`) supports the following compression algorithms:

- gzip
- pkzip
- bzip2

The compression is performed using a configurable compression library. Compression can be configured on a per-ORB basis, and also on a per-binding basis (using ORB policies).

Per-ORB settings can be made in the client or server scope of your configuration file (described in this chapter). More fine grained per-binding settings can be made programmatically (see the *Orbix CORBA Programmer's Guide* for details).

Additional components

The following Orbix components have also been updated for ZIOP compression:

- The `giop_snoop` plug-in has been updated to detect ZIOP compressed messages.
- The `iordump` tool has been updated to parse the new IOR component for ZIOP compression.

Configuring Compression

Overview

Orbix uses symbolic names to configure plug-ins and then associates them with a Java or a C++ implementation. The compression/decompression plug-in is named `ziop`. This is implemented in Java by the `com.iona.corba.ziop.ZIOPPlugIn` class, and in C++ by the `it_ziop` shared library.

This section shows how to configure the behavior of the compression plug-in for your client or servers. It includes the following:

- "Configuring the ziop plug-in".
- "Configuring binding lists".
- "Enabling compression".
- "Setting the compression algorithm".
- "Setting the compression level".
- "Setting the compression threshold".

Note: These settings must be added to your client or server configuration scope, as appropriate.

Configuring the ziop plug-in

To configure the `ziop` plug-in, perform the following steps:

1. Ensure that the following entries are present in your Orbix configuration file:

```
plugins:ziop:shlib_name = "it_ziop";
plugins:ziop:ClassName = "com.iona.corba.ziop.ZIOPPlugIn";
```

2. Include the `ziop` plug-in the ORB plug-ins list:

```
orb_plugins = [ .... "ziop" ...];
```

For example:

```
orb_plugins = ["local_log_stream", "iiop_profile", "giop",
   "ziop", "iiop"];
```

Configuring binding lists

To enable compression/decompression for CORBA IIOP communication, ensure that your binding lists contain the following entries.

For clients:

```
binding:client_binding_list = ["GIOP+ZIOP+IIOP"];
```

For servers:

```
plugins:giop:message_server_binding_list = ["ZIOP+GIOP"];
```

The client or server binding lists can be much more complicated than these simple examples, although these are adequate for compressed GIOP/IIOP communication. Here is an example of more complex binding lists:

```
binding:client_binding_list = ["OTS+GIOP+ZIOP+IIOP_TLS",
   "CSI+GIOP+ZIOP+IIOP_TLS", "GIOP+ZIOP+IIOP_TLS",
   "CSI+GIOP+ZIOP+ZIOP+IIOP", "GIOP+ZIOP+IIOP"];
plugins:giop:message_server_binding_list = [ "BiDir_GIOP",
   "ZIOP+GIOP", "GIOP"];
```

Enabling compression

To enable or disable compression, use the `policies:ziop:compression_enabled` configuration variable. For example:

```
policies:ziop:compression_enabled = "true";
```

The default value is `true`. This means that even when this entry does not appear in the configuration, compression is enabled. However, the `ziop` plug-in must first be loaded in the `orb_plugins` list, and selected by a server or client binding.

Setting the compression algorithm

The default compression algorithm can be set using the `policies:ziop:compressor_id` configuration variable. For example:

```
policies:ziop:compressor_id = "1";
```

Possible values are as follows:

`1` gzip algorithm

`2` pkzip algorithm

`3` `bzip2` algorithm

If this configuration variable is not specified, the default value is `1` (`gzip` compression).

The ZIOP compression plug-in can be extended with additional compression algorithms using the `IT_ZIOP::CompressionManager` API. See the *Orbix CORBA Programmer's Guide* for details.

Setting the compression level

To set compression levels, use the `policies:ziop:compressor:`*compressor_id*`:level` variable.

Using this variable, you can specify the compression level for each of the algorithms registered in the `ziop` plug-in. The permitted values are specific to the selected algorithm. For example:

```
policies:ziop:compressor:1:level = "9";
```

For the gzip and pkzip algorithms, possible values are in the range between `0` (no compression) and `9` (maximum compression). The default value is `9`.

For the bzip2 algorithm, (`compressor_id` = `3`), possible values are in the range between `1` (least compression) and 9 (maximum compression). The default value is `9`.

Setting the compression threshold

The compression threshold defines the message size above which compression occurs.

To specify the minimum message size that is compressed, use the `policies:ziop:compression_threshold` variable. For example:

```
policies:ziop:compression_threshold = "50";
```

Using this setting, messages smaller than `50` bytes are not compressed.The default setting is `0`, which means that all messages are compressed.

If you set this to a negative value, the compression threshold is equal to infinity, which means that messages are never compressed. This can be of use if you want to enable compression in one direction only. For example, you can compress messages sent from the server to the client, while in the other direction, messages from the client to the server remain uncompressed.

Example Configuration

Overview

This section shows some example compression configurations. It includes the following:

- "Standard ziop configuration".
- "Debug configuration with giop_snoop".

Standard ziop configuration

The following example shows a standard compression configuration in the `ziop_test` configuration scope:

```
ziop_test {
#These settings are necessary for the ziop plug-in
plugins:ziop:ClassName = "com.iona.corba.ziop.ZIOPPlugIn";
plugins:ziop:shlib_name = "it_ziop";
orb_plugins = ["local_log_stream", "iiop_profile", "giop",
   "ziop", "iiop"];
binding:client_binding_list = ["GIOP+ZIOP+IIOP"];
plugins:giop:message_server_binding_list = ["ZIOP+GIOP"];

#These settings are optional
policies:ziop:compression_enabled = "true";
policies:ziop:compressor_id = "1";
policies:ziop:compression_level = "9";
policies:ziop:compression_threshold = "80";
};
```

Depending on the particular circumstances, these settings must be added to the client or the server scope, as appropriate.

If you do not use a scope for your client or server, you can put the settings into the global scope, however, this is not recommended.

Debug configuration with giop_snoop

The following example shows a debug configuration using the `giop_snoop` plug-in:

```
ziop_test {

plugins:ziop:ClassName = "com.iona.corba.ziop.ZIOPPlugIn";
plugins:ziop:shlib_name = "it_ziop";

plugins:giop_snoop:shlib_name = "it_giop_snoop";
plugins:giop_snoop:ClassName =
   "com.iona.corba.giop_snoop.GIOPSnoopPlugIn";

orb_plugins = ["local_log_stream", "iiop_profile", "giop",
   "giop_snoop", "ziop", "iiop"];

binding:client_binding_list = ["GIOP+ZIOP+GIOP_SNOOP+IIOP"];
plugins:giop:message_server_binding_list =
   ["GIOP_SNOOP+ZIOP+GIOP"];

event_log:filters = ["IT_GIOP=*"];
policies:ziop:compression_enabled = "true";
policies:ziop:compressor_id = "1";
policies:ziop:compression_level = "9";
policies:ziop:compression_threshold = "80";
};
```

Using this configuration, you can trace the compression/decompression behavior. The `giop_snoop` plug-in logs the parameters to standard out before or after the `ziop` plug-in (depending on its position before or after the `ZIOP` plug-in).

To send the output to a file instead of standard out, use the following setting:

```
plugins:local_log_stream:filename = "c:\temp\test.log";
```

Message Fragmentation

Overview

The GIOP/IIOP protocol from version 1.1 can fragment messages. The default setting for Orbix is to use message fragmentation. The default fragment size is 16 KB.

This is relevant to the `ziop` plug-in, because the compression algorithm can access at most a single fragment at a time. The compression plug-in therefore operates at the granularity of a single fragment. In this way, message fragmentation can potentially have a large effect on the compression rate.

Increasing message fragment size

Depending on the structure of your data, it might make sense to increase the fragment size so that the compression algorithm is optimized for larger blocks of data. You can configure the fragment size using the `policies:iiop:buffer_sizes_policy:default_buffer_size` configuration variable. For example:

```
policies:iiop:buffer_sizes_policy:default_buffer_size = "65536";
```

This sets the fragment size to 64 KB.

Fragmentation example

Only the overall message size is transmitted. For example, if the message is only 4 KB, only these 4 KB are transmitted. Only if the message is larger than the maximum fragment size will it be transmitted in fragments.

For example, if the maximum fragment size is 16 KB. And the message size is 44 KB. The message will be sent in fragments of 16 KB, 16 KB, and 12 KB.

CHAPTER 11

Configuring Advanced Features

This chapter explains some how to configure advanced features such as Internet Protocol version 6.0 and bidirectional GIOP.

In this chapter

This chapter includes the following topics

Configuring Internet Protocol Version 6

Overview

Orbix provides support for Internet Protocol version 4 (IPv4) and Internet Protocol version 6 (IPv6). Orbix supports IPv4 connections by default. IPv6 fixes a number of issues in IPv4, such as the limited number of available IPv4 addresses, and adds improvements in routing and network configuration.

Supported platforms

Orbix supports IPv6 on the following platforms:

- Windows XP and Vista
- Sun Solaris version 8, 9,and 10
- Red Hat Linux AS version 3 and 4
- IBM z/OS v2.3, and v2.4

Configuring IPv6 in Orbix

You can configure Orbix servers to listen for the following connections:

- IPv4 only
- IPv4 and IPv6
- IPv6 only

The default behavior is for servers to listen for IPv4 connections only. The following configuration variables control this behavior for Orbix servers and clients:

- `policies:network:interfaces:prefer_ipv4`
- `policies:network:interfaces:prefer_ipv6`

For example, to enable Orbix communication over IPv6 only, add the following setting in the ORB or global configuration scope:

```
policies:network:interfaces:prefer_ipv6 = "true";
```

Configuring IPv4 only communication

By default, `prefer_ipv4` is set to `true` and `prefer_ipv6` is set to `false`. To continue using Orbix where there is no requirement for IPv6 communication, you do not need to make any changes to configuration.

Note: This is the default behavior on any host (including dual-stack) where the hostname can be resolved to an IPv4 address. However, if the hostname can only be resolved to an IPv6 address, by default, `prefer_ipv4` set to `false` and `prefer_ipv6` is set to `true`.

Configuring IPv4 and IPv6 communication

On hosts that are configured for IPv4 and IPv6 (dual-stack), you can configure Orbix servers to listen for connections from clients communicating over both IPv4 and IPv6. To run Orbix servers in this mode, use the following setting in the ORB or global configuration scope:

```
policies:network:interfaces:prefer_ipv4 = "false";
```

Servers started with this configuration listen for both IPv4 and IPv6 client connections. No special configuration is required for Orbix clients connecting to an Orbix server started in this mode.

Configuring IPv6 only communication

On hosts that are configured for IPv4 and IPv6 (dual-stack), you can configure Orbix servers and clients to communicate over IPv6 only using the following setting

```
policies:network:interfaces:prefer_ipv6 = "true";
```

Servers started with this variable set at the global or ORB configuration scope listen for connections from clients connecting over IPv6. Clients with this configuration try to connect over IPv6 to the server.

Note: When this is set to `true`, no communication is possible from IPv4 clients trying to connect to the server where the server is running on Windows or the server is configured to write numeric addresses into the IOR.

If the hostname can only be resolved to an IPv6 address, by default, the server only listens for IPv6 communication; there is no need to set any configuration for the server or client.

Deploying an IPv6 domain

The LOCAL_HOSTNAME setting in your top-level configuration file, located in *orbixhlq*.CONFIG(DEFAULT@) by default, specifies a hostname for publishing IORs. If you have already created an Orbix configuration domain that uses a hostname for publishing IORs, you can reuse this configuration, and simply add the prefer_ipv4 and prefer_ipv6 configuration settings as appropriate.

Alternatively, if you wish to publish numeric addresses in your IORs (and are switching from IPv4 addresses to IPv6 addresses), or if you are creating a configuration domain from scratch, you should update the *orbixhlq*.CONFIG(BASETMPL) configuration template with the prefer_ip4 and prefer_ipv6 settings as appropriate. You should also specify the IP address of your local hostname in the LOCAL_HOSTNAME setting in the deploy job.

Note: If you wish to publish an IPv6 numeric address in the IORs for your services, you need to manually update any settings in your configuration domain that use the corbaloc URL scheme (see the examples that follow).

For example, the locator uses:

```
IT_LocatorReplicas =
   ["iona_services.locator=corbaloc:iiop:1.2@%{LOCAL_HOSTNAME}:%
   {LOCAL_LOCATOR_PORT}/IT_LocatorReplica"];
```

The LOCAL_HOSTNAME variable, which refers to a numeric IPv6 address, must be enclosed in square brackets:

```
IT_LocatorReplicas =
   ["iona_services.locator=corbaloc:iiop:1.2@[%{LOCAL_HOSTNAME}]
   :%{LOCAL_LOCATOR_PORT}/IT_LocatorReplica"];
```

Backward compatibility

Enabling a server for IPv6 by setting prefer_ipv4 to false does not affect the ability of older Orbix clients, or clients that have not been enabled for IPv6, to connect to the server.

For more details on compatibility, see "Configuring IPv6 only communication" on page 173.

Direct persistence and replica failover with IPv6

When configuring direct persistence and replica failover, you should enclose IPv6 numeric addresses in brackets. For example:

```
MyConfigApp {
 ...
   wka_1:iiop:addr_list=["[2001:cc1e:1:3:203:baff:fe66:240b]:107
   5", "+host2.com:2075"];
 ...
}
```

For more details, see "Fault Tolerance and Replicated Servers" on page 81.

Specifying IPv6 addresses in a corbaloc

The use of numeric IPv6 addresses is supported since IIOP version 1.2. You must enclose numeric IPv6 addresses in brackets in the corbaloc URL. For example:

```
"corbaloc::1.2@[fe80::203:baff:fe6f:3e91]:125/OneAddress"
```

Configuring Bidirectional GIOP

Overview

This section explains how to set up your system to use bidirectional GIOP. This allows callbacks to be made using a connection opened by the client, instead of requiring the server to open a new connection for the callback.

Bidirectional GIOP is decoupled from IIOP, and is applicable over arbitrary connection-oriented transports (for example, IIOP/TLS or SHMIOP). Bidirectional GIOP may be used regardless of how the callback IOR is passed to the server. For example, it can be passed over an IDL interface, using a shared file, or using a naming or trader service.

GIOP specifications

Orbix supports bidirectional GIOP (General Inter-ORB Protocol), as described in the firewall submission:

http://www.omg.org/docs/orbos/01-08-03.pdf.

As originally specified, GIOP connections were restricted to unidirectional. This proved to be very inconvenient in certain deployment scenarios where the callback pattern was in use, and clients could not accept incoming connections (for example, due to sandbox restrictions on Java applets, or the presence of client-side firewalls). This restriction was relaxed for GIOP 1.2, allowing bidirectional connections to be used under certain conditions.

This section includes the following:

- "Enabling Bidirectional GIOP" on page 177.
- "Migration and Interoperability Issues" on page 180.

Enabling Bidirectional GIOP

Overview

Bidirectional GIOP is enabled by overriding policies in the client and server applications. To enable bidirectional GIOP, perform the following steps:

1. "Set the export policy to allow".
2. "Set the offer policy to allow".
3. "Set the accept policy to allow".

Set the export policy to allow

The POA used to activate the client-side callback object must have an effective `BiDirPolicy::BiDirExportPolicy` set to `BiDirPolicy::ALLOW`. You can do this programmatically by including this policy in the list that is passed to `POA::create_POA()`. Alternatively, you can do this in configuration, using the following setting:

```
policies:giop:bidirectional_export_policy="ALLOW";
```

This results in including an `IOP::TAG_BI_DIR_GIOP` component in the callback IOR. This indicates that bidirectional GIOP is enabled and advertising a `GIOP::BiDirId` generated for that POA.

If necessary, you can control the lifespan of the `BiDirId` by using the proprietary `IT_BiDirPolicy::BiDirIdGenerationPolicy`, either allowing random or requiring repeatable IDs be generated. This is only an issue if the callback POA is persistent, in which case repeatable IDs are required. This would be unusual because callbacks are usually purely transient, in which case the default `BiDirIdGenerationPolicy` is appropriate.

Note: Setting policies programatically gives more fine-grained control than setting policies in configuration. See "Implications for pre-existing application code" on page 180 for more details.

Set the offer policy to allow

A bidirectional offer is triggered for an outgoing connection by setting the effective `BiDirPolicy::BiDirOfferPolicy to ALLOW` for an invocation. This policy may be overridden in the usual way—in descending order of precedence, either on the object reference, current thread, ORB policy manager. Alternatively, you can do this in configuration, using the following setting:

```
policies:giop:bidirectional_offer_policy="ALLOW";
```

The `client_policy` demo illustrates the different ways of overriding client policies. This results in an `IOP::BI_DIR_GIOP_OFFER` service context being passed with the request, unless the policies effective for the callback POA conflict with the outgoing connection (for example, if the former requires security but the latter is insecure).

Set the accept policy to allow

On the server side, the effective `BiDirPolicy::BiDirAcceptPolicy` for the callback invocation must be set to `ALLOW`. You can do this in configuration, using the following setting:

```
policies:giop:bidirectional_accept_policy="ALLOW";
```

This accepts the client's bidirectional offer, and uses an incoming connection for an outgoing request, as long the policies effective for the invocation are compatible with the connection.

Confirming bidirectional GIOP is in use

The simplest way to check that bidirectional GIOP is in use is to examine your log file. First, ensure that the level configured for the `IT_GIOP` sub-system includes `INFO_LOW` events, for example:

```
event_log:filters = [ "IT_GIOP=INFO_LOW+WARN+ERROR+FATAL", ...];
```

For each client binding established, LocateRequest/Request and/or LocateReply/Reply sent or received in the bidirectional sense, the log message includes a `[bidirectional]` suffix.

You can also use the `iordump` utility to check that the `TAG_BI_DIR_GIOP` component is present in the callback IOR. For information on using `iordump`, see Chapter 15 on page 217.

Server and client binding lists

In a generated configuration domain, by default, your client and server binding lists are set to include `BiDir_GIOP`. You do not have to configure these configuration settings manually. The default settings are explained as follows:

- On the server-side, the `binding:client_binding_list` includes an entry for `BiDir_GIOP`, for example:

```
binding:client_binding_list = [ "OTS+BiDir_GIOP",
   "BiDir_GIOP", "OTS+GIOP+IIOP", "GIOP+IIOP", ... ];
```

  This enables the existing incoming message interceptor chain to be re-used, so that the outgoing client binding dispatches the callback invocation.

- On the client-side, the `plugins:giop:message_server_binding_list` includes an entry for `BiDir_GIOP`, for example:

```
plugins:giop:message_server_binding_list=
["BiDir_GIOP","GIOP" ];
```

  This enables the existing outgoing message interceptor chain to be re-used for an incoming server binding.

Migration and Interoperability Issues

Overview

This section includes the following bidirectional GIOP issues:

- "Implications for pre-existing application code".
- "Incompatible ORBs".
- "Interoperability with Orbix 3".
- "Orbix 6.x restrictions".

Implications for pre-existing application code

There are no implications for existing applications that do not need bidirectional GIOP. This feature is disabled by default.

Otherwise, the code impact can be minimized by setting the relevant policies using configuration, as explained "Enabling Bidirectional GIOP" on page 177. However, this is quite a coarse grained approach, and often its not necessary or desirable to enable bidirectional GIOP for the entire ORB. The recommended approach is to selectively override the relevant programmatic policies in a fine-grained manner on exactly those elements (POAs, ORBs, threads, object references) that require it.

Also, currently existing persistent callback IORs (for example, those bound in the naming service) must be regenerated to include the `TAG_BI_DIR_GIOP` component. However, this is unlikely to impact many real applications as callback references are usually transient and regenerated every time the client application is run.

Incompatible ORBs

There are several incompatible bidirectional schemes in use. For example, Orbacus uses a proprietary mechanism, and several commercial and open source ORBs support the soon-to-be obsolete bidirectional standard; while Orbix 2000 and Orbix E2A 5.x/6.0 do not have any analogous functionality.

All of these schemes are mutually incompatible and non-interoperable. Hence, Orbix 6.x reverts to unidirectional GIOP when interoperating with any of these ORBs.

Interoperability with Orbix 3

Orbix 6.x includes support for interoperability with Orbix 3.x (Generation 3). This enables an Orbix 6.x server to invoke on an Orbix 3.x callback reference in a bidirectional fashion. To configure interoperability with Orbix 3.x, perform the following steps:

1. Set the `IT_BiDirPolicy::BidirectionalGen3AcceptPolicy` to `ALLOW`. This is a proprietary policy analogous to `BiDirPolicy::BidirectionalAcceptPolicy`. It enables an Orbix 6.x server to accept an Orbix 3.x bidirectional offer.

   You can do this either programmatically or using the following configuration setting:

```
policies:giop:bidirectional_gen3_accept_policy="ALLOW";
```

2. Include the appropriate `BiDir_Gen3` entry in the server's configured `binding:client_binding_list`. For example,

```
binding:client_binding_list =
["OTS+BiDir_GIOP", "BiDir_GIOP", "BiDir_Gen3",
   "OTS+GIOP+IIOP", "GIOP+IIOP", ...];
```

   For more details, see "Server and client binding lists" on page 179.

Orbix 3 restrictions The following restrictions apply to bidirectional GIOP in Orbix 3:

- Orbix 3 bidirectional callback references may only be passed to the server as a request parameter. Orbix 6.x bidirectional callback references can be passed in any way (for example, using the naming service, or a shared file).
- Orbix 3 bidirectional callback references may only be invoked on in a bidirectional fashion during the lifetime of the connection over which it was received. Orbix 6.x bidirectional invocations may be made after the connection is reaped by Active Connective Management and re-established.

The Orbix 6.x and Orbix 3 bidirectional mechanisms will co-exist peacefully. An incoming connection can only be considered for bidirectional invocations by, at most, one of the two schemes, depending on whether the client is based on Orbix 6.x or Orbix 3.x.

Orbix 6.x restrictions

Orbix 6.x includes the following restrictions:

- Orbix 6.x support for Orbix 3 bidirectional GIOP is asymmetric. An Orbix 6.x server can invoke on a Orbix 3 callback reference using bidirectional GIOP. However, an Orbix 6.x client can not produce a callback reference that an Orbix 3 server could invoke on using bidirectional GIOP.
- To be compatible with GIOP 1.2 (that is, not be dependent on GIOP 1.4 `NegotiateSession` messages), only weak `BiDirIds` are used, and the challenge mechanism to detect client spoofing is not supported. This functionality will be added in a future release, when GIOP 1.4 is standardized.

CHAPTER 12

Orbix Mainframe Adapter

The Orbix Mainframe Adapter (MFA) plugin enables you to communicate with Orbix Mainframe CICS and IMS server adapters from Windows and UNIX. It includes a Mapping Gateway interface and an itmfaloc URL resolver. This chapter introduces the CICS and IMS server adapters, and explains how to use the Mapping Gateway interface and the itmfaloc URL resolver.

In this chapter

This chapter contains the following sections:

Note: In addition to Orbix, you must have Orbix Mainframe installed and running before you can use the MFA.

CICS and IMS Server Adapters

Overview

The Orbix Mainframe product includes a CICS server adapter and an IMS server adapter. This section gives a brief description of each of these adapters and includes the following to topics:

- CICS server adapter
- IMS server adapter
- More information

CICS server adapter

The Orbix CICS server adapter is an Orbix Mainframe service that can be deployed in either a native z/OS or UNIX System Services environment. The CICS server adapter acts as a bridge between CORBA/EJB clients and CICS servers. It enables you to set up a distributed system that combines the powerful online transaction processing capabilities of CICS with the consistent and well-defined structure of a CORBA environment.

IMS server adapter

The Orbix IMS server adapter is an Orbix Mainframe service that can be deployed in a native z/OS or UNIX System Services environment. It provides a simple way to integrate distributed CORBA and EJB clients on various platforms with existing and new IMS transactions running on z/OS. The IMS server adapter allows you to develop and deploy Orbix COBOL and PL/I servers in IMS, and to integrate these IMS servers with distributed CORBA clients running on various platforms. It also facilitates the integration of existing IMS transactions, not developed using Orbix, with distributed CORBA clients, without the need to change these existing transactions.

More information

For more information, see the *Orbix Mainframe CICS Adapters Administrator's Guide* and *IMS Adapters Administrator's Guide*, which are available on the documentation web pages at:

https://www.microfocus.com/documentation/orbix/orbixmf631/

Using the Mapping Gateway Interface

Overview

The Mapping Gateway interface is used to control CICS or IMS server adapters running on the mainframe. You can use the Mapping Gateway interface to list the transaction mappings that the server adapter supports, to add or delete individual interfaces and operations, or to change the transaction that an operation is mapped to. A new mapping file can be read, or the existing mappings can be written to a new file. Access to the Mapping Gateway interface using `itadmin` is provided as a plug-in. This plug-in is selected with the `mfa` keyword.

In this section

This section provides some examples of how you can to use the `itadmin mfa` plugin to control CICS and IMS server adapters running on the mainframe. The following topics are covered:

Configuring the Mapping Gateway interface

The Mapping Gateway interface is configured by default. The following configuration values are added to the configuration file:

```
plugins:mfa_adm:grammar_db = "admin_plugins = [..., "mfa_adm"];
plugins:mfa_adm:shlib_name = "it_mfa_adm";
plugins:mfa_adm:grammar_db = "mfa_adm_grammar.txt";
plugins:mfa_adm:help_db = "mfa_adm_help.txt";
```

You must, however, add the mainframe IOR to the configuration file as follows:

```
initial_references:IT_MFA:reference = "IOR: .....";
```

For details of how to obtain the IOR, see the *CICS Adapters Administrator's Guide* and the *IMS Adapters Administrator's Guide*.

Listing itadmin mfa commands

To obtain a list of all the commands provided by the `itadmin mfa` plug-in, use the following command:

```
$ itadmin mfa -help
```

The output is follows:

```
mfa list
    add     -interface <name> -operation <name> <mapped value>
    change  -interface <name> -operation <name> <mapped value>
    delete  -interface <name> -operation <name>
    resolve <interface name>
    refresh [-operation <name>] <interface name>
    reload
    save    [<mapping_file name>]
    switch  <mapping_file name>
    stats
    resetcon
    stop
```

Items shown in angle brackets (`<...>`) must be supplied and items shown in square brackets (`[...]`) are optional. Modules names form part of the interface name and are separated from the interface name with a `/` character. For detailed information on these commands, see Chapter 24.

Printing a list of supported mappings

To print a list of the mappings (interface, operation and name) that the server adapter supports, use the following command:

```
itadmin mfa list
```

For example, the output is as follows:

```
Simple/SimpleObject,call_me, SIMPLESV
nested_seqs,test_bounded,NSTSEQSV
nested_seqs,test_unbounded,NSTSEQSV
```

Changing an operation's transaction mapping

You can use the `mfa change` command to change the transaction to which an existing operation is mapped. For example, to change the transaction to which the `call_me` operation is mapped, from `SIMPLESV` to `NSTSEQSV`, use the following command:

```
itadmin mfa change -interface Simple/SimpleObject -operation
   call_me NSTSEQSV
```

To view the result, use the `mfa list` command:

```
itadmin mfa list
```

For example, the output is as follows:

```
Simple/SimpleObject,call_me, NSTSEQSV
nested_seqs,test_bounded,NSTSEQSV
nested_seqs,test_unbounded,NSTSEQSV
```

Saving mappings to a specified file and reloading current mappings

You can use the `mfa save` command to get the server adapter to save its current mappings to either its current mapping file or to a filename that you provide. For example, to cause the server adapter to save its current mappings to a file called `myMappings.map`, but reload the list of mappings from its mapping file, use the following commands:

```
itadmin mfa save "c:\myMappings.map"
itadmin mfa reload
```

To view the result, use the `mfa list` command:

```
itadmin mfa list
```

For example, the output is as follows:

```
Simple/SimpleObject,call_me, SIMPLESV
nested_seqs,test_bounded,NSTSEQSV
nested_seqs,test_unbounded,NSTSEQSV
```

Switching the mapping file

You can get the server adapter to switch to using a new mapping file and export only the mappings contained within it. For example, to get the server adapter to switch from its current mapping file to `myMappings.map`, use the following command:

```
itadmin mfa switch "c:\myMappings.map"
```

To view the result, use the `mfa list` command:

```
itadmin mfa list
```

The output looks as follows:

```
Simple/SimpleObject,call_me, NSTSEQSV
nested_seqs,test_bounded,NSTSEQSV
nested_seqs,test_unbounded,NSTSEQSV
```

Invoking on exported interfaces

The Mapping Gateway interface provides the means by which IIOP clients can invoke on the exported interfaces. Using the `resolve` operation, an IOR can be retrieved for any exported interface. This IOR can then be used directly by IIOP clients, or registered with an Orbix naming service as a way of publishing the availability of the interface. For example, to retrieve an IOR for `Simple` IDL, use the following command:

```
itadmin mfa resolve Simple/SimpleObject
```

Selecting a specific server adapter

To select a specific server adapter, provide the `ORBname` for the server adapter on a request. For example, to specify the CICS server adapter and obtain the IOR for the `Simple` interface, use the following command:

```
itadmin -ORBname iona_utilities.cicsa mfa resolve
   Simple/SimpleObject
```

Locating Server Adapter Objects Using itmfaloc

Overview

The CICS and IMS server adapter maintains object references that identify CORBA server programs running in CICS and IMS respectively. A client must obtain an appropriate object reference in order to access the target server. The `itmfaloc` URL resolver plug-in facilitates and simplifies this task.

Note: The itmfaloc URL resolver is only available in C++.

In this section

This section discusses how you can use the `itmfaloc` URL resolver as an alternative to the `itadmin mfa resolve` command. The following topics are covered:

Locating server adapters using IORs

One way of obtaining an object reference for a target server, managed by the CICS or IMS server adapter, is to retrieve the IOR using the `itadmin` tool. This calls the `resolve()` method on the server adapter's Mapping Gateway interface and returns a stringified IOR. For example, to retrieve an IOR for `Simple` IDL, use the following command:

```
itadmin mfa resolve Simple/SimpleObject
```

When retrieved, the IOR can be distributed to the client and used to invoke on the target server running inside CICS.

Locating objects using itmfaloc

In some cases, the use of `itadmin` and the need to persist stringified IORs is not very manageable, and a more dynamic approach is desirable. The `itmfaloc` URL resolver is designed to provide an alternative approach. It follows a similar scheme to that of the corbaloc URL technique.

In this way, the Orbix CORBA client can specify a very simple URL format which identifies the target service required. This text string can be used programmatically in place of the rather cumbersome stringified IOR representation.

Format of an itmfaloc URL

An `itmfaloc` URL is a string of the following format:

```
itmfaloc:<InterfaceName>
```

<InterfaceName> is the fully-scoped name of the IDL interface implemented by the target server (as specified in the server adapter mapping file).

What happens when itmfaloc is used

When an `itmfaloc` URL is used in place of an IOR, the Orbix client application contacts the server adapter to attain an object reference for the desired CICS or IMS server. The `itmfaloc` URL string only encodes the interface name and not the server adapter's location. To establish the initial connection to the server adapter, the value of the `IT_MFA:initial_references` variable is used.

If multiple server adapters are deployed, the client application must specify the correct `IT_MFA:initial_references` setting in order to contact the correct server adapter. You can do this by specifying the appropriate ORB name, which represents the particular configuration scope. For example, for the CICS server adapter, `-ORBname iona_utilities.cicsa`

If the client application successfully connects to the server adapter, it calls the `resolve()` operation on the Mapping Gateway object reference, retrieving an object reference for the target server managed by the server adapter.

Example of using itmfaloc

The simple demo client code that is shipped with Orbix uses a file-based mechanism to access the target server's stringified IOR. If the target server resides in CICS or IMS, an alternative approach is to specify an `itmfaloc` URL string in the `string-to-object` call; for example:

```
objref = orb->string_to_object("itmfaloc:Simple/SimpleObject");
if (CORBA::is_nil(objref))
        {
            return 1;
        }
simple = Simple::SimpleObject::_narrow(objref);
```

Part 3

Monitoring Orbix Applications

In this part

This part contains the following chapters:

CHAPTER 13

Setting Orbix Logging

Orbix logging lets you collect system-related information, such as significant events, and warnings about unusual or fatal errors.

Through a configuration domain's logging variables, you can specify the kinds of messages to collect, and where to direct them.

In this chapter

This chapter covers the following topics:

Setting Logging Filters

Overview

The `event_log:filters` configuration variable sets the level of logging for specified subsystems, such as POAs or the naming service. This variable is set to a list of filters, where each filter sets logging for a specified subsystem with the following format:

```
subsystem=severity-level[+severity-level]...
```

For example, the following filter specifies that only errors and fatal errors for the naming service should be reported:

```
IT_NAMING=ERR+FATAL
```

The `subsystem` field indicates the name of the Orbix subsystem that reports the messages (see Table 6 on page 198). The `severity` field indicates the severity levels that are logged by that subsystem (see Table 7 on page 201).

You can set this variable by directly editing a configuration file, or using `itadmin` commands. In the examples that follow, logging is enabled as follows:

- For POAs, enable logging of warnings, errors, fatal errors, and high-priority informational messages.
- For the ORB core, enable logging of all events.
- For all other subsystems, enable logging of warnings, errors, and fatal errors.

Set in a configuration file

In a configuration file, `event_log:filters` is set as follows:

```
event_log:filters=["log-filter"[,"log-filter"]...]
```

The following entry in a configuration file explicitly sets message severity levels for the POA and ORB core, and all other subsystems:

```
event_log:filters = ["IT_POA=INFO_HI+WARN+ERROR+FATAL",
                     "IT_CORE=*", "*=WARN+ERR+FATAL"];
```

Set with itadmin

You can use `itadmin` commands `variable create` and `variable modify` to set and modify `event_log:filters`. For example, the following command creates the same setting as shown before, this time specifying to set this logging for the locator daemon:

```
itadmin variable modify -scope locator -type list -value\
   IT_POA=INFO_HI+WARN+ERROR+FATAL, \
   IT_CORE=*, \
   *=WARN+ERR+FATAL \
   event_log:filters
```

Dynamic logging

The `itadmin logging get` and `logging set` commands enable the Orbix event log filters setting to be displayed or updated dynamically in a deployed Orbix Mainframe application. You can also perform these actions using the Administrator Web Console. For more details, see "Dynamic Logging" on page 205.

Logging Subsystems

You can apply one or more logging severity levels to any or all ORB subsystems. Table 6 shows the available ORB subsystems. By default, Orbix logs warnings, errors, and fatal errors for all subsystems.

Table 6: *Orbix Logging Subsystems*

Subsystem	Description
*	All logging subsystems.
IT_ACTIVATOR	Activator daemon.
IT_ATLI2_IOP	Abstract Transport Layer Inter-ORB protocol.
IT_ATLI2_IP	Abstract Transport Layer Internet Protocol plug-in.
IT_ATLI2_ITMP	Abstract Transport Layer Multicast plug-in.
IT_ATLI2_ITRP	Abstract Transport Layer Reliable Multicast plug-in.
IT_ATLI2_SHM	Abstract Transport Layer Shared Memory plug-in.
IT_ATLI2_SOAP	Mainframe SOAP plug-in.
IT_ATLI_TLS	Abstract Transport Layer (secure).
IT_ClassLoading	Classloading plug-in (Java).
IT_CODESET	Internationalization plug-in.
IT_CONFIG_REP	Configuration repository.
IT_CORE	Core ORB.
IT_CSI	Common Secure Interoperability.
IT_GIOP	General Inter-Orb Protocol (transport layer).
IT_GSP	Generic Security plug-in.
IT_HTTP	HTTP plug-in
IT_HTTPS	HTTPS plug-in

Table 6: *Orbix Logging Subsystems*

Subsystem	Description
IT_IFR	Interface repository.
IT_IIOP	Internet Inter-Orb Protocol (transport layer).
IT_IIOP_PROFILE	Internet Inter-Orb Protocol profile (transport layer).
IT_IIOP_TLS	Internet Inter-Orb Protocol (secure transport layer).
IT_JAVA_SERVER	Java server plug-in
IT_LEASE	Session management service.
IT_LOCATOR	Server locator daemon.
IT_MGMT	Management instrumentation plug-in.
IT_MGMT_SVC	Management service.
IT_MFA	Mainframe adapter service.
IT_MFU	Mainframe client adapter service.
IT_NAMING	Naming service.
IT_NOTIFICATION	Event service.
IT_NodeDaemon	Node daemon.
IT_OTS_LITE	Object transaction service.
IT_POA	Portable object adapter.
IT_POA_LOCATOR	Server locator daemon (POA specific).
IT_PSS	Persistent state service.
IT_PSS_DB	Persistent state service (raw database layer).
IT_PSS_R	Persistent state service (database driver).
IT_SAF	Mainframe SAF plug-in.
IT_SCHANNEL	Microsoft Schannel (Windows only).
IT_TLS	Transport Layer Security.

Table 6: *Orbix Logging Subsystems*

Subsystem	Description
`IT_TS`	Threading/synchronization package.
`IT_XA`	X/Open XA standard (transactions).

Logging Severity Levels

Overview

Orbix supports four levels of message severity:

- Informational
- Warning
- Error
- Fatal error

Informational

Informational messages report significant non-error events. These include server startup or shutdown, object creation or deletion, and information about administrative actions.

Informational messages provide a history of events that can be valuable in diagnosing problems. Informational messages can be set to low, medium, or high verbosity.

Warning

Warning messages are generated when Orbix encounters an anomalous condition, but can ignore it and continue functioning. For example, encountering an invalid parameter, and ignoring it in favor of a default value.

Error

Error messages are generated when Orbix encounters an error. Orbix might be able to recover from the error, but might be forced to abandon the current task. For example, an error message might be generated if there is insufficient memory to carry out a request.

Fatal error

Fatal error messages are generated when Orbix encounters an error from which it cannot recover. For example, a fatal error message is generated if the ORB cannot connect to the configuration domain.

Table 7 shows the syntax used to specify Orbix logging severity levels.

Table 7: *Orbix Logging Severity Levels*

Severity Level	Description
`INFO_LO[W]`	Low verbosity informational messages.

Table 7: *Orbix Logging Severity Levels*

Severity Level	Description
`INFO_MED[IUM]`	Medium verbosity informational messages.
`INFO_HI[GH]`	High verbosity informational messages.
`INFO_ALL`	All informational messages.
`WARN[ING]`	Warning messages.
`ERR[OR]`	Error messages.
`FATAL[_ERROR]`	Fatal error messages.
`*`	All messages.

Redirecting Log Output

Overview

By default, Orbix is configured to log messages to standard error. You can change this behavior for an ORB by setting a logstream plug-in to be loaded by the ORB. For example, you can set the output stream to a local file owned by the ORB, or to the host's system error log.

As with all other configuration variables, these can be set using the `itadmin` commands `variable create` and `variable modify`.

This section includes the following:

- "Setting the output stream to a local file".
- "Using rolling log files".
- "Setting the output stream to the system log".
- "Buffering the output stream before writing to a file".

Setting the output stream to a local file

To set the output stream to a local file, set the following configuration variable:

```
plugins:local_log_stream:filename = filename
```

The following example uses the `itadmin variable modify` command:

```
itadmin variable modify -type string -value
   "/var/adm/mylocal.log" plugins:local_log_stream:filename
```

If your configuration domain is file-based, you can also set this variable in your configuration file. For example:

```
plugins:local_log_stream:filename = "/var/adm/mylocal.log";
```

Using rolling log files

Normally, the local log stream uses a rolling file to prevent the log from growing indefinitely. In this model, the stream appends the current date to the configured filename. This produces a complete filename (for example, `/var/adm/art.log.02172002`). A new file begins with the first event of the day and ends at 23:59:59 each day.

You can disable rolling file behavior by setting the `rolling_file` variable to false. For example:

```
plugins:local_log_stream:rolling_file = "false";
```

Setting the output stream to the system log

The system log stream reports events to the host's system log—`syslog` on UNIX, and the event log on Windows. Each log entry is tagged with the current time and logging process ID, and the event priority is translated into a format appropriate for the native platform.

To set the output stream to the system log, add the `system_log_stream` value to the `orb_plugins` configuration variable. You can use the `system_log_stream` output stream concurrently with the `local_log_stream`, if necessary.

The following `orb_plugins` variable includes the `system_log_stream` value:

```
orb_plugins=["system_log_stream", "iiop_profile", "giop",
   "iiop",];
```

Buffering the output stream before writing to a file

You can also set the output stream to a buffer before writing to a local log file. Use the `plugins:local_log_stream:buffer_file` configuration variable to specify this behavior. When this variable is set to true, by default, the buffer is output to the local file every `1000` milliseconds when there are more than `100` messages logged. The log interval and the number of log elements can also be configured.

For example, the following configuration writes the log output to the `/var/adm/art.log` file every `400` milliseconds if there are more then `20` log messages in the buffer.

```
plugins:local_log_stream:filename = "/var/adm/art.log";
plugins:local_log_stream:buffer_file = "true";
plugins:local_log_stream:milliseconds_to_log = "400";
plugins:local_log_stream:log_elements = "20";
```

Dynamic Logging

Overview

Dynamic logging enables you to modify the `event_log:filters` setting of a deployed Orbix Mainframe server application on the fly, without needing to stop, update the static setting in the configuration file, and restart the server. You can change logging dynamically using `itadmin` commands or the Administrator Web Console (running off-host).

Using itadmin commands

The `itadmin logging get` and `logging set` commands enable the Orbix event log filters to be displayed or updated dynamically on the command line.

For example, the following command dynamically updates the event log filters that are used by the currently running IMS server adapter:

```
itadmin logging set -orbname iona_services.imsa -value
   IT_GIOP=*,IT_MFA=*
```

For full details, see "Event Log" on page 251.

Note: The `ORXADMIN` JCL job may be used in the z/OS batch environment.

C++ management agent registration plug-in

The C++ management agent registration plug-in is designed for use in a domain were the Java-based Orbix management service is not deployed.

This plug-in is not intended as a replacement for the management service but as a means to dynamically change logging levels of servers using `itadmin`.

You do not need to use this plug-in in a domain were the management service is deployed. In such a domain, you can dynamically update log levels of servers using `itadmin` and the Administrator Web Console.

Note: To use the C++ management agent registration plug-in, the naming service must be available. The naming service may be deployed on z/OS, or on a remote host.

To enable this plug-in, review the following configuration updates:

```
# Initial reference must be set
initial_references:IT_MgmtService:plugin = "it_mgmt_agent_reg";

# The plugins library name and version (these are already set
# in ORXINTRL config member)
plugins:it_mgmt_agent_reg:shlib_name = " ORXMGAR";
plugins:it_mgmt_agent_reg:shlib_version = "5";

# The ORB in question must be managed for dynamic logging
plugins:orb:is_managed = "true";

# The naming service must be deployed and the registration of
# the agent with the naming service set to true in the same
# ORB scope.
plugins:it_mgmt:register_agent_with_ns = "true";

# The it_mgmt_agent_reg plug-in must be added to the ORB
# plugins list.
orb_plugins = ["it_mgmt_agent_reg", "local_log_stream",
"iiop_profile", "giop", "ots", "iiop"];
```

Administrator Web Console

For details on using the Administrator Web Console, see the *Orbix Mainframe Management User's Guide*.

CHAPTER 14

Monitoring GIOP Message Content

Orbix includes the GIOP Snoop tool for intercepting and displaying GIOP message content.

In this chapter

This chapter contains the following sections:

Introduction to GIOP Snoop

Overview

GIOP Snoop is a GIOP protocol level plug-in for intercepting and displaying GIOP message content. This plug-in implements message level interceptors that can participate in client and/or server side bindings over any GIOP-based transport. The primary purposes of GIOP Snoop are to provide a protocol level monitor and debug aid.

GIOP plug-ins

The primary protocol for inter-ORB communications is the General Inter-ORB Protocol (GIOP) as defined the CORBA Specification. Orbix provides several GIOP based plug-ins that map GIOP to a number of transports. For example, CORBA IIOP (for TCP/IP), and proprietary Orbix transport mappings, such as SIOP (a shared memory transport), and MPI (a multicast transport for GIOP). GIOP Snoop may be used with these (and any future) GIOP-based plug-ins.

Configuring GIOP Snoop

Overview

GIOP Snoop can be configured for debugging in client, server, or both depending on configuration. This section includes the following configuration topics:

- "Loading the GIOP Snoop plug-in".
- "Client-side snooping".
- "Server-side snooping".
- "GIOP Snoop verbosity levels".
- "Directing output to a file".
- "Using the Java version of GIOP Snoop"

Loading the GIOP Snoop plug-in

For either client or server configuration, the GIOP Snoop plug-in must be included in the Orbix `orb_plugins` list (`...` denotes existing configured settings):

```
orb_plugins = [..., "giop_snoop", ...];
```

In addition, the `giop_snoop` plug-in must be located and loaded using the following settings:

```
// C++
plugins:giop_snoop:shlib_name = "it_giop_snoop";
```

```
// Java
plugins:giop_snoop:ClassName =
   "com.iona.corba.giop_snoop.GIOPSnoopPlugIn";
```

Client-side snooping

To enable client-side snooping, include the `GIOP_SNOOP` factory in the client binding list. In this example, GIOP Snoop is enabled for IIOP-specific bindings:

```
binding:client_binding_list =
      [..., "GIOP+GIOP_SNOOP+IIOP", ...];
```

Server-side snooping

To enable server-side snooping, include the `GIOP_SNOOP` factory in the server binding list.

```
plugins:giop:message_server_binding_list =
     [..., "GIOP_SNOOP+GIOP", ...];
```

Note: For Orbix 6.x, the ordering of this setting has been reversed to correct consistency issues in previous releases of Orbix across Java and C++ configuration.

GIOP Snoop verbosity levels

You can use the following variable to control the GIOP Snoop verbosity level:

```
plugins:giop_snoop:verbosity = "1";
```

The verbosity levels are as follows:

```
1          LOW
2          MEDIUM
3          HIGH
4          VERY HIGH
```

These verbosity levels are explained with examples in "GIOP Snoop Output" on page 212.

Directing output to a file

By default, output is directed to standard error (`stderr`). However, you can specify an output file using the following configuration variable:

```
plugins:giop_snoop:filename = "<some-file-path>";
```

A month/day/year time stamp is included in the output filename with the following general format:

```
<filename>.MMDDYYYY
```

As a result, for a long running application, each day results in the creation of a new log file. To enable administrators to control the size and content of output files GIOP Snoop does not hold output files open. Instead, it opens and then closes the file for each snoop message trace. This setting is enabled with:

```
plugins:giop_snoop:rolling_file = "true";
```

Using the Java version of GIOP Snoop

To use the Java version of the GIOP Snoop plug-in, add the `giop_snoop.jar` file to your classpath. For example:

UNIX

```
export CLASSPATH=
        $CLASSPATH:$IT_PRODUCT_DIR/asp/6.0/lib/asp-corba.jar
```

Windows

```
set CLASSPATH=
        %CLASSPATH%;%IT_PRODUCT_DIR%\asp\6.0\lib\asp-corba.jar
```

GIOP Snoop Output

Overview

The output shown in this section uses a simple example that shows client-side output for a single binding and operation invocation. The client establishes a client-side binding that involves a message interceptor chain consisting of IIOP, GIOP Snoop, and GIOP. The client then connects to the server and first sends a `[LocateRequest]` to the server to test if the target object is reachable. When confirmed, a two-way invocation `[Request]` is sent, and the server processes the request. When complete, the server sends a `[Reply]` message back to the client.

Output detail varies depending on the configured verbosity level. With level `1` (`LOW`), only basic message type, direction, operation name and some GIOP header information (version, and so on) is given. More detailed output is possible, as described under the following examples.

LOW verbosity client-side snooping

An example of `LOW` verbosity output is as follows:

```
[Conn:1] Out:(first for binding)  [LocateRequest] MsgLen: 39 ReqId: 0
[Conn:1] In: (first for binding)  [LocateReply] MsgLen: 8 ReqId: 0
    Locate status: OBJECT_HERE
[Conn:1] Out:  [Request] MsgLen: 60 ReqId: 1  (two-way)
    Operation (len 8) 'null_op'
[Conn:1] In:   [Reply] MsgLen: 12 ReqId: 1
    Reply status (0) NO_EXCEPTION
```

This example shows an initial conversation from the client-side perspective. The client transmits a `[LocateRequest]` message to which it receives a `[LocateReply]` indicates that the server supports the target object. It then makes an invocation on the operation `null_op`.

The `Conn` indicates the logical connection. Because GIOP may be mapped to multiple transports, there is no transport specific information visible to interceptors above the transport (such as file descriptors) so each connection is given a logical identifier. The first incoming and outgoing GIOP message to pass through each connection are indicated by `(first for binding)`.

The direction of the message is given (`Out` for outgoing, `In` for incoming), followed by the GIOP and message header contents. Specific information includes the GIOP version (version 1.2 above), message length and a unique request identifier (`ReqId`), which associates `[LocateRequest]` messages with their corresponding `[LocateReply]` messages. The `(two-way)` indicates the operation is two way and a response (`Reply`) is expected. String lengths such as `len 8` specified for `Operation` includes the trailing null.

MEDIUM verbosity client-side snooping

An example of `MEDIUM` verbosity output is as follows:

```
16:24:39 [Conn:1] Out:(first for binding)  [LocateRequest]   GIOP v1.2  MsgLen: 39
   Endian: big  ReqId: 0
   Target Address (0: KeyAddr)
   ObjKey (len 27) ':>.11........\..A..........'

16:24:39 [Conn:1] In: (first for binding)  [LocateReply]     GIOP v1.2  MsgLen: 8
   Endian: big  ReqId: 0
   Locate status: OBJECT_HERE

16:24:39 [Conn:1] Out:  [Request]          GIOP v1.2  MsgLen: 60
   Endian: big  ReqId: 1  (two-way)
   Target Address (0: KeyAddr)
   ObjKey (len 27) ':>.11........\..A..........'
   Operation (len 8) 'null_op'

16:24:39 [Conn:1] In:   [Reply]            GIOP v1.2  MsgLen: 12
   Endian: big  ReqId: 1
   Reply status (0) NO_EXCEPTION
```

For `MEDIUM` verbosity output, extra information is provided. The addition of time stamps (in *`hh:mm:ss`*) precedes each snoop line. The byte order of the data is indicated (`Endian`) along with more detailed header information such as the target address shown in this example. The target address is a GIOP 1.2 addition in place of the previous object key data.

HIGH verbosity client side snooping

The following is an example of `HIGH` verbosity output:

```
16:24:39 [Conn:1] Out:(first for binding)  [LocateRequest]    GIOP v1.2  MsgLen: 39
    Endian: big  ReqId: 0
    Target Address (0: KeyAddr)
      ObjKey (len 27) ':>.11..........A.........'
    GIOP Hdr (len 12): [47][49][4f][50][01][02][00][03][00][00][00][27]
    Msg Hdr  (len 39): [00][00][00][00][00][00][00][00][00][00][00][1b][3a][3e]
[02][31][31][0c][00][00][00][00][00][00][0f][05][00][00][41][c6][08][00][00][00]
[00][00][00][00][00]
[---- end of message ----]

16:31:37 [Conn:1] In: (first for binding)  [LocateReply]      GIOP v1.2  MsgLen: 8
    Endian: big  ReqId: 0
    Locate status: OBJECT_HERE
    GIOP Hdr (len 12): [47][49][4f][50][01][02][00][04][00][00][00][08]
    Msg Hdr  (len 8): [00][00][00][00][00][00][00][01]
[---- end of message ----]

16:31:37 [Conn:1] Out:  [Request]         GIOP v1.2  MsgLen: 60
    Endian: big  ReqId: 1  (two-way)
    Target Address (0: KeyAddr)
      ObjKey (len 27) ':>.11..........A.........'
    Operation (len 8) 'null_op'
    No. of Service Contexts: 0
    GIOP Hdr (len 12): [47][49][4f][50][01][02][00][00][00][00][00][3c]
    Msg Hdr  (len 60): [00][00][00][01][03][00][00][00][00][00][00][00][00][00]
[00][1b][3a][3e][02][31][31][0c][00][00][00][00][00][00][0f][05][00][00][41][c6]
[08][00][00][00][00][00][00][00][00][00][00][00][00][08][6e][75][6c][6c][5f][6f]
[70][00][00][00][00][00]
[---- end of message ----]

16:31:37 [Conn:1] In:   [Reply]           GIOP v1.2  MsgLen: 12
    Endian: big  ReqId: 1
    Reply status (0) NO_EXCEPTION
    No. of Service Contexts: 0
    GIOP Hdr (len 12): [47][49][4f][50][01][02][00][01][00][00][00][0c]
    Msg Hdr  (len 12): [00][00][00][01][00][00][00][00][00][00][00][00]
[---- end of message ----]
```

This level of verbosity includes all header data, such as service context data. ASCII-hex pairs of GIOP header and message header content are given to show the exact on-the-wire header values passing through the interceptor. Messages are also separated showing inter-message boundaries.

VERY HIGH verbosity client side snooping

This is the highest verbosity level available. Displayed data includes `HIGH` level output and in addition the message body content is displayed. Because the plug-in does not have access to IDL interface definitions, it does not know the data types contained in the body (parameter values, return values and so on) and simply provides ASCII-hex output. Body content display is truncated to a maximum of 4 KB with no output given for an empty body. Body content output follows the header output, for example:

```
...
GIOP Hdr (len 12): [47][49][4f][50][01][02][00][01][00][00][00][0c]
Msg Hdr  (len 12): [00][00][00][01][00][00][00][00][00][00][00][00]
Msg Body (len <x>): <content>
...
```

CHAPTER 15

Debugging IOR Data

Orbix includes iordump tool for analyzing IOR data and finding possible causes for badly formed IORs.

In this chapter

This chapter contains the following sections:

IOR Data Formats

Overview

CORBA Inter-operable Object Reference (IOR) data can be presented in one of two forms:

- Stringified form which is coded by converting each binary byte of coded data into an ASCII pair of characters representing the hex equivalent in readable form.
- CDR encoded (and aligned) binary data, which encodes each CORBA defined data type on its natural boundary. Short values are encoded on a 2-byte boundary, long values on a 4-byte boundary and, so on. Data contains padding between data types in order to ensure aligned data.

Stringified IOR data

Stringified IOR data is in the format `IOR:` followed by a series of hex value pairs. For example:

```
IOR:010000001c00000049444c3a53696d706c652f53696d706c654f626a6
```

It is best known as the CORBA IOR: URL passed to the IDL operation `CORBA::ORB::string_to_object()`. The stringified IOR data format of an encoded IOR can be obtained by using the IDL operation `CORBA::ORB::object_to_string()`.

IDL definition

Raw IOR data is encoded as the CDR representation of the IOR structure, defined in the CORBA GIOP specification, declared by the IDL shown in Example 3:

Example 3: *IOR data IDL definition*

```
// IDL
typedef unsigned long ProfileId;

const ProfileId TAG_INTERNET_IOP = 0;
const ProfileId TAG_MULTIPLE_COMPONENTS = 1;

// A TaggedProfile contains opaque profile and component
// data and a tag to indicate the type and format of the data.
struct TaggedProfile
  {
  ProfileId tag;
  sequence <octet> profile_data;
  };

// IOR is a sequence of object specific protocol profiles
// (TaggedProfiles) plus a type id.
struct IOR
  {
  string type_id;
  sequence <TaggedProfile> profiles;
  };

// A MultipleComponentProfile is contained in a TaggedProfile
// with the tag TAG_MULTIPLE_COMPONENTS.
typedef unsigned long ComponentId;

struct TaggedComponent
  {
  ComponentId tag;
  sequence <octet> component_data;
  };

typedef sequence <TaggedComponent> MultipleComponentProfile;
```

Example 3: *IOR data IDL definition*

```
// This declares IIOP ProfileBody data contained in a
// TaggedProfile with the tag TAG_INTERNET_IOP.
// IIOP 1.0/1.1/1.2 revisions are given.
struct Version
  {
  octet major;
  octet minor;
  };

struct ProfileBody_1_0
  {
  Version iiop_version;
  string host;
  unsigned short port;
  sequence <octet> object_key;
  };

struct ProfileBody_1_1
  {
  Version iiop_version;
  string host;
  unsigned short port;
  sequence <octet> object_key;
  sequence <IOP::TaggedComponent> components;   // Added in 1.1
  };

typedef ProfileBody_1_1 ProfileBody_1_2;        // Same as 1.1
```

Using iordump

Overview

`iordump` is a utility that decodes CORBA inter-operable object reference (IOR) content and presents it in readable format through `stdout`. This utility's output also includes debugging information to assist in analyzing the cause of malformed IOR data. On z/OS, `iordump` can be run as a job using the JCL in *`orbixhlq`*`.JCLLIB(IORDUMP)`.

Synopsis

```
iordump [-no_host_check] {file | -}
iordump [-no_host_check] IOR:...
```

When running `iordump` as a job, see the JCL in *`orbixhlq`*`.JCLLIB(IORDUMP)`.

Description

`iordump` reads the IOR data either from a specified file (`-` for `stdin`), or given as a command-line argument, and prints the detailed contents of the IOR data. The IOR may be specified either in the standard CORBA-defined stringified form or raw binary CDR encoded data. The IOR content is displayed in both stringified and ASCII-hex formats. The tools emphasis is on reporting all possible erroneous values or suspect data, while also displaying the meaning and value of each data item.

When running `iordump` as a job, the IOR is supplied as a file on the command line in the form:

`DD:`*`IORFILE`*

where *`IORFILE`* is the `DDNAME` that points to the file containing the IOR. Supplying the IOR as a command-line argument, or reading the IOR from `stdin` is not supported when running `iordump` as a job.

Parameters

`iordump` takes the following parameters:

`-no_host_check` The default behavior is to attempt a host lookup on each host specified in the IOR. This option prevents this host lookup check.

`file` Specifies the name of the file from which to read the IOR data. When running `iordump` as a job, use the form:

`DD:`*`IORFILE`*

where *`IORFILE`* is the `DDNAME` that points to the file containing the IOR.

-	Specifies that the IOR data is to be read from `stdin`. This is not supported when running `iordump` as a job.
IOR:...	Specifies the IOR to decode on the command line. Supplying the IOR on the command line is not supported when running `iordump` as a job.

Examples

To analyze the contents of a stringified IOR read from `stdin`:

```
> echo "IOR:..." | iordump -
```

To analyze the contents of the IOR generated by the simple CORBA demo:

```
> iordump simple1.ior
```

To analyze the contents of a stringified IOR specified as a command line argument:

```
> iordump IOR:000001.....
```

Running iordump as a job

To run `iordump` as a job on z/OS:

1. Edit the JCL in *orbixhlq*`.JCLLIB(IORDUMP)`.
2. Supply an appropriate job card for your installation.
3. Update `SET IORFILE=&ORBIX..DEMO.IORS(SIMPLE)` to point to the file containing the IOR that you want to dump. The file must contain a single IOR of the form `IOR:...` For example, `SET IORFILE=MY.IORS(IOR1)`
4. Update `SET OUTLCLE='IBM-1047'` if you are running in a different locale than `IBM-1047`. For example, `SET OUTLCLE='IBM-500'`.
5. Update `PPARM='DD:IORFILE'` in `STEP01` if you want to supply the `-no_host_check` parameter, or use a `DDNAME` other than `IORFILE`. If you use a `DDNAME` other than `IORFILE`, be sure to update the corresponding `DDNAME` in `STEP01`. For example, to run with the `-no_check_host parameter`, update `PPARM` as follows:
   `PPARM='-no_chost_check DD:IORFILE'`
6. Run the job. The job output contains the `iordump` results.

Notes

Data other than a single IOR in a file results in the whole data being analyzed as a single IOR. Trailing newlines, carriage returns, and nulls are removed only in the case of stringified IORs.

iordump Output

Overview

`iordump` decodes the IOR data provided and outputs the data to the screen in both stringified format and ASCII-hex fomat. All lines beginning with a '>>' prefix contain ASCII-hex data. Interspersed with the ASCII-hex data may be errors, warnings, and other data messages. These are explained in "Data, Warning, Error and Information Text" on page 231.

Example

Example 4 shows a sample output from `iordump`.

Example 4: *Sample iordump Output*

```
C:\>iordump simple1.ior

Stringified IOR is: ([string/coded data] length: 312 / 154 bytes)

>>
   IOR:010000001c00000049444c3a53696d706c652f53696d706c654f626a6
   563743a312e3000010000000000000006a000000010102000e00000036332e
   36352e3133332e32353000a70f1b0000003a3e0231310c00000000ec09000
   08d2000000800000000000000000002000000010000001800000001000000
   0100010000000000000101000100000009010100060000000600000001000
   0001100
------------------------------------------------------------
>> +0 [01]
        Byte order of IOR: (1) Little Endian
>> +1 [00][00][00]
        (padding)
>> +4 [1c][00][00][00]
        TypeId length: 28 bytes (including null)
>> +8
   [49][44][4c][3a][53][69][6d][70][6c][65][2f][53][69][6d][70][
   6c][65][4f][62][6a][65][63][74][3a][31][2e][30][00]
        TypeId value: 'IDL:Simple/SimpleObject:1.0.'
>> +36 [01][00][00][00]
        Number of tagged profiles: 1
```

Example 4: *Sample iordump Output*

```
        Profile 1:
>> +40 [00][00][00][00]
                Tag: (0) TAG_INTERNET_IOP
>> +44 [6a][00][00][00]
                Profile length: 106 bytes
>> +48 [01]
                Byte Order: (1) Little Endian
>> +49 [01][02]
                Version: 1.2
>> +52 [0e][00][00][00]
                Host length: 14 bytes (including null)
>> +56 [36][33][2e][36][35][2e][31][33][33][2e][32][35][30][00]
                Host string: '63.65.133.250.'
               * host IP address lookup succeeded, but failed to
   find a hostname (warning)
>> +70 [a7][0f]
                Port: 4007
>> +72 [1b][00][00][00]
                Object Key length: 27 bytes (including any
   trailing null)
>> +76
   [3a][3e][02][31][31][0c][00][00][00][00][ec][09][00][00][8d][
   20][00][00][
08][00][00][00][00][00][00][00][00]
                Object key data: ':>.11.......... ...........'
                (looks like an Orbix ART Transient key)
>> +103 [00]
       (padding)
>> +104 [02][00][00][00]
                Number of tagged components: 2
```

Example 4: *Sample iordump Output*

```
        Profile 1:
>> +40 [00][00][00][00]
                Tag: (0) TAG_INTERNET_IOP
>> +44 [6a][00][00][00]
                Profile length: 106 bytes
>> +48 [01]
                Byte Order: (1) Little Endian
>> +49 [01][02]
                Version: 1.2
>> +52 [0e][00][00][00]
                Host length: 14 bytes (including null)
>> +56 [36][33][2e][36][35][2e][31][33][33][2e][32][35][30][00]
                Host string: '63.65.133.250.'
               * host IP address lookup succeeded, but failed to
   find a hostname (warning)
>> +70 [a7][0f]
                Port: 4007
>> +72 [1b][00][00][00]
                Object Key length: 27 bytes (including any
   trailing null)
>> +76
   [3a][3e][02][31][31][0c][00][00][00][00][ec][09][00][00][8d][
   20][00][00][
08][00][00][00][00][00][00][00][00]
                Object key data: ':>.11.......... ...........'
                (looks like an Orbix ART Transient key)
>> +103 [00]
        (padding)
>> +104 [02][00][00][00]
                Number of tagged components: 2
```

Example 4: *Sample iordump Output*

```
        Profile 1:
>> +40 [00][00][00][00]
                Tag: (0) TAG_INTERNET_IOP
>> +44 [6a][00][00][00]
                Profile length: 106 bytes
>> +48 [01]
                Byte Order: (1) Little Endian
>> +49 [01][02]
                Version: 1.2
>> +52 [0e][00][00][00]
                Host length: 14 bytes (including null)
>> +56 [36][33][2e][36][35][2e][31][33][33][2e][32][35][30][00]
                Host string: '63.65.133.250.'
               * host IP address lookup succeeded, but failed to
   find a hostname (warning)
>> +70 [a7][0f]
                Port: 4007
>> +72 [1b][00][00][00]
                Object Key length: 27 bytes (including any
   trailing null)
>> +76
   [3a][3e][02][31][31][0c][00][00][00][00][ec][09][00][00][8d][
   20][00][00][
08][00][00][00][00][00][00][00][00]
                Object key data: ':>.11.......... ..........'
                (looks like an Orbix ART Transient key)
>> +103 [00]
       (padding)
>> +104 [02][00][00][00]
                Number of tagged components: 2
```

Example 4: *Sample iordump Output*

```
                Component 1:
>> +108 [01][00][00][00]
                          Tag: (1) CODE_SETS
>> +112 [18][00][00][00]
                          Component length: 24 bytes
>> +116 [01]
                          Component Byte Order: (1) Little Endian
>> +117 [00][00][00]
        (padding)
>> +120 [01][00][01][00]
                          Native CodeSet id (for char): 65537
                          (ISO 8859-1:1987; Latin Alphabet No. 1)
>> +124 [00][00][00][00]
                          Number of conversion code sets (CCS): 0
>> +128 [00][01][01][00]
                          Native CodeSet id (for wchar): 65792
                          (ISO/IEC 10646-1:1993; UCS-2, Level 1)
>> +132 [01][00][00][00]
                          Number of conversion code sets (CCS): 1
>> +136 [09][01][01][00]
                          CCS(1) CodeSet Id 65801
                          (ISO/IEC 10646-1:1993; UTF-16, UCS
   Transformation Format 16-bit form)

                Component 2:
>> +140 [06][00][00][00]
                          Tag: (6) ENDPOINT_ID_POSITION
>> +144 [06][00][00][00]
                          Component length: 6 bytes
>> +148 [01]
                          Component Byte Order: (1) Little Endian
>> +149 [00]
        (padding)
>> +150 [00][00]
                          EndpointId begin (index): 0
>> +152 [11][00]
                          EndpointId end (index): 17
```

Stringified Data Output

All output begins with the stringified IOR such as:

```
Stringified IOR is: ([string/coded data] length: 312 / 154 bytes)

>>
   IOR:010000001c00000049444c3a53696d706c652f53696d706c654f626a6
   563743a312e3000010000000000000006a000000010102000e00000036332e
   36352e3133332e32353000a70f1b0000003a3e0231310c00000000ec09000
   08d2000000800000000000000000002000000010000001800000001000000
   0100010000000000000101000100000090101000600000006000000010000
   0001100
```

The first line gives the string length as the number of characters in the following IOR string, including the `IOR:` prefix. The coded data length indicates the number of bytes of encoded data which is represented by the stringified IOR, as per the CDR rules for encoding IOR data.

ASCII-Hex Data Output

Display format

All ASCII-hex pairs are printed as `[ab]` pairs in the output, where *ab* is a character pair in the range `00` to `FF`.

Each line of ASCII-hex output contain segments of ASCII-hex data taken from the stringified IOR, including the byte offset of the data relative to the start of the equivalent binary coded IOR, beginning at byte zero:

```
>> +offset [ab][ab][ab]...
```

Example

For example, the following output text:

```
>> +4 [00][00][00][18]
```

indicates the four ASCII pairs which are coded four bytes into the IOR binary data, in this case being the `TypeId` string length value of 24 bytes.

Note also that all printed data is shown in the byte order as coded into the IOR. The above, for example, is the value 24 as coded on a Big Endian machine and is displayed as such regardless of the byte order of the machine iordump is running on. Iordump only byte-swaps the values, if needed, in order to decode and print their actual value.

Data, Warning, Error and Information Text

Overview

All other output consists of data text for each data type and its value, and any relevant text to inform of errors, warnings or simple informative message text of conditions detected for each specific data item.

Example

For example, the following output shows the data type/value output `TypeId length:...` and an error message indicating an invalid data value.

```
>> +4 [40][32][40][32]
          TypeId length: 843067968 bytes (including null)
          * bad TypeId sequence length (843067968)
```

In this section

This section discusses the following topics:

Errors

The errors include the following:

*** unknown** General error indicating the specified data value is not a known or standard value. This typically includes `Tag` values and other well known values.

*** number of profiles is zero (should at least have one!)** The IOR `TaggedProfile` sequence length value indicates there are no tagged profiles, only a `TypeId` string. If this is not the case, the length value may be set incorrectly to zero.

*** empty profile (zero length); skip to next profile** A`TaggedProfile` is of zero length. This may be possible although it is currently flagged as a possible error.

*** gone beyond the end of the profile data; must exit (number of profiles suggests more data)** The number of profiles value has caused `iordump` to skip beyond the end of the data. The tool expects to see more profiles. This occurs because the value is corrupt or has been coded in the IOR incorrectly. A few reasons for this error is: a value is encoded using the wrong alignment, or a value is decoded based on an incorrect byte order setting, or the wrong value was encoded.

*** unknown IIOP version (attempting to read as 1.0 data)** The `ProfileBody` is not one of the supported IIOP versions recognized by `iordump`. An attempt is made to interpret the initial part of the data as 1.0 IIOP profile data.

*** unknown profile tag/format** The profile tag is unknown, either because it is corrupt or because it is an unknown vendor-defined tag not registered with the OMG.

*** gone beyond the end of the component data; skip component** An invalid length has caused the component data to be exhausted. If possible, `iordump` will skip the invalid component data and move onto the next to the next component.

*** only one ORB_TYPE component allowed** The OMG specification only allows one `TAG_ORB_TYPE` component per profile, so the IOR is not OMG-compliant.

*** missing CodeSetComponent for wchar / * missing conversion code sets for wchar** A`TAG_CODE_SETS` component consists of two `CodeSetComponents`, one for `char` conversions and one for `wchar` conversions. Each `CodeSetComponent` is a struct containing a native `CodeSetId`, specified as a `ulong` and conversion code sets, specified as a sequence of `CodeSetId`. The encapsulated data contained in the tagged component is a `CodeSetComponentInfo` which is defined as follows:

```
typedef unsigned long CodeSetId;
struct CodeSetComponent
  {
  CodeSetId            native_code_set;
  sequence<CodeSetId> conversion_code_sets;
  };
struct CodeSetComponentInfo
  {
  CodeSetComponent ForCharData;
  CodeSetComponent ForWcharData;
  };
```

These errors are reported if part of this data structure is missing from the IOR tagged component.

*** null wchar native code set; client will throw INV_OBJREF** The CORBA specification includes a requirement that a native code set is specified at least for a server that supports the IDL `wchar` type because there is no default `wchar` conversion code set. If the native code set for `wchar` is set to zero this is an error and according to the spec; the client will throw an `INV_OBJREF` exception.

*** a zero string length is illegal, client will throw MARSHAL** A string is encoded as *<length><characters>* where the length includes a terminating `null`. All strings contain a `null`, therefore a zero length is illegal.

*** should be 0 or 1; assuming (1) Little Endian** The `octet` containing the byte order flag in an IOR may only contain the values `0` or `1` to indicate Big or Little Endian.

*** bad <data type> sequence length (<n>)** The length check on a `sequence<octet>` coded length value indicates an invalid length field.

*** stringified IOR should have an even length; added trailing'0' to continue**

The stringified IOR always contains an even number of characters because it contains ASCII-Hex pairs. An additional `0` is added to the data to allow it to be decoded and analyzed. Possible errors will result when analyzing the last bytes.

*** tried to skip <n> byte(s) of padding beyond the remaining data; exit..**

Tried to align for a data type when the alignment has skipped beyond the amount of remaining data.

*** attempt to read <n> byte data type, only <m> remaining; exit..** After skipping padding bytes and aligning to read the next data item, a check is also made that the number of bytes required to read the data type does not exceed what data is actually left to read.

*** no more data; exit..** Unexpectedly ran over the end of data.

Warnings

The warnings include the following.

*** non zero padding (warning)** This indicates that unused `octets` in the data contain non-zero values. Unused bytes exist because of required padding bytes between data values in order to maintain the correct data alignment. The CORBA specification does not insist on having all padding zeroed although this potentially creates problems when an IOR is published, or used for hashing, or any situation which results in two IORs being considered different simply because of differences in unused padding data.

*** no null character at end (warning)** In some cases, a `sequence<octet>` may be used to store string values. This warning indicates that a data value that can be interpreted as a string does not contain a terminating `null`. If the data is meant to be used as a string, this can cause problems when trying to decode and use the string. An example is the use of strings to represent the object key by some vendors. Otherwise, this warning may be ignored.

A simple mistake made when coding such a string is in using the string length given by `strlen(1)` to code the sequence length, without adding 1 for the `null`.

*** should TypeId begin with 'IDL:' prefix? (warning)** A check was made on the `TypeId` string and the expected `IDL:` prefix was not found.

*** num profiles sounds excessive, only printing <n>** If the value containing the number of profiles exceeds a reasonable limit (100 as set by `iordump`), only the number of profiles up to the limit is printed.

*** IOR contains <n> garbage trailing byte(s):** Any remaining bytes in the data, beyond the last decoded data value are printed before exit.

*** empty component data, zero length (warning)** A `TaggedComponent` length field indicates a zero length component.

*** previous component sequence length may be wrong (warning)** The sequence length of a previous component may be wrong and caused the data of the following component to be considered part of it. This is only a possible explanation for a missing component, particularly if the previous component reported an unknown or illegal data value.

*** host unknown; possibly unqualified (warning)** An attempt is made to do a lookup of the host contained in an IIOP profile. If the host lookup fails, this is printed as a warning. This would result if the host is really unknown, or is not fully qualified with the complete domain.

*** host name lookup succeeded, but failed to find an IP address (warning)**

The specified host lookup succeeded, but an attempt to lookup the IP address mapping for the specified host failed.

*** host IP address lookup succeeded, but failed to find a hostname (warning)**

The specified IP address lookup succeeded, but an attempt to lookup the host mapping for the specified address failed.

Part 4

Command Reference

In this part

This part contains the following chapters:

CHAPTER 16

Starting Orbix Services

This chapter describes commands that start Orbix services.

In this chapter

This chapter contains the following sections:

Starting and Stopping Configured Services

Start and stop scripts

The Orbix configuration tool generates two scripts that start and stop all configured Orbix services:

UNIX

```
start_domain-name_services.sh
stop_domain-name_services.sh
```

Windows

```
start_domain-name_services.bat
stop_domain-name_services.bat
```

The startup script starts all Orbix services you configured using the configuration tool. For example, given a domain name of `AcmeServices`, the following command starts all services on Windows:

```
start_AcmeServices_services.bat
```

Start-up order

Orbix services, when configured, start up in the following order:

1. Configuration repository
2. Locator daemon
3. Node daemon
4. Naming service
5. Interface repository
6. Event service

For example, you might decide to configure the event service but not the naming service. In this case, the event service takes a priority of 5.

Starting Orbix Services Manually

Orbix also provides separate commands for starting each service manually, with the following syntax:

```
itservice-name [run]
```

`run` is optional. For example, the following commands both start the interface repository:

```
itifr
itifr run
```

Table 8 lists all commands for running services manually:

Table 8: *Commands to Manually Start Orbix Services.*

Command	Starts
`itconfig_rep run`	Configuration repository
`itlocator run`	Locator daemon
`itnode_daemon run`	A node daemon
`itnaming run`	Naming service database
`itifr run`	Interface repository
`itevent run`	Event service
`itnotify run`	Notification service

Note: In a repository-based configuration domain, the configuration repository must be running before starting additional services.

itconfig_rep run

Synopsis

```
itconfig_rep -ORBdomain_name cfr-domain-name [-ORBname ORB-name]
[run] [-background]
```

Description

Starts the configuration repository. The configuration repository must already be configured in your Orbix environment. This command requires you to be logged in as administrator (Windows) or root (UNIX).

UNIX

You can push the process into the background.

Windows

Leave the command window open.

Options

`-ORBdomain_name` *`cfr-domain-name`*	The configuration repository's domain file name, which is generated when you create the domain. The generated configuration domain file has the name `cfr-`*`domain-name`*`.cfg`. For example, given configuration domain `acmeproducts`, the configuration repository initializes itself from `cfr-acmeproducts.cfg`.
`-ORBname` *`ORB-name`*	Directs the initializing configuration repository to retrieve its configuration from the specified configuration scope. By default, this is the `config_rep` scope. Use the `-ORBname` argument to specify a different configuration scope. For example: `itconfig_rep -ORBname config_rep.config2 run`
`-background`	Runs the configuration repository in the background. Control returns to the command line only after the service successfully launches. If you omit the `-background` argument, the configuration repository runs in the foreground. This argument can be abbreviated to `-bg`. For example: `itconfig_rep run -bg` The `-background` argument is especially useful in scripts that start multiple services. It guarantees that services always launch in the same sequence as the script specifies.

itlocator run

Synopsis

```
itlocator [-ORBname ORB-name] run [-background]
```

Description

Starts the locator daemon. The locator daemon must already be configured in your Orbix environment. In a location domain, the locator daemon controls read and write operations to the implementation repository. By default, entering `itlocator` without specifying the `run` command starts the default locator daemon.

UNIX

You can push the process into the background.

Windows

Leave the command window open.

Options

`-ORBname` *ORB-name*	Directs the initializing locator daemon to retrieve its configuration from the specified configuration scope. By default, this is the `locator` scope. Use the `-ORBname` argument to specify a different configuration scope. For example: `itlocator -ORBname locator.locator2 run`
`-background`	Runs the locator daemon in the background. Control returns to the command line only after the service successfully launches. If you omit the `-background` argument, the locator daemon runs in the foreground. You can abbreviate this argument to `-bg`. For example: `itlocator run -bg` The `-background` argument is especially useful in scripts that start multiple services. It guarantees that services always launch in the same sequence as the script specifies.

itnode_daemon run

Synopsis

```
itnode_daemon [-ORBname ORB-name] run [-background]
```

Description

Starts a node daemon. A node daemon controls registered server processes to ensure that they are always running, starts processes on demand, or disables them from starting. The node daemon also monitors all child processes of registered server processes, and informs the locator daemon about any events relating to these child processes—in particular, when a child process terminates. By default, entering `itnode_daemon` without specifying the `run` command starts the default node daemon.

UNIX

You can push the process into the background.

Windows

Leave the command window open.

Options

`-ORBname` *ORB-name*	Directs the initializing node daemon to retrieve its configuration from the specified configuration scope. By default, this is the `iona_services.node_daemon` scope. Use the `-ORBname` argument to specify a different configuration scope. For example: `itnode_daemon -ORBname iona_services.node_daemon.nd2 run`
`-background`	Runs the node daemon in the background. Control returns to the command line only after the service successfully launches. If you omit the `-background` argument, the node daemon runs in the foreground. You can abbreviate this argument to `-bg`. For example: `itnode_daemon run -bg` The `-background` argument is especially useful in scripts that start multiple services. It guarantees that services always launch in the same sequence as the script specifies.

`-ORBsecure_directories` Specifies a list of secure directories in which the node daemon launches processes. This overrides the path specified for the registered process. For example:

```
itnode_daemon -ORBsecure_directories
   [c:\Acme\bin,c:\my_app]
```

You must enclose the directory list in square brackets. If you omit this argument, the node daemon launches processes from the path specified in the location domain.

itnaming run

Synopsis

`itnaming [-ORBname ORB-name] run`

Description

Starts the naming service, assuming it is already configured in your Orbix environment. By default, entering `itnaming` without specifying the `run` command starts the naming service.

UNIX

You can push the process into the background.

Windows

Leave the command window open.

Options

`-ORBname ORB-name` Directs the initializing naming service to retrieve its configuration from the specified configuration scope.

By default, this is the `naming` scope. Use the `-ORBname` argument to specify a different configuration scope. For example:

```
itnaming -ORBname naming.naming2 run
```

`-background`	Runs the naming service in the background. Control returns to the command line only after the service successfully launches. If you omit the `-background` argument, the naming service runs in the foreground. You can abbreviate this argument to `-bg`. For example: `itnaming run -bg` The `-background` argument is especially useful in scripts that start multiple services. It guarantees that services always launch in the same sequence as the script specifies.

itifr run

Synopsis

`itifr [-ORBname ORB-name] run [-background]`

Description

Starts the interface repository daemon. The interface repository must already be configured in your Orbix environment. By default, entering `itifr` without specifying the `run` command starts the interface repository.

UNIX

You can push the process into the background.

Windows

Leave the command window open.

Options

`-ORBname ORB-name`	Directs the initializing interface repository to retrieve its configuration from the specified configuration scope. By default, this is the `ifr` scope. Use the `-ORBname` argument to specify a different configuration scope. For example: `itifr -ORBname ifr.ifr2 run`

`-background`	Runs the interface repository in the background. Control returns to the command line only after the service successfully launches. If you omit the `-background` argument, the interface repository runs in the foreground. You can abbreviate this argument to `-bg`. For example: `itifr run -bg` The `-background` argument is especially useful in scripts that start multiple services. It guarantees that services always launch in the same sequence as the script specifies.

itevent run

Synopsis

`itevent` *[*`-ORBname` *ORB-name]* `run` *[*`-background`*]*

Description

Starts the event service. The event service must already be configured in your Orbix environment. By default, entering `itevent` without specifying the `run` command starts the event service.

UNIX

You can push the process into the background.

Windows

Leave the command window open.

Options

`-ORBname` *ORB-name*	Directs the initializing event service to retrieve its configuration from the specified configuration scope. By default, this is the `event` scope. Use the `-ORBname` argument to specify a different configuration scope. For example: `itevent -ORBname event.event2 run`

`-background`	Runs the event service in the background. Control returns to the command line only after the service successfully launches. If you omit the `-background` argument, the event service runs in the foreground. You can abbreviate this argument to `-bg`. For example: `itevent run -bg` The `-background` argument is especially useful in scripts that start multiple services. It guarantees that services always launch in the same sequence as the script specifies.

itnotify run

Synopsis

`itnotify` *[*`-ORBname` *ORB-name]* `run` *[*`-background`*]*

Description

Starts the notification service. The notification service must already be configured in your Orbix environment. By default, entering `itnotify` without specifying the `run` command starts the event service.

UNIX

You can push the process into the background.

Windows

Leave the command window open.

Options

`-ORBname` *ORB-name*	Directs the initializing notification service to retrieve its configuration from the specified configuration scopes. By default, this is the `notify` scope. Use the `-ORBname` argument to specify a different configuration scope. For example: `itnotify -ORBname notify.notify2 run`

`-background`	Runs the notification service in the background. Control returns to the command line only after the service successfully launches. If you omit the `-background` argument, the notification service runs in the foreground. You can abbreviate this argument to `-bg`. For example: `itnotify run -bg` The `-background` argument is especially useful in scripts that start multiple services. It guarantees that services always launch in the same sequence as the script specifies.

Stopping Services Manually

Any service that can be started manually can also be stopped manually using `itadmin` commands. The order in which you shut down services should be determined by the dependencies among them. For example, in a repository-based domain, you should not shut down the configuration repository until all other services are shut down.

Shut-down commands have the following syntax:

```
itadmin service-name stop
```

Table 9 lists the `itadmin` commands for shutting down Orbix services:

Table 9: *Commands for Stopping Orbix Services*

Service	Shut-down command
Configuration repository	`itadmin config stop`
Locator	`itadmin locator stop`
Node daemon	`itadmin node_daemon stop`
Naming service	`itadmin ns stop`
Interface repository	`itadmin ifr stop`
Event service	`itadmin event stop`

CHAPTER 17

Event Log

Overview

The event log commands enable the Orbix event log filters to be displayed or updated dynamically using the `itadmin` command line. You can also perform these actions using the Administrator Web Console:

Table 10: *Event Log Commands*

`logging get`	Displays the event log filter settings.
`logging set`	Updates the event log filter.

logging get

Synopsis

```
logging get -orbname orb_name
```

Description

Displays the event log filter settings for the specified ORB name.

Arguments

`-orbname`	The specified ORB name of the event log to display.

Examples

```
itadmin logging set -orbname iona_services.naming
```

This command displays the event log filter settings that are used by the currently running naming service.

logging set

Synopsis

```
logging set -orbname orb_name -value new_event_log_filter
```

Description

Updates the event log filter settings for the specified ORB name.

Arguments

`-orbname`	The specified ORB name of the event log to update.
`-value`	The new event log setting.

Examples

```
itadmin logging set -orbname iona_services.naming -value
   IT_GIOP=*,IT_MGMT=*
```

This command updates the event log filters that are used by the currently running naming service.

CHAPTER 18

Managing Orbix Services With itadmin

This chapter provides an overview of using the command-line tool `itadmin` *to manage Orbix services. Typical management tasks in Orbix include creating, viewing, and removing data stored in service repositories.*

In this chapter

This chapter contains the following sections:

Using itadmin

Overview

`itadmin` lets you manage information used by Orbix services. You can use `itadmin` in various modes and contexts:

- Command-line utility
- Command shell
- Tcl script
- Transactions

Command-line utility

To use `itadmin` as a command-line utility, simply enter the appropriate command at the command prompt. For example, the following command registers an ORB name with the locator daemon:

```
itadmin orbname create my_orb_name
```

In command-line mode, you must specify the `itadmin` prefix before each command. For a list of itadmin commands, see "Services and Commands" on page 262.

Command shell

To use the `itadmin` shell, enter `itadmin` at the command line. The `itadmin` prompt is displayed. Once you have entered the command shell, you do not need to enter `itadmin` before each command. For example:

```
itadmin
% orbname create my_orb_name
```

To leave the `itadmin` shell mode, enter `exit`.

Nested itadmin commands

In shell and Tcl script mode, you can use nested `itadmin` commands by enclosing each command in square brackets. When `itadmin` commands are nested, innermost command are executed first.

Tcl script

You can write your own Tcl scripts that incorporate `itadmin` commands. For example, you could develop a Tcl script called `my_script` that contains one `itadmin` command per line. You would invoke this script by entering:

```
itadmin my_script.tcl
```

You can use Tcl scripts at the command prompt and in the command shell. Incorporating `itadmin` commands in reusable Tcl scripts provides an extremely powerful way of automating administration tasks (for example, populating a configuration domain or location domain).

Sample scripts

The following example shows the contents of a simple Tcl script that calls an `itadmin variable create` command:

```
if { [catch {variable create -type string -value poa
      initial_references:POACurrent:plugin} result] } {
          puts $result
          flush stdout
          exit 1
   }
```

This command creates a configuration variable named `initial_references:POACurrent:plugin` and assigns it a value of `poa`. The remaining Tcl in this simple example is used for Tcl script management. For example, `catch` prevents a Tcl stack dump if an exception is thrown during execution.

The following is a more realistic example of how to use `itadmin` commands within Tcl scripts:

```
# do_cmd installs an exception handler for each itadmin command

proc do_cmd {cmd} {
  set fail [catch {eval $cmd} result]
  if {$fail} {
     puts stderr "Problem in \"$cmd\": $result"
     flush stderr
     exit 1
  }
}

# Each itadmin command is sent as a parameter to do_cmd

  do_cmd {variable create -type string -value poa
          initial_references:RootPOA:plugin}
  do_cmd {variable create -type string -value poa
          initial_references:POACurrent:plugin}
  do_cmd {variable modify ... }
  do_cmd {poa create ...}
  exit 0
```

The `do_cmd` procedure installs an exception handler for each `itadmin` command. Each `itadmin` command is in turn sent as a parameter to `do_cmd`. For example, the first call to `do_cmd` creates `initial_references:RootPOA:plugin` and assigns it a value of `poa`.

Transactions

`itadmin` supports the object transaction service (OTS). Using `itadmin` commands in transactions provides `itadmin` with multiple undo capability.

Orbix provides `itadmin` commands to start, commit, rollback, suspend, and resume transactions. This enables you to use other `itadmin` commands in transactional mode. For more details, see "Object Transaction Service" on page 363.-

Multiple itadmin sessions

`itadmin` does not perform any record locking while it is making changes to the configuration database. Therefore, running multiple sessions of `itadmin` in parallel will corrupt your Orbix configuration.

Command Syntax

Overview

`itadmin` syntax takes the following general form:

`actor [actor modifiers] action [action modifiers] [target]`

For example, the following command registers a process name with the locator daemon:

```
orbname create -process  process-name  ORB-name
```

In this example, the *`actor`* is `orbname`, the *`action`* is `create`, the *`action modifier`* is `-process`, and the *`target`* is *`ORB-name`*.

Note: The order of `itadmin` components is significant. Each component must be separated by a space.

In this section

The following topics are discussed in this section:

Specifying lists

When a command takes a list, separate the list elements with spaces and enclose the entire list in double quotation marks. For example, the following command creates a server process entry in the location domain with the specified environment values:

```
% process create -env "mode=listen priority=low startup=yes"
   process-name
```

In this example, the value of the `-env` modifier is a list with three elements, and the equal sign is treated as a character.

Double quotation marks group a set of elements into a single entity in which spaces are not significant. For example, the `-args` argument to the `process create` command is treated as a single list element, which must be enclosed by double quotes:

```
% process create -args "foo bar baz" process-name
```

When using `itadmin` in command line mode, the quotation marks must be escaped or they will be stripped away by the command line interpreter. It is unnecessary to escape the quotation marks when using `itadmin` in shell or script modes.

Specifying negative values

When the first character of a value supplied to an argument is a minus sign or hyphen, you must supply an additional hyphen. For example:

```
-modifier --3
```

When the first character is not a hyphen, an additional hyphen is not necessary. For example:

```
-modifier 4,-1,99
```

You must supply an additional hyphen even if the first character is enclosed in quotation marks. For example:

```
% variable create -type long -value "--99" my_variable
```

Abbreviating command parameters

You can abbreviate all `itadmin` command parameters. For example, the following commands all have the same effect:

```
% orbname list -p process-name
% orbname list -pr process-name
% orbname list -pro process-name
  ...
% orbname list -process process-name
```

Abbreviations must be unique. For example, if two parameters begin with the same letter, their abbreviations must use at least the minimum number of letters that differentiate between them.

Note: Abbreviations are not supported on z/OS. You must supply full command parameter names on z/OS.

Obtaining help

To obtain command line help for `itadmin`, enter:

```
itadmin -help
```

You can obtain context-sensitive help by entering a command (in its entirety, or in part) and adding the keyword `help`. For example, for help on the `orbname create` command, enter any of the following:

```
% orbname -help
% orbname create -help
% orbname create -process -help
% orbname create -process process-name -help
% orbname create -process process-name ORB-name -help
% orbname create ORB-name -help
```

Services and Commands

In this section

The following sections group `itadmin` commands according to Orbix services:

CHAPTER 19

Bridging Service

Overview

The bridge service allows JMS and CORBA notification clients to share messages. `itadmin` provides a set of commands for managing the bridging service:

Table 11: *Bridging Service Commands*

`bridge create`	Creates a bridge.
`bridge destroy`	Destroys a bridge.
`bridge list`	Lists all of the instantiated bridges in a deployment.
`bridge show`	Displays the status of a bridge.
`bridge start`	Starts the flow of messages through a bridge.
`bridge stop`	Stops the flow of messages through a bridge.
`bridge suspend`	Suspends the flow of messages through a bridge.
`endpoint_admin show`	Displays a bridge's endpoint admin's name and the type of endpoints it supports.
`endpoint destroy`	Destroys an endpoint.

Table 11: *Bridging Service Commands*

`endpoint list`	Lists the endpoints associated with an endpoint admin.
`endpoint show`	Display the status and attributes of a particular endpoint for the specified bridge.

bridge create

Synopsis

```
bridge create [-source_admin IOR | INIT_REF_KEY] [-source_type topic
| queue | channel] -source_name source name [-sink_admin IOR |
INIT_REF_KEY] -sink_type [topic | queue | channel] -sink_name sink
name bridge name
```

Description

Creates a bridge.

Arguments

`-source_admin`	The IOR or initial reference of the administrative object used to connect to the message source. To use the default notification endpoint admin use `IT_NotificationEndpointAdmin`; to use the default JMS endpoint admin use `IT_JMSEndpointAdmin`.
`-source_type`	The type of object that passes messages into the bridge. It can take one of three values: topic if the messages originate from a JMS topic, queue if the messages originate from a JMS queue and channel if the messages originate from a notification channel.
`-source_name`	The name of the object that passes messages to the bridge.
`-sink_admin`	The IOR or initial reference of the administrative object used to connect to where messages are being forwarded. If the message source is a notification channel, the message sink should be a JMS `Destination`. To use the default notification admin use `IT_NotificationEndpointAdmin`; to use the default JMS admin use `IT_JMSEndpointAdmin`.
`-sink_type`	The type of object that receives messages from the bridge. It can take one of three values: topic if the messages are being forwarded to a JMS topic, queue if the messages are being forwarded to a JMS queue and channel if the messages are being forward to a notification channel.
`-sink_name`	The name of the object that receives messages from the bridge.
`bridge name`	The name of the bridge. This must be a unique string value that is used to identify this bridge.

bridge destroy

Synopsis `bridge destroy` *`bridge name`*

Description Destroys a bridge.

bridge list

Synopsis `bridge list`

Description Lists all of the instantiated bridges in a deployment.

bridge show

Synopsis `bridge show` *`bridge name`*

Description Displays the status of a bridge.

bridge start

Synopsis `bridge start` *`bridge name`*

Description Starts the flow of messages through a bridge.

bridge stop

Synopsis `bridge stop` *`bridge name`*

Description Stops the flow of messages through a bridge.

bridge suspend

Synopsis `bridge suspend` *`bridge name`*

Description Suspends the flow of messages through a bridge.

endpoint_admin show

Synopsis

```
endpoint_admin show [IOR | INIT_REF_KEY]
```

Description

Displays a bridge's endpoint admin's name and the type of endpoints it supports.

endpoint destroy

Synopsis

```
endpoint destroy [-source | -sink] [-admin IOR | INIT_REF_KEY] bridge name
```

Description

Destroys an endpoint.

Arguments

`-source | -sink`	Specify whether the endpoint is a message source or a message sink.
`-admin`	Specify what type of admin object with which it is associated.

endpoint list

Synopsis

```
endpoint list [-source | -sink] [-admin IOR | INIT_REF_KEY]
```

Description

Lists the endpoints associated with an endpoint admin.

Arguments

`-source | -sink`	Specify whether the endpoint is a message source or a message sink.
`-admin`	Specify what type of admin object with which it is associated.

endpoint show

Synopsis

`endpoint show [-source | -sink] [-admin` *`IOR | INIT_REF_KEY]`* *`bridge name`*

Description

Display the status and attributes of a particular endpoint for the specified bridge.

Arguments

`-source	-sink`	Specify whether the endpoint is a message source or a message sink.
`-admin`	Specify what type of admin object with which it is associated.

JMS Broker

Overview

The Java Messaging Service (JMS) provides a native mechanism for Java applications to participate in messaging systems.

`itadmin` provides a set of commands for managing the JMS broker:

Table 12: *JMS Broker Commands*

`jms start`	Starts the JMS broker.
`jms stop`	Shuts down the JMS broker.

jms start

Synopsis

```
jms start
```

Description

Starts the JMS broker.

jms stop

Synopsis

```
jms stop
```

Description

Shuts down the JMS broker.

CHAPTER 20

Configuration Domain

Overview

A subset of `itadmin` commands let you manage a configuration domain, both file-based and configuration repository-based. These commands manage the following components of a configuration domain:

Note: To use `itadmin` in a repository-based configuration domain, the configuration repository must be running (see "Starting Orbix Services" on page 239).

Configuration Repository

Overview

The following commands enable you to manage the configuration repository (CFR):

Table 13: *Configuration Repository Commands*

`config dump`	Displays the entire contents of the configuration domain.
`config list_servers`	Shows all deployed replicas of the configuration repository.
`config stop`	Stops the configuration repository.
`file_to_cfr.tcl`	Converts from a file-based to a CFR-based configuration.

config dump

Synopsis

```
config dump [-compatible]
```

`-compatible`	Formats the CFR configuration so that it can be used in a file-based configuration. You can copy the output into a configuration file.

Description

Outputs the entire contents of the configuration domain to `stdout` in a form similar to a configuration file.

Examples

The following extract shows the values of some initial object references and plug-ins in the `initial_references` configuration namespace:

```
itadmin config dump
...
initial_references:IT_Locator:reference =
   "IOR:010000002500000049444c3a696...723a312e300000000001000000
   00001a00"

initial_references:POACurrent:plugin = "poa"

initial_references:NameService:reference =
   "IOR:010000002f00000049444c3a696f6e61...2e6362f49545f4e616d69
   6e606000000010000003500"

initial_references:DynAnyFactory:plugin = "it_dynany"

initial_references:ConfigRepository:reference =
   "IOR:010000002000000049444c3a495000002000...00006000000010000
   000900"
...
```

config list_servers

Synopsis

```
config list_servers [-active]
```

Description

Shows all active deployed replicas of the configuration repository.

Arguments

`-active` Displays the total number of active deployed replicas.

config show_server

Synopsis

```
config show_server cfr replica name
```

Description

Displays runtime information about the specified CFR server.

config stop

Synopsis

```
config stop [replica-name | -ior replica-ior]
```

Description

Stops the configuration repository. An unqualified `config stop` command stops all running replicas of the configuration repository.

Arguments

replica-name	Stops the specified replica of the configuration repository. You can obtain the replica's name with `itadmin config list`.
`-ior` *replica-ior*	Stops the specified replica, as specified by its IOR.

file_to_cfr.tcl

Synopsis

```
file_to_cfr.tcl [-scope scope] [-output_to_file file]
```

Description

Converts from a file-based configuration to a CFR-based configuration. Running this script creates `itadmin variable create` arguments in the output file, which you can then run against a CFR.

Examples

The recommended way to run this is to set `$IT_DOMAIN_NAME` to your file-based domain name, and execute the script. Then set `$IT_DOMAIN_NAME` to your CFR domain name, and finally run the generated output script.

Because a file-based configuration contains no data type information, the `file_to_cfr.tcl` script must make educated guesses about the types being processed. However, you can edit the generated script to ensure that the correct data types were chosen before running it against your CFR.

Note: Because this tcl script creates a temporary file, the user will need write access to the current directory.

Arguments

`-scope`	Processes configuration in the specified scope only.
`-output_to_file <`*`filename`*`>`	Specifies the newly generated script used to populate a CFR.

If the `-scope` argument is omitted, the script processes the whole configuration. If the `-output_to_file` argument is omitted, the output goes to `stdout` instead.

Namespaces

Overview

The following commands let you manage configuration namespaces:

Table 14: *Configuration Namespace Commands*

`namespace create`	Creates namespaces in the specified scope.
`namespace list`	Lists the namespaces in the given namespace or configuration scope.
`namespace remove`	Removes a namespace and all its contained namespaces and variables from the configuration domain.
`namespace show`	Displays all sub-namespaces, variables and their values contained within a namespace.

namespace create

Synopsis

```
namespace create [-scope scoped-name] namespace
```

Description

Creates a namespace and any intermediate namespaces, if they do not already exist.

Arguments

`-scope`	Creates the namespace in the specified scope. If you omit this argument, the namespace is created in the root scope.

Examples

The following example creates the `plugins:local_log_stream` namespace within the `node_daemon` configuration scope:

```
itadmin namespace create -scope node_daemon
   plugins:local_log_stream
```

namespace list

Synopsis

```
namespace list [-scope scoped-name] [namespace]
```

Description

Lists the namespaces in the specified namespace or configuration scope. If you specify a namespace, `itadmin` lists only the namespaces nested within it; otherwise, it shows all namespaces within the specified or root scope.

Arguments

`-scope`	Narrows the namespaces to a specific configuration scope. If you omit this argument, namespaces in the root scope are listed.

Examples

The following example lists namespaces in the root configuration scope:

```
itadmin namespace list
binding
plugins
url_protocols
url_resolvers
domain_plugins
initial_references
```

The following example lists namespaces nested within the `initial_references` namespace:

```
itadmin namespace list initial_references
PSS
RootPOA
PICurrent
IT_Locator
POACurrent
NameService
XAConnector
EventService
IT_Activator
DynAnyFactory
IT_NodeDaemon
...
IT_MulticastReliabilityProtocol
```

namespace remove

Synopsis

```
namespace remove [-scope scoped-name] namespace
```

Description

Removes a namespace.

Arguments

`-scope`	Removes the namespace from the specified scope. If you omit this argument, the namespace is removed from the root scope.

namespace show

Synopsis

```
namespace show [-scope scoped-name] namespace
```

Description

Displays all namespaces, variables and their values within the specified namespace.

Arguments

`-scope`	Narrows the namespaces to a specific scope. If you omit this argument, namespaces and their contents in the root scope are displayed.

Examples

The following example shows the contents of the `initial_references` namespace in the root configuration scope:

```
itadmin namespace show initial_references
initial_references:RootPOA:plugin = "poa";
initial_references:POACurrent:plugin = "poa";
initial_references:DynAnyFactory:plugin = "it_dynany";
initial_references:TransactionCurrent:plugin = "ots_lite";
initial_references:TransactionFactory:plugin = "ots_lite";
initial_references:PSS:plugin = "pss_db";
initial_references:NameService:reference = "IOR:0100...00900";
initial_references:ConfigRepository:reference="IOR:0100...00900"
   ;
initial_references:IT_Locator:reference = "IOR:0100...00900";
```

Scopes

Overview

The following commands let you manage configuration scopes:

Table 15: *Configuration Scope Commands*

`scope create`	Creates a configuration scope.
`scope list`	Displays all sub-scopes defined within a scope.
`scope remove`	Removes a configuration scope and all its contained namespaces, variables, and scopes.
`scope show`	Displays all namespaces, variables, and their values defined within a scope.

scope create

Synopsis

`scope create` *`scoped-name`*

Description

Creates a configuration scope. Unless qualified by higher-level scope names, the scope is created in the root configuration scope. To create a scope in a scope other than the root, specify its fully qualified name.

Examples

For example, the following command creates the `test` scope within `company.production`:

```
itadmin scope create company.production.test
```

After you create the scope, add the desired namespaces and variables within it with `itadmin variable create` and `itadmin namespace create`.

scope list

Synopsis

`scope list [`*`scoped-name`*`]`

Description

Lists all the sub-scopes in the specified configuration scope. If no scope is specified, this command lists the sub-scopes in the root scope.

Examples

The following command lists all the sub-scopes defined within the `node_daemon` configuration scope:

```
itadmin scope list node_daemon
node_daemon2
node_daemon3
```

scope remove

Synopsis

`scope remove` *`scoped-name`*

Description

Removes the specified scope from the configuration. This includes all its contained namespaces, variables, and configuration scopes.

scope show

Synopsis

`scope show [`*`scoped-name`*`] [-compatible] [-output_to_file` *`filename`*`]`

Description

Displays all sub-namespaces, variables, and their values in the specified configuration scope. If no scope is specified, this command displays the contents of the root scope.

Arguments

`-compatible`	Formats the displayed configuration so that it can be used in a file-based configuration. This enables you to produce file-based configuration segments from a scope (rather than the entire CFR).
`-output_to_file <`*`filename`*`>`	Directs the output to the specified file.

Examples

The following command displays the contents of the `node_daemon` configuration scope:

```
itadmin scope show node_daemon
orb_plugins = local_log_stream, iiop_profile, giop, iiop;
event_log:filters=IT_NODE_DAEMON=INFO_ALL+WARN+ERROR+FATAL;
plugins:node_daemon:shlib_name = "it_node_daemon_svr";
plugins:node_daemon:nt_service_dependencies = "IT locator
   orbix2000";
plugins:node_daemon:name = "it_node_daemon";
```

Variables

Overview

The following commands let you manage configuration variables:

Table 16: *Configuration Variable Commands*

`variable create`	Creates a variable or namespace within the configuration domain.
`variable modify`	Changes one or more variable values.
`variable remove`	Removes a variable from the configuration domain.
`variable show`	Displays a variable and its value.

variable create

Synopsis

`variable create [-scope` *`scoped-name`*`] -type long|bool|list|string -value` *`value var-name`*

Description

Creates the specified variable in the configuration domain. Any configuration namespaces specified in the variable name that do not exist are also created.

Arguments

The following arguments are supported:

`-scope` *`scoped-name`*

The configuration scope in which to define the variable. If you omit this argument, the variable is created in the root configuration scope.

`-type` *`type`*

The type of the variable. Supply one of the following types:

- `long`
- `bool`
- `list` (a comma-separated list of strings)
- `string`

For more about variable types, see "Data types" on page 45.

`-value` *value*

The variable's value. The value must match the type specified by the `-type` switch.

The following values are valid for the specified type:

long: any signed long value

bool: `true` or `false`

list: list items must be separated by commas. Empty elements or list items containing spaces must be quoted—for example:

```
foo,"bar none",baz
```

See "Specifying lists" on page 259 for more details.

string: Enclose values in double quotes.

Examples

The following example creates a variable named `orb_plugins` in the root configuration scope:

```
itadmin variable create -type list -value IIOP,GIOP,PSS
   orb_plugins
```

The following example creates variable `service_name` in scope `IFR`:

```
itadmin variable create -scope IFR -type string -value "ARTIFR"
   service_name
```

The following example creates a namespace in the root configuration scope:

```
itadmin variable create -type string -value
   "IOR:00433243423523423593 3..."
   initial_references:IntefaceRepository:reference
```

Note: In shell mode, do not specify IORs to the `-value` argument. Specify IORs in command-line and script modes only.

variable modify

Synopsis

```
variable modify [-scope scoped-name] -type long|bool|list|string
-value value var-name
```

Description

Modifies the value of a variable or namespace in the configuration domain in the specified scope.

Arguments

The following arguments are supported:

`-scope` *scoped-name*
: The configuration scope in which to modify the variable or namespace. The default is the root configuration scope.

`-type` *type*
: The type of the variable. Supply one of the following types:
- `long`
- `bool`
- `list` (a comma-separated list of strings)
- `string`

`-value` *value*
: The variable's value. The value must match the type specified by the `-type` switch.

The following values are valid for the specified type:

long: any signed long value

bool: `true` or `false`

list: list items must be separated by commas. Empty elements or list items containing spaces must be quoted—for example:

```
foo,"bar none",baz
```

See "Specifying lists" on page 259 for more details.

string: Enclose values in double quotes.

Examples

The following example modifies the event log filters for the naming service:

```
itadmin variable modify -scope naming -type list -value
   IT_NAMING=ERR+FATAL event_log:filters
```

variable remove

Synopsis

```
variable remove [-scope scoped-name] var-name
```

Description

Removes the specified variable from the configuration domain. This operation does not remove a configuration namespace.

Arguments

`-scope scoped-name`	The configuration scope from which to remove the variable. If you omit this argument, the variable is removed from the root scope.

variable show

Synopsis

```
variable show [-scope scoped-name] var-name
```

Description

Displays the specified variable and its value, within the specified scope. The default is the root configuration scope.

Arguments

`-scope`	Narrows the displayed variable to a specific configuration scope.

Examples

The following example shows a variable in the default root configuration scope:

```
itadmin variable show orb_plugins
orb_plugins = iiop_profile, giop, iiop
```

The following example shows the same variable as it is set for the event service in the configuration scope `event`:

```
itadmin variable show -scope iona_services.event orb_plugins
orb_plugins = iiop_profile, giop, iiop
```

CHAPTER 21

Event Service

Overview

The event service is a CORBA service that enables applications to send events that can be received by any number of objects. For more about the event service, see the *CORBA Programmer's Guide*.

`itadmin` commands let you manage the following event service components:

Event Service Management

Overview

The following commands let you manage an event service instance:

Table 17: *Event Service Commands*

`event show`	Displays the attributes of the specified event service.
`event stop`	Stops an instance of the event service.

event show

Synopsis

```
event show
```

Description

Displays the attributes of the default event service.

Multiple instances of the event service are also supported. To show the attributes of a non-default event service, specify the ORB name used to start the event service (using the `-ORBname` parameter to `itadmin`).

Examples

The following command shows the attributes of a default event service:

```
itadmin event show
Event Service Name: IT_EventNamedRoot
  Host Name: podge
  Event Channel Name List:
    my_channel
```

The following command shows the attributes of a non-default event service:

```
itadmin -ORBname event.event2 event show
Event Service Name: IT_EventNamedRoot2
  Host Name: rodge
  Event Channel Name List:
    my_channel
    my_channel2
```

Each event service instance must have a unique name. You can specify this is in your configuration, using the `plugins:poa:root_name` variable. The event service uses named roots to support multiple instances.

In this example, the `plugins:poa:root_name` variable is set to `IT_EventNamedRoot2` in the `event.event2` configuration scope:

```
...
event{
    plugins:poa:root_name = "IT_EventNamedRoot";
    ...

    event2
    {
     plugins:poa:root_name = "IT_EventNamedRoot2";
    };
}
...
```

event stop

Synopsis

```
event stop
```

Description

Stops the default event service.

Multiple instances of the event service are also supported. To stop a non-default event service, qualify the `itadmin` command with the `-ORBname` argument and supply the ORB name used to start the event service.

To start the event service, use the `itevent` command. You can also use the `start_domain-name_services` command. For more information, see "Starting Orbix Services" on page 239.

Examples

The following command stops the default event service.

```
itadmin event stop
```

The following command stops the event service that was started with ORB name `event.event2`:

```
itadmin -ORBname event.event2 event stop
```

Event Channel

The following commands let you manage an event channel:

Table 18: *Event Channel Commands*

ec create	Creates an untyped event channel with the specified name.
ec create_typed	Creates a typed event channel with the specified name.
ec list	Displays all untyped event channels managed by the event service.
ec remove	Removes the specified untyped event channel.
ec remove_typed	Removes the specified typed event channel.
ec show	Displays all attributes of the specified untyped event channel.
ec show_typed	Displays all attributes of the specified typed event channel.

ec create

Synopsis

`ec create` *channel-name*

Description

Creates an untyped event channel with the specified name. If specified with an unqualified itadmin command, the event channel is created in the default event service. You can create an event channel in another (non-default) event service by qualifying the `itadmin` command with the `-ORBname` argument and supplying the ORB name used to start the service.

Examples

The following command creates an untyped event channel, `my_channel`:

```
itadmin ec create my_channel
```

The following command creates an untyped event channel (for a non-default event service) named `my_channel2`:

```
itadmin -ORBname event.event2 ec create my_channel2
```

ec create_typed

Synopsis

`ec create_typed` *`channel_name`*

Description

Creates a typed event channel with the specified name.

ec list

Synopsis

`ec list [-count]`

Description

Displays all the untyped event channels managed by an event service.

Arguments

`-count` Displays the total number of untyped event channels.

Examples

The following example displays the untyped event channels that are in the default event service:

```
itadmin ec list
my_channel
mkt_channel
eng_channel
```

The following example displays the untyped event channels that are in a non-default event service:

```
itadmin -ORBname event.event2 ec list
my_channel
my_channel2
mkt_channel
eng_channel
```

The following example displays the number of untyped event channels managed by an event service:

```
itadmin ec list -count
3
```

ec remove

Synopsis

`ec remove` *`channel-name`*

Description

Removes the specified untyped event channel.

Examples

The following command removes untyped event channel `my_channel`:

```
itadmin ec remove my_channel
```

The following command removes untyped event channel `my_channel2` from a non-default event service:

```
itadmin -ORBname event.event2 ec remove my_channel2
```

ec remove_typed

Synopsis

`ec remove_typed` *`channel_name`*

Description

Removes the specified typed event channel.

ec show

Synopsis

`ec show` *`channel-name`*

Description

Displays all attributes of the specified untyped event channel.

Examples

The following command displays the attributes of `my_channel`:

```
itadmin ec show my_channel
Channel Name: my_channel
  Channel ID: 1
  Event Communication: Untyped
```

The following command displays the attributes of `my_channel2` from a non-default event service:

```
itadmin -ORBname event.event2 ec show my_channel2
Channel Name: my_channel2
  Channel ID: 2
  Event Communication: Untyped
```

Note: For information about event service configuration variables, see the section on the `plugins:notification` namespace in the *Orbix Configuration Reference*.

ec show_typed

Synopsis

`ec show_typed` *`channel_name`*

Description

Displays all attributes of the specified typed event channel.

CHAPTER 22

Interface Repository

Overview

A subset of `itadmin` commands let you create, browse, and remove IDL definitions from the interface repository. You can manage the following interface repository components:

IDL Definitions

Overview

`itadmin` provides a single `itadmin idl` command, which lets you modify the contents of an interface repository with new IDL definitions.

idl -R=-v

Synopsis

```
idl -R=-v idl-filename
```

Description

Writes IDL definitions from a single IDL source file into the interface repository. The `-R=-v` argument setting causes the interface repository to use verbose mode to indicate command progress. The *`idl-filename`* argument names the IDL file. You must execute the `idl` command from the command line.

Examples

The following example writes the IDL definitions in the `foo.idl` file to the interface repository:

```
bash $ idl -R=-v foo.idl
  Created Alias MyLong.
  Created Operation op1.
  Created Operation op2.
  Created Interface Foo.
```

Note: The `idl -R=-v` command does not require the `itadmin` command.

Repository Management

Overview

The following commands let you browse and modify the contents of an interface repository:

Table 19: *Interface Repository Commands*

`ifr cd`	Changes the current container (in shell mode).
`ifr destroy_contents`	Destroys the contents of the interface repository.
`ifr ifr2idl`	Outputs the contents of the interface repository to the specified file.
`ifr list`	Lists the contents of the current container.
`ifr pwd`	Prints the name of the current container (in shell mode).
`ifr remove`	Removes an IDL definition from the interface repository.
`ifr show`	Prints specified IDL definitions contained in the interface repository.
`ifr stop`	Stops the interface repository.

ifr cd

Synopsis

`ifr cd [`*`scoped-name`*` | .. ]`

Description

Changes the current container to the specified scoped name. Using the argument "`..`" changes the current container to the next outermost container. If no arguments are given, `ifr cd` changes the current container to the interface repository. Use `ifr cd` in command shell mode only.

Examples

The following command changes to the specified scoped name:

```
itadmin ifr cd MYCO.PRODUCTION.TOOLS
```

ifr destroy_contents

Synopsis

```
ifr destroy_contents
```

Description

Destroys the entire contents of the interface repository, leaving the repository itself intact.

ifr ifr2idl

Synopsis

```
ifr ifr2idl filename
```

Description

Converts the entire contents of the interface repository to text and writes it to the specified *filename*.

ifr list

Synopsis

```
ifr list [-l] [ scoped-name | . ]
```

Description

Lists the contents of the specified container. If no container name is provided, this command lists the contents of the current container.

Arguments

`-l`	Lists the contents in long form: absolute name, kind, repository ID.
scoped-name	Specifies the container to list the contents of. The default is the root name.
. (dot)	Specifies the current container.

ifr pwd

Synopsis

```
ifr pwd
```

Description

Displays the name of the current container. Use `ifr pwd` in command shell mode only. Command-line mode does not store persistent state.

ifr remove

Synopsis `ifr remove` *`scoped-name`*

Description Removes the scoped name by invoking the function `IRObject::destroy()` on the scoped name. The *`scoped-name`* argument is the name of the interface repository entry to be removed, and is relative to the current container.

ifr show

Synopsis `ifr show` *`scoped-name`*

Description Displays the scoped name in IDL format. The *`scoped-name`* argument is relative to the current container.

ifr stop

Synopsis `ifr stop`

Description Stops the interface repository.

CHAPTER 23

Location Domain

Overview

This section describes `itadmin` commands that manage a location domain and its components. Some commands modify static information in the implementation repository; others affect runtime components.

`itadmin` commands let you manage the following location domain components:

Locator Daemon

Overview

The following commands manage locator daemons:

Table 20: *Locator Daemon Commands*

`locator heartbeat_daemons`	Pings all the of the node daemons known to the specified locator, removing those that are no longer active.
`locator list`	Displays all locators in the location domain.
`locator show`	Displays all attributes of the specified locator daemon.
`locator stop`	Stops the locator daemon.

Locator daemon name

Most commands require you to supply the locator daemon name. The default name has the following format:

```
iona_services.locator_daemon.unqualified-hostname
```

For example:

```
iona_services.locator_daemon.oregon
```

locator heartbeat_daemons

Synopsis

```
locator heartbeat_daemons locator_name
```

Description

Pings all the of the node daemons known to the specified locator, removing those that are no longer active.

locator list

Synopsis

```
locator list [-count] [-active]
```

Description

Displays all locators in the location domain.

Arguments

`-count` Displays the number of locators in the location domain.

`-active` Displays all active locators in the location domain.

locator show

Synopsis

```
locator show [-ior] locator-name
```

Description

Displays all attributes of the specified locator.

Arguments

`-ior` Indicates that the target is an IOR, rather than the name of the Locator.

Examples

The following example shows the attributes displayed for a default locator:

```
itadmin locator show iona_services.locator.wicklow
Locator Name:   iona_services.locator
  Domain name:  enterprise_services
  Host name:    wicklow
  Start time:   Sun, 05 Aug 2001 07:55:59.5380000 +0500
  Replica type: Master
```

The following example shows the attributes for a locator running on `wicklow`, port 3076.

```
itadmin locator show -ior corbaloc::1.2@wicklow:3076/IT_Locator
Locator Name:   iona_services.locator
  Domain name:  enterprise_services
  Host name:    wicklow
  Start time:   Sun, 05 Aug 2001 07:55:59.5380000 +0500
  Replica type: Master
```

locator stop

Synopsis

```
locator stop [-alldomain] [-ior] locator-name
```

Description

Stops the specified locator daemon.

Arguments

`-alldomain`	Stops the locator, all registered node daemons, and monitored processes running in a location domain.
`-ior`	Indicates that the target is an IOR, rather than the name of the Locator.

Named Key

Overview

Named keys allow users to specify human readable URLs in place of a server's IOR. Named keys work best when used with persistent objects. If the object's IOR changes, the named key will need to recreated.

To pass the IOR of a server to a client using a named key, the user will need to supply an address is the following format:

```
corbaloc:iiop:ver@host:port/named_key
```

ver	The IIOP version the server uses to communicate.
host	The hostname for the machine running the locator daemon.
port	The port used by the locator.
named_key	The named key created for the server.

For example, the corbaloc reference for a replicated locator daemon would look like:

```
corbaloc:iiop:1.2@fox:8035,iiop:1.2@hound:8035/hunter
```

One instance of the locator daemon is hosted on `fox` and listens on port 8035. The other instance is hosted on `hound` and also listens on port 8035. The named key associated with this replicated locator daemon's IOR is `hunter`.

For more information on corbaloc references read section 13.6.10, "Object URLs," of the OMG CORBA specification.

Commands

The following commands let you manage named keys:

Table 21: *Named Key Commands*

`named_key create`	Creates an association between a specified well-known object key and a specified object reference.

Table 21: *Named Key Commands*

named_key list	Lists all well known object keys that are registered with the locator daemon.
named_key remove	Removes the specified *object-key* from the location domain.
named_key show	Displays the object reference associated with the given key.

named_key create

Synopsis

```
named_key create -key object-key object-reference
```

Description

Associates a well-known object key name with an object reference. The `-key` argument specifies the human-readable string name of the key to use when referring to the specified *object-reference*.

After entering this command, object requests destined for the specified object key are forwarded to the specified object reference.

Use `named_key create` in command-line mode only.

Examples

The following example shows the named key created for the default naming service when Orbix is installed:

```
itadmin named_key create -key NameService IOR:010000002...003500
```

named_key list

Synopsis

```
named_key list [-count]
```

Description

Lists all well-known object keys registered in the location domain.

Arguments

`-count` Displays the number of well-known object keys in the location domain.

Examples

The following command lists the named keys that are created in a default Orbix environment:

```
itadmin named_key list
NameService
InterfaceRepository
```

named_key remove

Synopsis

```
named_key remove object-key
```

Description

Removes the specified human-readable *object-key* from the location domain.

named_key show

Synopsis

```
named_key show object-key
```

Description

Displays the object reference associated with the specified human-readable *object-key*.

Examples

```
itadmin named_key show NameService
Named Object Key           : NameService
Associated Object Reference:
   IOR01000002f0000004944...00100003500
```

Node Daemon

Overview

The following commands manage node daemons:

Table 22: *Node Daemon Commands*

`node_daemon list`	Displays all node daemon names implicitly registered with the locator daemon.
`node_daemon remove`	Removes a node daemon from the location domain that is created implicitly when the specified node daemon starts.
`node_daemon show`	Displays all attributes of the specified node daemon.
`node_daemon stop`	Stops the node daemon.
`add_node_daemon.tcl`	Adds node daemons to a host.

Node daemon name

Most commands require you to supply the node daemon name. The default name has the following format:

```
iona_services.node_daemon.unqualified-hostname
```

For example:

```
iona_services.node_daemon.oregon
```

node_daemon list

Synopsis

```
node_daemon list [-count]
```

Description

Displays all node daemon names implicitly registered with the locator daemon. Node daemon entries are implicitly created in the implementation repository (IMR) when the specified node daemon starts.

Arguments

`-count` Displays the total node daemon count.

node_daemon remove

Synopsis

`node_daemon remove` *`node-daemon-name`*

Description

Removes a node daemon entry from the implementation repository. Node daemon entries are created implicitly when the specified node daemon starts.

Use this command only when the specified node daemon shuts down prematurely due to a host crash or termination signal.

WARNING: Do not use `node_daemon remove` on a running node daemon.

node_daemon show

Synopsis

`node_daemon show` *`node-daemon-name`*

Description

Displays the attributes for the specified node daemon.

Examples

The following example shows the attributes displayed for the node daemon on host `dali`:

```
itadmin node_daemon show dali
Node Daemon Name: dali
  Host Name: dali
  File Access Permissions:
  User:  mstephens
  Group: o2kadm
  Start time: Mon, 06 Aug 2001 06:55:53.4480000 +0500
```

The default node name is *`host`*. To change the default name, modify `plugins:node_daemon:name`, using `itadmin variable modify`. In a file-based configuration domain, you can also edit this variable in your configuration file.

node_daemon stop

Synopsis

```
node_daemon stop node-daemon-name
```

Description

Stops the specified node daemon. This command also stops all the processes monitored by that node daemon.

To view all processes monitored by the specified node daemon, use `process list -node_daemon`.

add_node_daemon.tcl

Synopsis

```
itadmin add_node_daemon.tcl -number<add> -port <base_port>
-script_dir <script_dir> [-host <cluster>] [-out <IOR_file>]
```

Arguments

`add`	The number of node daemons to add to the host.
`base_port`	The port number to be used by the first new node daemon. Each additional node daemon will be assigned a port numbers incrementing upward by one.
`script_dir`	The directory where the domain's start and stop scripts reside. This is typically, *<install_dir>*\etc\bin.
`cluster`	Indicates the name of the cluster or federated name of which the host is associated. This parameter is optional.
`IOR_file`	The full path name of the file store the IORs of the new node daemons. This parameter is optional and the default location is *<current_working_dir>*\node_daemons.ior.

To add node daemons to a host:

1. Ensure that the domain to which additional node daemons are to be added is running.
2. Source the *<domain>*_env file to set the configuration environment variables.
3. Run the command. It silently configures and deploys the new node daemons into the running configuration. The domain start and stop scripts will be modified to include the new node daemons.
4. Once the command finishes, stop the domain's services using the domain's stop script, stop_*<domain>*_services.
5. Manually modify the value of initial_references:IT_NodeDaemon:reference for the CORBA servers you want to use the additional node daemons so that it contains a reference to the new node daemon.
6. If the servers are started on demand, you must also modify their process information to reflect the server's new node daemon.
7. Restart the domain using its start script, start_*<domain>*_services.

ORB Name

Overview

The following commands manage ORB names:

Table 23: *ORB Name Commands*

`orbname create`	Creates an ORB name in the location domain.
`orbname list`	Displays all ORB names in the location domain.
`orbname modify`	Modifies the specified ORB name entry either by associating it with another process entry, or by disassociating it from any process.
`orbname remove`	Removes an ORB name from the location domain.
`orbname show`	Displays attributes for the specified ORB name.

orbname create

Synopsis

`orbname create [-process` *`process-name`*`] ` *`ORB-name`*

Description

Creates the specified ORB name in the location domain. This designates a server-side ORB that is subject to POA or process activation. In the location domain, the ORB name is associated with a POA name and is used for process activation.

Arguments

`-process` Associates the ORB name with the specified process. The process name must previously be registered with the locator daemon (see "process create" on page 320).

Examples

The following command creates a scoped ORB name:

```
itadmin orbname create MutualFunds.Tracking.GroInc.Stocks
```

orbname list

Synopsis

```
orbname list [-active] [-count] [-process process-name]
```

Description

Lists all ORB names in the location domain.

Arguments

`-active`	Lists only the name in the locator's active ORB table.
`-count`	Lists the total number of ORB names in the location domain.
`-process`	Lists only the ORB name entries that are associated with *process-name*.

Examples

The following example lists all registered ORB names in the location domain:

```
itadmin orbname list
ifr
naming
production.test.testmgr
production.server
```

orbname modify

Synopsis

```
orbname modify [-process process-name] ORB-name
```

Description

Modifies the specified ORB name entry by associating it with the specified process name. If the process name is omitted, the ORB name is disassociated from any process.

Arguments

process-name	The name of the process to which the ORB name will be associated.

orbname remove

Synopsis

```
orbname remove [-active|-deep|-force] ORB-name
```

Description

Removes an ORB name from the location domain. You might need to remove an ORB name, if its application is removed from the environment, or if the ORB name has changed, or to prevent process activation.

If there is an active ORB entry for the ORB name in the locator's active ORB table, this is also removed.

An ORB name can be the same as the `ORB_id` (used to identify an ORB within a process) and has the following syntax:

```
ORBNameSegment.ORBNameSegment.ORBNameSegment
```

Arguments

The following arguments are mutually exclusive:

`-active`	Removes only the active ORB entry from the locator's active ORB table, and does not remove the ORB name.
`-deep`	Removes the ORB name and all POA names in the location domain that refer to it.
`-force`	Forces ORB name removal, even though some POA names in the location domain might have references to it.

Examples

The following example removes the `production.test` ORB name:

```
itadmin orbname list
ifr
naming
production.test.testmgr
production.server

itadmin orbname remove -active production.test.testmgr

itadmin orbname list
ifr
naming
production.server
```

orbname show

Synopsis

`orbname show` *`ORB-name`*

Description

Displays all the attributes for the specified ORB name.

Examples

The following example displays the attributes for the `company.sales` ORB name:

```
itadmin orbname show company.sales
ORB Name: company.sales
Process Name: sales_process
Active: yes
```

POA

Overview

The following commands manage POA entries:

Table 24: *POA Commands*

`poa create`	Creates a POA name in the location domain.
`poa list`	Displays POA names in the location domain.
`poa modify`	Modifies the indicated POA name as specified.
`poa remove`	Removes a POA name from the location domain.
`poa show`	Displays all data that is entered for *POA-name*.

poa create

Synopsis

```
poa create [-orbname ORB-name] [-replicas replica-list]
           [-persistent] [-transient] [-allowdynamic]
           [-allowdynreplicas] [-clear_replicas]
           [-load_balancer lb-name] FQPN
```

Registers a POA in the location domain. The required *FQPN* argument is the fully-qualified POA name. An *FQPN* has the following syntax:

```
FQPNsegment/FQPNsegment/FQPNsegment
```

Arguments

`-orbname` *ORB-name* Associates an ORB name with the specified POA. This argument requires an *ORB-name* argument with the following syntax:

`ORBNameSegment.ORBNameSegment.ORBNameSegment`

-orbname cannot be combined with `-persistent`, `-replicas`, or `-transient`

`-replicas` *replica-list*	Associates the specified POA with multiple ORBs specified in *replica-list*, where *replica-list* is a comma-delimited list of ORBs: *orb[,orb]...* `-replicas` cannot be combined with `-persistent`, `-orbname`, or `-transient`.
`-persistent`	Marks the POA as persistent without associating it with an ORB. `-persistent` cannot be combined with `-replicas`, `-orbname`, or `-transient`.
`-transient`	Marks the POA as transient. `-transient` cannot be combined with `-replicas`, `-orbname`, or `-persistent`
`-allowdynamic`	Enables dynamic registration of a POA in the location domain. The default is no dynamic registration. Enabling dynamic creation allows servers to register information (although administrators must create the top-level name manually).
`-allowdynreplicas`	Must be set to `yes` or `no`: • `yes`: (default) Any ORB creating the POA is automatically added to the POA's replica list. • `no`: Only those ORBs that are configured in the cluster through replicas are allowed to create the POA.
`-load_balancer` *lb-name*	Determines the load balancer used to select a replica response to client requests. If a load balancer is not specified, requests will be routed to the first server that creates the POA. The Orbix distribution provides support for the following algorithms: • `round_robin`: the locator uses a round-robin algorithm to select from the list of active servers—that is, the first client is sent to the first server, the second client to the second server, and so on. • `random`: the locator randomly selects an active server to handle the client.

Examples

The following command creates a transient POA name in the location domain:

```
itadmin poa create -transient banking_service
```

The following command creates a persistent POA name in the location domain:

```
itadmin poa create -orbname banking_services_app
   banking_service/account
```

The following command creates a persistent POA name associated with multiple ORBs:

```
itadmin poa create -replicas bank_server_1,bank_server_2
   -load_balancer round_robin banking_service/account
```

poa list

Synopsis

```
poa list [-active] [-children FQPN] [-count] [-persistent]
[-transient]
```

Description

Shows all POA names in the location domain.

Arguments

`-active`	Lists only entries for POAs that are currently active. `-active` and `-transient` parameters are mutually exclusive.
`-children` *FQPN*	Lists only entries for child POAs of the specified parent POA.
`-count`	Lists the total number of POA names in the location domain.
`-persistent`	Lists only POA names for persistent POAs.
`-transient`	Lists only POA names for transient POAs. `-transient` and `-active` arguments are mutually exclusive.

Examples

```
itadmin poa list
banking_service
banking_service/account
banking_service/account/checking
banking_service/account/checking/deposit
```

poa modify

Synopsis

```
poa modify [-allowdynamic] [-allowdynreplicas]
           [-orbname ORB-name]
           [-replicas replica-list]
           [-clear_replicas]
           [-load_balancer lb-name] FQPN
```

Description

Modifies the specified POA name. The required *FQPN* argument is the fully-qualified POA name. A *FQPN* has the following syntax:

```
FQPNsegment/FQPNsegment/FQPNsegment
```

Arguments

`-allowdynamic` Enables dynamic registration of a POA in the location domain. The default is no dynamic registration. Enabling dynamic creation allows servers to register information (although administrators must create the top-level name manually).

`-allowdynreplicas` Must be set to `yes` or `no`:

- `yes`: (default) Any ORB creating the POA is automatically added to the POA's replica list.
- `no`: Only those ORBs that are explicitly configured in the cluster through replicas are allowed to create the POA.

`-orbname` *ORB-name* Associates the specified ORB name with the specified POA. This argument requires an *ORB-name* argument with the following syntax:

```
ORBNameSegment.ORBNameSegment.ORBNameSegment
```

`-replicas` *`replica-list`*	Associates the specified POA with multiple ORBs specified in *`replica-list`*, where *`replica-list`* is a comma-delimited list of ORBs: *`orb[,orb]...`* `-replicas` cannot be combined with `-orbname`.
`-clear_replicas`	Disassociates the POA from any ORBs.
`-load_balancer`	Determines the load balancer used to select a replica response to client requests. If a load balancer is not specified, requests will be routed to the first server that creates the POA. The Orbix distribution provides support for the following algorithms: • `round_robin`: the locator uses a round-robin algorithm to select from the list of active servers—that is, the first client is sent to the first server, the second client to the second server, and so on. • `random`:: the locator randomly selects an active server to handle the client.

poa remove

Synopsis

```
poa remove [-active|-allactive] FQPN
```

Description

Removes the entry for the specified POA and its descendants from the location domain. By default, all active entries for the POA and its children are also removed. Use the `-active` argument to remove only the active entry for the specified POA.

Arguments

`-active`	Removes currently active entries for the specified POA only. `-active` and `-allactive` arguments are mutually exclusive.
`-allactive`	Removes only active entries for the specified POA and all its children.

Examples

The following example removes the specified POA and its children:

```
itadmin
% poa list
banking_service
banking_service/account
banking_service/account/checking
banking_service/account/checking/deposit

% poa remove banking_service/account/checking
% poa list
banking_service
banking_service/account
```

poa show

Synopsis

```
poa show FQPN
```

Description

Displays all the attributes for the specified POA name. A *FQPN* (fully-qualified POA name) has the following syntax:

```
FQPNsegment/FQPNsegment/FQPNsegment
```

Examples

The following example shows the attributes for the `IFR` POA name:

```
itadmin poa show IFR
FQPN: IFR
  Active: no
  Lifespan: persistent
  ORB Names:
    iona_services.ifr
  Allow Replicas outside this list: no
  Load Balancing Algorithm:  <NONE>
  Allow Dynamic Registration: no
  Parent FQPN: <NONE>
  Children FQPN: <NONE>
```

Server Process

Overview

The following commands let you manage server process entries:

Table 25: *Server Process Commands*

process create	Creates a server process name in the location domain.
process disable	Disables the specified server process for process activation, using the node daemon.
process enable	Enables a target server process for on-demand activation by the node daemon.
process kill	Kills the specified process that was started by its associated node daemon.
process list	Lists names of server processes in the location domain.
process modify	Modifies the process as specified.
process remove	Removes a server process name from the location domain.
process show	Displays a complete server process entry.
process start	Starts a registered server process.
process stop	Stops a registered server process.

process create

Synopsis

```
process create -args "-ORBname orb-name [arg-list]"
                [-description] [-startupmode mode]
              [-node_daemon node-daemon-name] [-pathname pathname]
            [-directory dir] [-env env] [-group group] [-user user]
               [-umask umask] process-name
```

Description

Registers a server process in a location domain's implementation's repository.

Arguments

The following arguments apply to all platforms.

`-args`

Arguments supplied to the process when it starts. At a minimum, supply the `-ORBname` argument with the name of the ORB associated with this server process.

Enclose all arguments within quotation marks, and separate multiple arguments with spaces. For example:

```
itadmin process create -args "-ORBname
   company.production.sver1" my_app
```

If you are registering a Java server, the argument list generally includes the class path.

If the process start-up mode is `per_client`, the locator creates a new ORB name and a new process entry for each request from a client to the persistent POA associated with this process. In this case, the `%o` and `%p` strings in the process's arguments are substituted with the new ORB name and the new process name respectively. For example:

```
-args "--ORBname %o"
```

For more details, see "Per-client activation" on page 54.

`-description`

A brief description of the target process. Enclose the description in double quotes.

`-startupmode`

Specifies the start-up mode of the target process:

- `on_demand` (default) starts the process when requested by a client.
- `per_client` starts a new process for each client.
- `disable` disables automatic startup.

`-node_daemon`

The name of the node daemon that starts or modifies this process.

`-pathname`

The full pathname of the executable to start when the process is activated.

On Windows platforms, specify a drive letter if not the current drive of the node daemon. Windows paths can be expressed with one forward slash separator or two backward slashes.

`-directory` Specifies the working directory to which the target process writes output files, error logs, and so on.

On UNIX the default current working directory is set to the root file system. On Windows, the default current drive is the node daemon's drive, and the current directory is set to the root directory.

On Windows, specify a drive letter if the working directory drive differs from the node daemon's current drive. Windows paths can be expressed with one forward slash separator or two backward slashes.

On UNIX, if the current working directory path does not exist, it is created automatically with permissions `drwx------`.

Use this argument in order to:

- Ensure that the server runs in a directory that is in the root file system. This avoids problems with running servers in mounted file systems.
- Use relative path names. This means that administrators can set the working directory for the activated server, without having to define other paths and directories.
- Ensure that core files cannot overwrite each other if the server is configured to run somewhere other than the root directory.

`-env` Explicitly sets the process environment. This argument takes an list of space-delimited *`variable=value`* pairs, enclosed in quotation marks:

```
env "DISPLAY=circus:0.0 CLOWN=Bozo HOME=/tent"
```

This option overrides any environment variables set by the node daemon. By default, the server inherits its environment from the node daemon. If you use this option, you must specify all environment variables that the server requires.

For more information about environment settings, see "Server Environment Settings" on page 56.

`-group` Group name that starts the target process. The default is nobody. For more information, see page 58.

`-user`	User name that starts the target process. The default is nobody. For more information, see page 58.
`-umask`	File mode creation mask for the activated target process. Specify as three octal digits ranging from `000` to `777`. The default is `022` (maximum file permissions: `755`, or `rwxr-xr-x`).

process disable

Synopsis

`process disable` *`process-name`*

Description

Disables on-demand activation of the specified server *`process-name`*.

process enable

Synopsis

`process enable` *`process-name`*

Description

Enables on-demand activation of the specified server *`process-name`*.

process kill

Synopsis

`process kill [-signal` *`signal_number`*`] [-force]` *`process_name`*

Description

Kills the specified process that was started by its associated node daemon. The `-signal` argument specifies the UNIX signal number to kill the process. This command has the following effects:

UNIX	Sends a signal to the process. The default is `9`.
Windows	Calls `TerminateProcess()`.

This command only works for processes activated by the node daemon. For manually launched processes, it has no effect.

Arguments

`-signal`	Specifies the UNIX signal number to kill a process. The default is `9`.

`-force` Forces the removal of the persistent data for the specified process from the implementation repository (IMR). This can be used when a previously active process has died or been killed, and the persistent data in the IMR was not cleaned up correctly. If the persistent data held by the locator and node daemon was not correctly cleaned up, there may be issues when trying to restart the process.

Note: This command should be used with caution, and only if the normal cleanup mechanisms have failed for some unknown reason.

process list

Synopsis

```
process list [-count] [-node_daemon node-daemon-name] [-active]
```

Description

Lists the target process names of all processes registered in the location domain. Listing process names is useful for verifying a target process name or its status.

Arguments

`-count` Displays the total number of process names in the location domain.

`-node_daemon` Lists all monitored processes for a given node daemon. This is useful if you want to perform the `node_daemon stop` command.

`-active` Lists all currently active processes.

Examples

The following example lists all registered process names in a location domain

```
itadmin process list
if
naming
my_app
```

process modify

Synopsis

```
process modify -args '-ORBname orb-name [arg-list]"
               [-description] [-startupmode mode]
               [-node_daemon node-daemon-name]
               [-pathname pathname] [-directory dir]
               [-env env] [-group group] [-user user]
               [-umask umask] process-name
```

Description

Modifies the specified process entry in the implementation repository.

Arguments

`-args` Arguments supplied to the process when it starts. At a minimum, supply the `-ORBname` argument with the name of the ORB associated with this server process.

Enclose all arguments with quotation marks, and separate multiple arguments with spaces. For example:

```
itadmin process create -args "-ORBname
   company.production.sver1" my_app
```

If you are registering a Java server, the argument list generally includes the class path.

If the start-up mode of the process is `per_client`, the locator creates a new ORB name and a new process entry for each request from a client to the persistent POA associated with this process. In this case, the `%o` and `%p` strings in the process's arguments are substituted with the new ORB name and the new process name respectively. For example:

```
-args "--ORBname %o"
```

For more details, see "Per-client activation" on page 54.

`-description` A brief description of the target process.

`-startupmode` Specifies the start-up mode of the target process:

- `on_demand` starts the process when requested by a client.
- `per_client` starts a new process for each client.
- `disable` disables automatic startup.

`-node_daemon` The name of the node daemon that will start or modify this process.

`-pathname` The complete pathname of the executable that will be started when the process is activated.

For Windows platforms, specify a drive letter if the executable is not the same as the current drive of the node daemon. Windows paths can be expressed with one forward slash separator or two backward slashes.

`-directory` Specifies the working directory where the target process writes output files, error logs, and so on.

On UNIX the default current working directory is set to the root file system. On Windows, the default current drive is the node daemon's drive, and the current directory is set to the root directory.

On Windows, specify a drive letter if the working directory drive differs from the node daemon's current drive. Windows paths can be expressed with one forward slash separator or two backward slashes.

On UNIX, if the current working directory path does not exist, it is created automatically with permissions `drwx------`.

Use this argument in order to:

- Ensure that the server runs in a directory that is in the root file system. This avoids problems with running servers in mounted file systems.
- Use relative path names. This means that administrators can set the working directory for the activated server without having to define other paths and directories.
- Ensure that core files cannot overwrite each other if the server is configured to run somewhere other than the root directory.

`-env` Explicitly sets the process environment. This argument takes a list of space-delimited *`variable=value`* pairs, enclosed in quotation marks:

```
env "DISPLAY=circus:0.0 CLOWN=Bozo HOME=/tent"
```

This option overrides any environment variables set by the node daemon. By default, the server inherits its environment from the node daemon. If you use this option, you must specify all environment variables that the server requires.

For more information about environment settings, see "Server Environment Settings" on page 56.

`-group` Group name that starts the target process. The default is nobody. For more information, see page 58.

`-user` User name that starts the target process. The default is nobody. For more information, see "File access permissions" on page 58.

`-umask` File mode creation mask for the activated target process. Specify as three octal digits, ranging from `000` to `777`. The default is `022` (maximum file permissions: `755`, or `rwxr-xr-x`).

---

process remove

Synopsis

```
process remove [-force|-deep|-active] process-name
```

Description

Removes a process implementation repository entry created using `process create`. If you omit the `-force` or `-deep` switch, POA entries that reference this process are not removed and an error is reported.

Removing a process also removes the active process entry from the locator's active process table. The `-active` argument removes only an active process entry from the locator's active process table; the process remains registered with the implementation repository.

Arguments

The following arguments are mutually exclusive. Choose one:

`-active` Removes only the active process entry from the locator's active process table.

`-deep` Removes the process entry and all object adapter implementation repository entries that refer to it.

`-force` Forces process removal even if other implementation repository entities have references to it.

Examples

The following example removes the `my_app` server process name:

```
itadmin process list
ifr
naming
my_app

itadmin process remove -force my_app

itadmin process list
ifr
naming
```

process show

Synopsis

`process show` *`process-name`*

Description

Displays all process data entered for the specified *`process-name`*. If the process is active, `process show` displays the active node daemon name. Viewing a target process is useful for verifying whether a process name is registered and has the appropriate settings.

Examples

The following example shows the information registered with the locator daemon for a target process:

```
itadmin process show my_app
Process Name: my_app
Description: Unknown services provided.
Startup Mode: on_demand
Node Daemon List:
  Node Daemon Name: oregon
    Host Name: oregon
    Max. Retries: 3
    Retry Interval: 2
    Path Name: c:\Program Files\Acme\bin\my_app.exe
    Arguments: -safe -sane
    Environment Variables: Inherited from node daemon
    File Access Permissions:
      User:  mstephen
      Group: PC-GROUP
    File Creation Permissions:
      Umask: 022
    Current Directory: /
    Resource Limits: Inherited from node daemon
```

process start

Synopsis

`process start` *`process-name`*

Description

Starts a target process on the host where the node daemon configured for the process resides.

process stop

Synopsis

```
process stop [-signal number] process-name
```

Arguments

Stops the specified process that was started by its associated node daemon. Depending on the environment used, this command has the following effect:

UNIX/C++	Sends a `SIGINT (2)` signal to the process.
Windows/C++	Calls `GenerateConsoleCtrlEvent(CTRL_BREAK_EVENT, 0)`.
Java	Calls `System.exit(0)`.

Arguments

`-signal`	Specifies the UNIX signal number to stop a process.

WARNING: The signal number is ignored for a Windows NT process.

CHAPTER 24

Mainframe Adapter

Overview

The following `itadmin` commands enable you to use the mapping gateway interface of the Orbix Mainframe Adapter (MFA).

These commands enable you to list transaction mappings supported by your CICS or IMS server adapter, add or delete interfaces and operations, and change transactions that operations are mapped to. A new mapping file can be read, or the existing mappings can be written to a new file.

Table 26: *Mainframe Adapter itadmin Commands*

`mfa add`	Adds a new mapping.
`mfa change`	Changes the transaction to which an existing operation is mapped.
`mfa delete`	Causes the server adapter to stop exporting a specified operation.
`mfa -help`	Prints a list of the operations that the `mfa plugin` supports.
`mfa list`	Prints a list of the mappings (interface, operation, and name) that the server adapter supports.

`mfa list`	Prints a list of the mappings (interface, operation, and name) that the server adapter supports.

mfa add

Synopsis

```
mfa add -interface <name> -operation <name> <mapped value>
```

Description

Adds a new mapping.

Parameters

You must supply the name of the interface, name of the operation and the mapped value that you want added. Module names form part of the interface name and are separated from the interface name with a `/` character.

Examples

For example, to add a new `Simple/SimpleObject` mapping, use the following command:

```
itadmin mfa add -interface Simple/SimpleObject -operation
   call_me SIMPLESV
```

mfa change

Synopsis

```
mfa change -interface <name> -operation <name> <mapped value>
```

Description

Changes the transaction to which an existing operation is mapped.

Parameters

You must supply the name of the interface, name of the operation and the mapped value that you want added. Module names form part of the interface name and are separated from the interface name with a `/` character.

Examples

For example, to change the transaction to which the `call_me` operation is mapped to `SIMPLESV`, use the following command:

```
itadmin mfa change -interface Simple/SimpleObject -operation
   call_me SIMPLESV
```

mfa delete

Synopsis

```
mfa delete -interface <name> -operation <name>
```

Description

Stops the server adapter exporting the specified operation.

Parameters

You must supply the interface name and the operation name that you want the server adapter to stop exporting. Module names form part of the interface name and are separated from the interface name with a `/` character.

Examples

For example, to stop the server adapter exporting the `call_me` operation, use the following command:

```
itadmin mfa delete -interface Simple/SimpleObject -operation
   call_me
```

mfa -help

Synopsis

```
mfa -help
```

Description

Lists all the operations provided by the `mfa itadmin` plugin.

mfa list

Synopsis

```
mfa list
```

Description

Prints a list of the mappings (interface, operation and name) that the adapter server supports.

Parameters

You must supply the interface name. Module names form part of the interface name and are separated from the interface name with a `/` character.

mfa refresh

Synopsis

```
mfa refresh [-operation <name>] <interface name>
```

Description

Causes the server adapter to obtain up-to-date type information for the specified interface.

Parameters

You must supply the interface name. Module names form part of the interface name and are separated from the interface name with a `/` character. The `-operation <name>` argument is optional. If you omit the `-operation <name>` argument, all operations mapped in the specified interface are refreshed.

Examples

For example, to cause the server adapter to get up-to-date type information for the `Simple` interface, use the following command:

```
itadmin mfa refresh Simple/SimpleObject
```

mfa reload

Synopsis

```
mfa reload
```

Description

Causes the server adapter to reload the list of mappings from its mapping file.

mfa resetcon

Synopsis

```
mfa resetcon
```

Description

If the IMS server adapter is using OTMA to communicate with IMS, when this operation is called on the Mapping Gateway interface, the server adapter closes its connection with OTMA and reconnects. This is done in such a way that it does not affect any clients connected to the server adapter by briefly queueing client requests in the server adapter until the connection is re-established. The purpose of this operation is to free any cached security ACEE's on the OTMA connection. You should, therefore, use this operation after changes that affect users access to IMS have been made to user security profiles in the z/OS security package; for example, RACF.

Note: This command has no effect on the CICS server adapter.

mfa resolve

Synopsis

```
mfa resolve <interface name>
```

Description

Prints a stringified IOR for the object in the server adapter that supports the specified interface. This IOR string can then be given to clients of that interface, or stored in an Orbix naming service. The IOR produced contains the TCP/IP port number for the locator if the server adapter is running with direct persistence set to `no`. Otherwise, it contains the server adapter's port number.

Examples

For example, to retrieve an IOR for `Simple` IDL, use the following command:

```
itadmin mfa resolve Simple/SimpleObject
```

Once retrieved, the IOR can be distributed to the client and used to invoke on the target server running inside CICS or IMS.

mfa save

Synopsis

```
mfa save [<mapping_file name>]
```

Description

Causes the server adapter to save its current mappings to either its current mapping file, or to a file name that you provide.

Parameters

The `[<mapping_file name>]` argument is optional. You need only provide it if you want the server adapter to save its current mappings to a specified file.

Examples

For example, to get the server adapter to save its current mappings to a `myMappings.map` file, use the following command:

```
itadmin mfa save "C:\myMappings.map"
```

mfa stats

Synopsis

```
mfa stats
```

Description

Displays some statistical information on the running server adapter. Information includes the current time according to the server adapter, the pending request queue length, the total number of worker threads, worker threads currently active, total number of requests processed by the server adapter since startup and the server adapter startup time.

mfa stop

Synopsis

```
mfa stop
```

Description

Causes the server adapter to shut down.

mfa switch

Synopsis

```
mfa switch <mapping_file name>
```

Description

Causes the server adapter to switch over to a new mapping file, and to export only the mappings contained in it.

Parameters

You must provide the name of the mapping file that you want the server adapter to switch over to.

Examples

For example, to get the server adapter to switch over to a `myMappings.map` mapping file, use the following command:

```
itadmin mfa switch "c:\myMappings.map"
```

CHAPTER 25

Naming Service

Overview

A subset of `itadmin` commands let you manage the naming service and its contents. You can use these commands to create, list, and remove naming contexts, objects, and object groups from the naming service.

All paths and compound names in the naming service conform to the CORBA Interoperable Naming Service (INS) string name format.

Naming service commands operate on two components:

Names

Overview

The following `ns` commands let you manage and browse the naming service:

Table 27: *Naming Service Commands*

`ns bind`	Creates an association between a context or object reference and the specified compound name.
`ns list`	Lists the contents of the specified path.
`ns list_servers`	Lists all active naming servers.
`ns newnc`	Creates a new naming context or object and binds it to the specified path.
`ns remove`	Removes the specified context or object.
`ns resolve`	Displays a resolved string name form of the IOR for a specified path.
`ns show_server`	Displays the naming server details for the server name specified.
`ns stop`	Stops the naming service.
`ns unbind`	Unbinds the path-specified context or object.

ns bind

Synopsis

```
ns bind {-context | -object} -path path  IOR
```

Description

Creates an association between a context or object reference and the *path*-specified compound name. Use this command in command-line mode only.

Arguments

`-context` Binds a context

`-object` Binds an object.

`-path` Specifies an INS string name as the path to the new binding.

Examples

The following example binds an object to the name `james.person`,in the `company/staff` naming context:

```
itadmin ns bind -o -path company/staff/james.person
   "IOR:0000000037e276f47a4b94874c64648e949..."
```

---

ns list

Synopsis

`ns list [`*`path`*`]`

Description

Displays the contents of the specified path. If *`path`* resolves to a context, its contents are displayed. If *`path`* resolves to an object, the object is displayed. If no path is specified, the contents of the initial naming context are displayed. The *`path`* argument takes the form of an INS string name.

The type of the binding is also listed. A binding of type `Object` names an object. A binding of type `Context` names a naming context.

Examples

The following command lists the bindings in `company/engineering` in the naming service:

```
itadmin ns list company/engineering
paula (Object)
production (Context)
john (Object)
manager (Object)
```

---

ns list_servers

Synopsis

`ns list_servers [-active]`

Description

Lists all the active servers.

Arguments

`-active` Displays all active naming servers.

ns newnc

Synopsis

```
ns newnc [path]
```

Description

Creates a naming context or object and binds it to the specified path. If *path* is not specified, `ns newnc` prints the IOR to standard out. The *path* argument takes the form of an INS string name.

Examples

```
itadmin
% ns newnc foo.bar/foo3.bar3
% ns list foo.bar
/foo2.bar2      Context
/foo3.bar3      Context
```

ns remove

Synopsis

```
ns remove [-recursive] path
```

Description

Unbinds the specified context or object. If *path* is a context, the context is also destroyed. The `ns remove` command checks whether a context is empty before destroying it. If the context is empty, `ns remove` destroys it and then unbinds it. If the context is not empty and you omit the `-recursive` argument, `ns remove` displays an error message. The required *path* argument specifies an INS string name.

Arguments

`-recursive` Recursively destroys and unbinds a context or object if the context is not empty.

Examples

For example, the following commands destroy the `manager` bindings:

```
itadmin ns remove company/engineering/manager.person
itadmin ns remove company/engineering/support/manager.person
```

ns resolve

Synopsis

```
ns resolve path
```

Description

Prints the resolved string form of the IOR for a given path specified by an INS string name. If a path is not specified, the string form of the root naming context is displayed. The *path* argument takes the form of an INS string name.

For example:

```
itadmin ns resolve company/engineering
"IOR:0003032272d9218a35d9614357f87c93800d7...6f3"
```

Examples

The following examples show that the names `company/staff/paula.person` and `company/engineering/manager.person` resolve to the same object:

```
itadmin ns resolve company/staff/paula.person
"IOR:00000000569a2e8034b94874d6583f09e24..."

itadmin ns resolve company/engineering/manager.person
"IOR:00000000569a2e8034b94874d6583f09e24..."
```

---

ns show_server

Synopsis

`ns show_server` *`server_name`*

Description

Displays the naming server details for the server name specified.

---

ns stop

Synopsis

`ns stop` *`server_name`*

Description

Stops the naming service.

---

ns unbind

Synopsis

`ns unbind` *`path`*

Description

Unbinds the context or object specified by *`path`*. The *`path`* argument takes the form of an INS string name.

Object Groups

Overview

The following `nsog` commands let you manage object groups:

Table 28: *Object Group Commands*

`nsog add_member`	Adds the specified member object to the specified object group.
`nsog bind`	Binds the specified object group to the specified path.
`nsog create`	Creates the specified object group, with the specified selection policy.
`nsog list`	Lists all object groups currently existing in the naming service.
`nsog list_members`	Lists the names of members belonging to the specified object group.
`nsog modify`	Modifies the selection policy for the specified object group.
`nsog remove`	Removes the specified object group from the naming service.
`nsog remove_member`	Removes the specified member object from the specified object group.
`nsog set_member_timeout`	Sets the load timeout period for a member of an active object group.
`nsog show_member`	Displays the object reference that corresponds to the specified member of an object group.
`nsog update_member_load`	Updates the load value of a member of an active object group.

nsog add_member

Synopsis

`nsog add_member -og_name` *`group-name`* `-member_name` *`member-name IOR`*

Description

Adds an object to the specified object group. After being added, the object is available for selection.

Arguments

The following arguments are all required:

`-og_name` *`group-name`*	Specifies the object group to which the member is added.
`-member_name` *`member-name`*	Specifies a unique group member name.
`IOR`	Specifies the member's object reference.

Examples

The following command adds a member, `paula`, to the `engineers` object group with an object reference of `IOR:0001def...`:

```
itadmin nsog add_member -og_name engineers -member_name paula
    IOR:0001def...
```

nsog bind

Synopsis

`nsog bind -og_name` *`group-name path`*

Description

Binds the specified object group to the specified path in the naming service. When clients resolve that path, they transparently obtain a member of the specified object group.

Arguments

`-og_name` *`group-name`*	Specifies the name of the object group to bind.
`path`	SPecifies the INS path to bind the object group.

Examples

The following example binds the `engineers` object group to the path `company/engineering/engineers.pool`:

```
itadmin nsog bind -og_name engineers
    company/engineering/engineers.pool
```

The `company/engineering` context must be already created.

nsog create

Synopsis

```
nsog create -type selection-policy group-name
```

Description

Adds the named object group *group-name* to the naming service with the specified selection policy. On creation, an object group contains no member objects.

The naming service directs client requests to object group members according to the specified selection algorithm. For more about active load balancing, see "Active load balancing" on page 120.

Arguments

`-type` *selection-policy*	Specifies the object group's selection algorithm with one of the following values:
	`rr`: round-robin
	`rand`: random
	`active`: active load balancing
group-name	Specifies the name of the new object group.

Examples

The following example creates an object group, `engineers`, with a random selection policy:

```
itadmin nsog create -type rand engineers
```

nsog list

Synopsis

```
nsog list
```

Description

Displays all object groups that currently exist in the naming service.

Examples

```
itadmin nsog list
Random Groups:   engineers
```

nsog list_members

Synopsis

```
nsog list_members -og_name group-name
```

Description Lists the members of the specified object group.

Arguments

`-og_name` *group-name*	Specifies the target object group.

Examples The following example lists the members of the `engineers` object group:

```
itadmin nsog list_members engineers
```

nsog modify

Synopsis `nsog modify -type` *selection-policy group-name*

Description Changes the selection algorithm for the specified object group. An object group's selection algorithm determines how the naming service directs client requests to object group members (see "Selection algorithms" on page 119).

Arguments

`-type` *selection-policy*	Specifies the object group's selection algorithm with one of the following values: `rr`: round-robin `rand`: random `active`: active load balancing (see "Active load balancing" on page 120).
group-name	Specifies the object group to modify.

Examples The following command changes the object group `engineers`'s selection algorithm:

```
itadmin nsog modify -type rr engineers
```

nsog remove

Synopsis `nsog remove` *group-name*

Description Removes the specified object group from the naming service.

Examples

The following example removes and unbinds the `engineers` object group:

```
itadmin nsog remove engineers
itadmin unbind company/engineering/engineers.pool
```

Note: If the object group is bound in a naming graph, you must also unbind it, as shown in this previous example.

nsog remove_member

Synopsis

`nsog remove_member -og_name` *`group-name member-name`*

Description

Removes an object group member. You might wish to remove a member of an object group if it no longer participates in the group—for example, the service it references is inaccessible.

Arguments

`-og_name` *`group-name`*	The target object group.
`member-name`	The member to remove from *`group-name`*.

Examples

The following example removes `paula` from the `engineers` object group:

```
itadmin nsog remove_member -og_name engineers paula
```

nsog set_member_timeout

Synopsis

`nsog set_member_timeout -og_name` *`group-name`* `-member_name` *`member timeout-value`*

Description

Specifies how long an object group member is eligible for load updates, in an object group that has active load balancing. If the member's load value is not updated before *`timeout-value`* elapses, the member is removed from the object group's selection pool.

This command has no effect on round-robin and random groups. However, the member timeout is stored and put to use if the object group's selection algorithm is modified to active load balancing (see "nsog modify" on page 347).

Arguments

`-og_name` *`group_name`*	Specifies the target object group.
`-member_name` *`member`*	Specifies the target object.
`timeout-value`	Specifies the timeout value in seconds. A value of `-1` sets an infinite timeout value.

Examples

The following command sets the load timeout period to `30` seconds for member `gate3` in the `gateway` active object group:

```
nsog set_member_timeout -og_name gateway -member_name gate3 30
```

---

nsog show_member

Synopsis

`nsog show_member -og_name` *`group-name member-name`*

Description

Displays the object reference that corresponds to the specified member of the specified object group.

Examples

For example, to display the IOR of member `paula` in the object group `engineers`:

```
itadmin nsog show_member -og_name engineers paula
"IOR:00000000569a2e8034b94874d6583f09e24..."
```

nsog update_member_load

Synopsis

```
nsog update_member_load -og_name group_name -member_name
member_name load_value
```

Description

Updates the load value for the specified member of an active object group. This load value is valid for a period of time specified by the timeout assigned to that member (see "nsog set_member_timeout" on page 348). In an active selection policy, the naming service selects the group member with the lowest load value.

This command has no effect on round-robin and random object groups. The naming service makes no interpretation of a member's load value, and only uses this information to select the lowest loaded member.

Examples

The following command updates the load value to `2.0` for `member1` in the `webrouter` active object group:

```
nsog update_member_load -og_name webrouter -member_name member1
   2.0
```

CHAPTER 26

Notification Service

Overview

The CORBA notification service enables applications to send events to any number of objects. For more details, see the *Orbix Enterprise Messaging Guide*.

Orbix `itadmin` commands enable you to manage the following components of a notification service:

Notification Service Management

The following commands let you manage an notification service instance.

Table 29: *Notification Service Commands*

`notify checkpoint`	Performs checkpoint operations on the notification service's Berkeley DB database.
`notify post_backup`	Performs post-backup operations on the notification service database.
`notify pre_backup`	Performs pre-backup operations on the notification service database.
`notify show`	Displays the attributes of the specified notification service.
`notify stop`	Stops a notification service.

notify checkpoint

Synopsis

```
notify checkpoint
```

Description

Performs checkpoint operations on the notification service's Berkeley DB database.

When using transactions, Berkeley DB maintains transaction log files. Each time a transaction commits, data is appended to the transaction log files, and the database files are not modified. Data in transaction log files is then transferred periodically to the database files. This transfer is called a *checkpoint*. You can specify the checkpoint interval with the following configuration variable:

```
plugins:notify:database:checkpoint_interval
```

The checkpoint operation performs a Berkeley DB checkpoint. The following configuration variable determines whether to delete the old log files, or move them to another directory:

```
plugins:notify:database:checkpoint_deletes_old_logs
```

The following configuration variable specifies the directory to which log files should be moved:

```
plugins:notify:database:old_log_dir
```

notify post_backup

Synopsis

```
notify post_backup
```

Description

Performs post-backup operations on the notification service database.

When backing up data files, it is important that no checkpoint occurs during the backup. The pre-backup operations force a checkpoint and then suspend checkpointing. The post-backup operations resume checkpointing.

notify pre_backup

Synopsis

```
notify pre_backup
```

Description

Performs pre-backup operations on the notification service database.

When backing up data files, it is important that no checkpoint occurs during the backup. The pre-backup operations force a checkpoint and then suspend checkpointing. The post-backup operations resume checkpointing.

notify show

Synopsis

```
notify show
```

Description

Displays the attributes of the default notification service.

Multiple instances of the notification service are also supported. To show the attributes of a non-default notification service, specify the ORB name used to start the notification service (using the `-ORBname` parameter to `itadmin`).

Examples

The following command shows the attributes of a default notification service:

```
itadmin notify show
Notification Service Name: IT_NotifyNamedRoot
  Host Name: podge
  Notification Channel Name List:
    my_channel
```

The following command shows the attributes of the specified non-default notification service:

```
itadmin -ORBname notify.notify2 notify show
Notification Service Name: IT_NotifyNamedRoot2
  Host Name: rodge
  Notification Channel Name List:
    my_channel
    my_channel2
```

The notification service name must be unique for each notification service instance. You can specify this is in your configuration, by setting `plugins:poa:root_name`. The notification service uses named roots to support multiple instances.

In the following example, `plugins:poa:root_name` is set to `IT_NotifyNamedRoot2` in the `notify.notify2` configuration scope:

```
...
event{
    plugins:poa:root_name = "IT_NotifyNamedRoot";
    ...

    notify2
    {
     plugins:poa:root_name = "IT_NotifyNamedRoot2";
    };
}
...
```

notify stop

Synopsis

```
notify stop
```

Description

Stops the default notification service.

Multiple instances of the notification service are also supported. To stop a non-default notification service, specify the ORB name used to start the notification service (using the `-ORBname` parameter to `itadmin`).

To start the notification service, use the `itnotify run` command. You can also use the `start_`*`domain-name`*`_services` command. For more information, see "Starting Orbix Services" on page 239.

Examples

The following command stops the default notification service:

```
itadmin notify stop
```

The following command stops a notification service that was started with an ORB name of `notify.notify2`:

```
itadmin -ORBname notify.notify2 notify stop
```

Event Channel

The following commands enable you to manage a notification service's event channel:

Table 30: *Event Channel Commands*

`nc create`	Creates an untyped event channel with the specified name.
`nc list`	Displays all untyped event channels managed by the notification service.
`nc remove`	Removes the specified untyped event channel.
`nc show`	Displays all attributes of the specified untyped event channel.
`nc set_qos`	Specifies qualities of service for the specified event channel.

nc create

Synopsis

`nc create -event_reliability -connection_reliability` *`channel-name`*

Creates an untyped event channel, in the default notification service, with the specified name.

Arguments

`-event_reliability`	Specifies the level of guarantee given on the delivery of individual events. Possible values are `best_effort` or `persistent`.
`-connection_reliability`	Specifies the level of guarantee given on the persistence of a clients connection to its notification channel. Possible values are `best_effort` or `persistent`.

Examples

The following command creates an untyped event channel named `my_channel`:

```
itadmin nc create -event_reliability persistent
   -connection_reliability persistent my_channel
```

The following command creates an untyped event channel named `my_channel2` in the `notify.notify2` notification service:

```
itadmin -ORBname notify.notify2 nc create -event_reliability
   persistent -connection_reliability persistent my_channel2
```

The event reliability and connection reliability must be set at the time of creation. When these values are set, they cannot be changed.

---

nc list

Synopsis

```
nc list -count
```

Description

Displays all the untyped event channels managed by the notification service.

To display the total number of untyped event channels, specify the `-count` argument. No value argument is required.

Examples

The following command displays the untyped event channels managed by a default notification service:

```
itadmin nc list
my_channel
mkt_channel
eng_channel
```

The following command displays the untyped event channels managed by a non-default notification service:

```
itadmin -ORBname notify.notify2 nc list
my_channel
my_channel2
mkt_channel
eng_channel
```

The following command displays the number of untyped event channels managed by a notification service:

```
itadmin nc list -count
3
```

nc remove

Synopsis

`nc remove` *`channel-name`*

Description

Removes the specified untyped event channel.

Examples

The following command removes an untyped event channel named `my_channel`:

```
itadmin nc remove my_channel
```

The following command removes an untyped event channel (from a non-default notification service) named `my_channel2`:

```
itadmin -ORBname notify.notify2 nc remove my_channel2
```

nc show

Synopsis

`nc show` *`channel-name`*

Description

Displays all attributes of the specified untyped event channel.

Examples

The following command displays all the attributes of an event channel named `my_channel`:

```
itadmin nc show my_channel
Channel Name: my_channel
  Channel ID: 1
  Event Communication: Untyped
```

The following command displays the attributes of an event channel (from a non-default notification service) named `my_channel2`:

```
itadmin -ORBname notify.notify2 nc show my_channel2
Channel Name: my_channel2
  Channel ID: 2
  Event Communication: Untyped
```

Note: For information about notification service configuration variables, see the section discussing the `plugins:notification` namespace in the *Orbix Configuration Reference*.

---

nc set_qos

Synopsis

```
nc set_qos
[-priority] [-order_policy] [-discard_policy]
[-start_time_supported] [-stop_time_supported]
[-max_events_per_consumer] [-max_batch_size] [-max_retries]
[-pacing_interval] [-timeout] [-pull_interval] [-retry_timeout]
[-max_retry_timeout] [-request_timeout] [-retry_multiplier]
channel name
```

Specifies various qualities of service (QoS) for the specified event channel name. Values of existing QoS properties can be changed, and new QoS properties can be added. All `set_qos` arguments are optional.

Arguments

-priority	Specifies the order that events are delivered to a consumer whose -order_policy is set to priority_order. It also affects the order that events are dequeued for consumers whose -discard_policy is set to priority_order. The -priority indicates the relative priority of the event compared to other events in the channel. Values can be in the range of -32,767 and 32,767. Higher priority events are delivered before lower. The default is 0.
-order_policy	Specifies the order to queue events for delivery. Possible values are: any_order fifo_order priority_order deadline_order
-discard_policy	Specifies the order that events are discarded when -max_events_per_consumer has been reached. Possible values are: any_order fifo_order priority_order deadline_order
-start_time_supported	Specifies whether start time is supported. This is an absolute time (e.g., 20/12/04 at 11:15) that determines the earliest time a channel can deliver the event. If set to true, the event is held until the specified time is reached.
-stop_time_supported	Specifies whether stop time is supported. This is an absolute time (e.g., 20/12/04 at 11:15) that determines the latest time a channel can deliver the event. If set to true, events later than the specified stop time are not sent.
-max_events_per_consumer	Specifies the maximum number of events that a channel queues for a consumer before it starts discarding them. Events are discarded in the order specified by -discard_policy. A setting of 0 specifies the channel to queue an unlimited number of events.

`-max_batch_size`	Specifies the maximum number of structured events sent in a sequence to consumers.
`-max_retries`	Specifies the maximum number of times that a proxy push supplier calls `push()` on its consumer before it gives up. The default value is `0`, which means an infinite number of retries.
`-pacing_interval`	Specifies the maximum amount of time that a channel is given to assemble structured events in a sequence, before delivering the sequence to consumers. The default value is `0`, which specifies an unlimited time.
`-timeout`	Specifies how long an event remains viable after the channel receives it. After the `-timeout` value expires, the event is discarded. The default is `0`, which means that events have an infinite lifetime.
`-pull_interval`	Specifies how much time elapses between attempts by a proxy pull consumer to call `pull()` or `try_pull()` on its supplier. The default value is `1` second.
`-retry_timeout`	Specifies how much time elapses between attempts by a proxy push supplier to call `push()` on its consumer. The default is `1` second.
`-max_retry_timeout`	Specifies the ceiling for `-retry_timeout`. This applies to timeouts directly assigned by developers as well as values reached by the multiplication of `-retry_multiplier` and`-retry_timeout`. The default value is `60` seconds.
`-request_timeout`	Specifies how much time is permitted to a channel object to perform an operation on a client. If the operation does not return within the specified limit, the operation throws a `CORBA::TRANSIENT` system exception.

`-retry_multiplier`	Specifies the number by which the current value of `-retry_timeout` is multiplied to determine the next `-retry_timeout` value. The `-retry_multiplier` value is applied until either the `push()` is successful or `-max_retry_timeout` is reached. The default value is `1.0`.

Examples

The following simple example sets the order and discard policies for an event channel named `my_channel`:

```
itadmin nc set_qos -order_policy fifo_order -discard_policy
   fifo_order my_channel
```

The following example sets the order policy and the priority for an event channel named `sales_channel`.

```
itadmin nc set_qos -order_policy priority_order sales_channel
itadmin nc set_qos -priority 3 sales_channel
```

The following enables start time for an event channel named `production_channel`:

```
itadmin nc set_qos -start_time_supported true production_channel
```

CHAPTER 27

Object Transaction Service

Overview

`itadmin` supports the object transaction service (OTS). Using `itadmin` commands in transactional mode ensures consistency and reliability in a distributed environment.

With `itadmin`, you can start, commit, rollback, suspend, and resume transactions. This lets you use other `itadmin` commands in transactional mode—for example, `process create`, or `orbname modify`.

A service can have several readers but only one writer. A transaction takes the writer thread. So, if you start a transaction in a service and then do not commit, roll back, or suspend the transaction, the service blocks until the timeout period expires (30 seconds). The transaction is then rolled back.

Similarly, if a transaction involving a service and the client (`itadmin` in this case) is terminated, the service is unaware of this and must be terminated.

You can manage transactions with the following `itadmin` commands:

Table 31: *Object Transaction Service Commands*

`tx begin`	Starts a transaction.
`tx commit`	Commits a transaction.
`tx resume`	Resumes a transaction.
`tx rollback`	Rolls back a transaction.

tx rollback	Rolls back a transaction.

tx begin

Synopsis

```
tx begin
```

Description

Starts a transaction. To use `itadmin` commands in a transaction, call `tx begin` followed by the other `itadmin` commands you wish to execute (for example, `orbname create`).

You must finalize the execution of these commands, using `tx commit`, or undo them, using `tx rollback`.

Examples

The following example starts a transaction, and then creates an ORB name:

```
itadmin
% tx begin
% orbname create MutualFunds.Tracking.GroInc.Stocks
```

tx commit

Synopsis

```
tx commit
```

Description

Commits a transaction. The commands executed after the transaction started using `tx begin` are finalized.

Examples

The following example commits the transaction:

```
itadmin
% tx begin
% orbname create MutualFunds.Tracking.GroInc.Stocks
% tx commit
```

tx resume

Synopsis

```
tx resume
```

Description Resumes a suspended transaction. Commands that occur after `tx resume` are part of the context of the transaction and are committed or rolled back at the conclusion of the transaction.

Examples The following example resumes the transaction:

```
itadmin
% tx begin
% orbname create MutualFunds.Tracking.GroInc.Stocks
% tx suspend
% tx resume
```

Note: You can not use more than one transaction at a time. You can not begin a transaction, suspend it and then begin another transaction. The `tx suspend` command should be only used to do non-transactional work before a subsequent `tx resume` command.

---

tx rollback

Synopsis `tx rollback`

Description Rolls back a transaction. The effects of commands executed after the transaction started using `tx begin` are undone.

Examples The following example rolls back the transaction:

```
itadmin
% tx begin
% orbname create MutualFunds.Tracking.GroInc.Stocks
% tx rollback
```

---

tx suspend

Synopsis `tx suspend`

Description Suspends a transaction. Commands that occur between `tx suspend` and `tx resume` are not part of the transaction, and are not committed or rolled back at the end of the transaction.

Examples

The following example suspends the transaction:

```
itadmin
% tx begin
% orbname create MutualFunds.Tracking.GroInc.Stocks
% tx suspend
```

CHAPTER 28

Object Transaction Service Encina

Overview

A subset of `itadmin` commands support the object transaction service (OTS) Encina plug-in.

In order to support the two-phase commit (*2PC*) protocol, an Encina OTS server needs a medium to log information about transactions—for example, IORs of the resources participating in a transaction. This medium is the *transaction log*, a logical entity consisting of or mirrored by one or more (physical) Encina volumes. Each volume in turn consists of one or more files or raw disks, which are said to back up the volume. Each of these volumes, or *mirrors*, contain the same information. This ensures recovery in case of failure of a machine that hosts some or all of a volume's constituent files/raw disks.

Transaction logs contain metadata, such as number and location of files or raw disks backing up the physical volumes that mirror the transaction log. Two files maintain this information:

- *Restart* file identifies an initialized transaction log.
- *Backup restart* file provides a backup to the restart file in case it is lost or corrupted by hardware failure.

For full information about two-phase commit and the Encina plug-in, see the *CORBA OTS Guide*.

You can manage the OTS Encina plug-in with the following `itadmin` commands:

`encinalog add`	Adds a file/raw disk to the list of files/raw disks backing up a physical volume of an Encina transaction log.
`encinalog add_mirror`	Creates a new physical volume and adds this to the list of volumes mirroring an Encina transaction log.
`encinalog create`	Creates a file for use in a transaction log—that is, a file that can be used to back up a physical volume mirroring an Encina transaction log.
`encinalog display`	Displays information about the physical volumes of an Encina transaction log.
`encinalog expand`	Expands an Encina transaction log.
`encinalog init`	Initializes an Encina transaction log, thereby creating restart and backup restart files.
`encinalog remove_mirror`	Removes a physical volume from an Encina transaction log.
`otstm stop`	Stops the otstm service.

Note: The commands described in this chapter assume the use of the `itadmin` command shell unless stated otherwise.

encinalog add

Synopsis

`encinalog add -restart` *`restart-file`* `[-backup` *`backup-file`*`] [-vol` *`vol-spec`*`] [-silent]` *`file-spec`*

Description

Adds a file/raw disk to the list of files/raw disks that back up the physical volume *`vol-spec`*, thereby increasing the total size of this volume.

If you omit the `-vol` argument, the file/raw disk is added to the list of files/raw disks backing up volume `logVol_physicalVol1`.

Arguments

`-restart` *`restart-file`*	Identifies the target transaction log.
`-backup` *`backup-file`*	Optionally identifies the target transaction log. If no backup restart file is specified, the default path is derived from *`restart-file`*`.bak`.

`-vol` *`vol-spec`*	Specifies a physical volume other than the default one.
`-silent`	Suppresses the display of the completion status.
`file-spec`	The path to an existing file (created with encinalog create) or raw disk.

Examples

The following example adds the file `ots2.log` to the physical volume `logVol_physicalVol2` which mirrors the transaction log identified by restart file `ots.restart` and backup restart file `ots.backup`:

```
itadmin encinalog add -restart ots.restart -backup ots.backup -
   vol logVol_physicalVol2 ots2.log
```

Note: Use the `encinalog display` command to list the named of the individual physical volumes mirroring the transaction log.

---

encinalog add_mirror

Synopsis

`encinalog add_mirror -restart` *`restart-file`* `-backup` *`backup-file`* `[-silent]` *`file-spec`*

Description

Creates a physical volume backed up by *`file-spec`*, and adds it to the list of physical volumes mirroring the transaction log.

The new physical volume is named `logVol_physicalVol`*`n`*, where *`n`* is the lowest number for which there is no physical volume mirroring the transaction log.

Arguments

`-restart` *`restart-file`*	Identifies the target transaction log.
`-backup` *`backup-file`*	Optionally identifies the target transaction log. If no backup restart file is specified, the default path is derived from *`restart-file`*`.bak`.
`-silent`	Suppresses the display of the completion status.
`file-spec`	The path name of a file or raw disk created with `encinalog create`.

Examples

The following example adds a physical volume backed up by file `otsmirror.log` to the to the list of volumes mirroring the transaction log identified by restart file `ots.restart` and backup restart file `ots.backup`:

```
itadmin encinalog add_mirror -restart ots.restart -backup
   ots.backup otsmirror.log
```

encinalog create

Synopsis

`encinalog create` *`[-size-type file-size]`* `[-replace]` `[-silent]` *`file-spec`*

Description

Creates a file, *`file-spec`*, which can be used to back up a physical volume of an Encina transaction log. The default size is 4 megabytes.

Arguments

`-size-type file-size`	Specifies a non-default size, where *`-size-type`* is one of the following literals: • `-msize` specifies the size in megabytes. • `-ksize` specifies the size in kilobytes. • `-size` specifies the size in bytes. The minimum size is 1 megabyte; the maximum size is 16 megabytes.
`-replace`	Overwrites an existing file.
`-silent`	Suppresses the display of the completion status.

Examples

The following example creates a file of size 2 megabytes and overwrites an existing file of the same name:

```
itadmin encinalog create -msize 2 -replace ots.log
```

encinalog display

Synopsis

`encinalog display -restart` *`restart-file`* `[-backup` *`backup-file`*`]`

Description

Displays information on the physical volumes mirroring the transaction log.

Arguments

`-restart` *`restart-file`*	Identifies the target transaction log.
`-backup` *`backup-file`*	Optionally identifies the target transaction log. If no backup restart file is specified, the default path is derived from *`restart-file`*`.bak`.

Examples

The following example displays information on the physical volumes of a transaction log identified by `ots.restart` and the backup restart file `ots.backup`:

```
itadmin encinalog display -restart ots.restart -backup
   ots.backup
%
Logical Volume:        logVol
Free Pages:            960
Total Number of Pages: 1016
Physical Volume:       logVol_physicalVol1
    File Name:          /tmp/ots.log
Physical Volume:       logVol_physicalVol2
    File Name:          /tmp/otsmirror.log
```

---

encinalog expand

Synopsis

`encinalog expand -restart` *`restart-file`* `[-backup` *`backup-file`*`] [-silent]`

Description

Expands the transaction log to its maximum size, which is the minimum of the individual physical volume sizes. These, in turn, are the accumulated sizes of the files/raw disks backing up the individual physical volumes. The operation is necessary after the size of all physical volumes has been increased by adding files/raw disks to the volumes.

Arguments

`-restart` *`restart-file`*	Identifies the transaction log to expand
`-backup` *`backup-file`*	Optionally identifies the transaction log to expand. If no backup restart file is specified, the default path is derived from *`restart-file`*`.bak`.
`-silent`	Suppresses the display of the completion status.

Examples

The following example expands the logical volume associated with `ots.restart` and the backup restart file `ots.backup`:

```
itadmin encinalog expand -restart ots.restart -mirror ots.backup
```

encinalog init

Synopsis

`encinalog init [-replace] [-restart` *`restart-file`*`] [-backup` *`backup-file`*`] [-silent]` *`file-spec`*

Description

Initializes an Encina transaction log, mirrored by one physical volume `logVol_physicalVol1`, and backed up by the file/raw disk *`file-spec`*.

The command also creates restart and backup files. You can explicitly name these files; otherwise, the default restart file and backup restart file names are *`file-spec`*`_restart` and *`file-spec`*`_restart.bak`, respectively.

Arguments

`-restart` *`restart-file`*	Specifies the restart file name.
`-backup` *`backup-file`*	Optionally identifies the transaction log to initialize. If no backup restart file is specified, the default path is derived from *`restart-file`*`.bak`.
`-replace`	Overwrites the existing restart files.
`-silent`	Suppresses the display of the completion status.

Examples

The following example initializes a transaction log using alternative names for the restart and backup restart files:

```
itadmin encinalog init -restart ots.restart -backup ots.backup
   ots.log
```

encinalog remove_mirror

Synopsis

`encinalog remove_mirror -restart` *`restart-file`* `[-backup` *`backup-file`*`] [-silent]` *`vol-spec`*

Description

Removes the physical volume *`vol-spec`* from the list of volumes mirroring the transaction log.

Arguments

`-restart` *restart-file*	Identifies the target transaction log.
`-backup` *backup-file*	Optionally identifies the target transaction log. If no backup restart file is specified, the default path is derived from *restart-file*`.bak`.
`-silent`	Suppresses the display of the completion status.

Examples

The following example removes the physical volume `logVol_physicalVol1` from the transaction log identified by `ots.restart` and backup restart file `ots.backup`:

```
itadmin encinalog remove_mirror -restart ots.restart -backup
   ots.backup logVol_physicalVol1
```

Note: See `encinalog init` and `encinalog add_mirror` for the possible names of a physical volume, or use the `encinalog display` command to get the names of the physical volumes mirroring a transaction log. Because a transaction log needs at least one mirror, `remove_mirror` will not allow you to remove a physical volume if it is the only volume.

---

otstm stop

Synopsis

```
otstm stop
```

Description

Stops the otstm service.

Persistent State Service

Overview

A subset of `itadmin` commands let you manage the persistent state service (PSS). PSS is a CORBA service for building CORBA servers that access persistent data and include transactional support. PSS is for use with C++ applications only. For more details about PSS, see the *CORBA Programmer's Guide*.

You can manage a PSS database using the following commands:

Table 32: *Persistent State Service Commands*

`pss_db archive_old_logs`	Archives old log files for the specified IOR.
`pss_db checkpoint`	Performs checkpoint operations on the database referenced in the specified file.
`pss_db delete_old_logs`	Deletes old log files for specified IOR.
`pss_db list_replicas`	Lists the replicas for the specified IOR.
`pss_db name`	Returns the name of the object reference to the database.
`pss_db post_backup`	Performs post-backup operations on the database referenced in the specified file.

Table 32: *Persistent State Service Commands*

`pss_db pre_backup`	Performs pre-backup operations on the database referenced in the specified file.
`pss_db remove_replica`	Removes a replica from the database's replica group.
`pss_db show`	Returns replication related information for the specified IOR.

pss_db archive_old_logs

Synopsis

```
pss_db archive_old_logs IOR-file
```

Description

Archives old log files for the specified IOR. The *IOR-file* argument specifies the full pathname to the file that contains the object reference.

pss_db checkpoint

Synopsis

```
pss_db checkpoint IOR-file
```

Description

Performs checkpoint operations on the database referenced in the file. The *IOR-file* argument specifies the full pathname to the file that contains the object reference.

When using transactions, Berkeley DB maintains transaction log files. Each time a transaction commits, data is appended to the transaction log files, and the database files are not modified. Data in transaction log files is then transferred periodically to the database files. This transfer is called a *checkpoint*. You can specify the checkpoint interval, using the following configuration variable:

```
plugins:pss_db:envs:env_name:checkpoint_interval
```

For example, `plugins:pss_db:envs:locator:checkpoint_interval`.

The checkpoint operation performs a Berkeley DB checkpoint. The following configuration variable specifies whether to delete the old log files, or move them to another directory:

```
plugins:pss_db:envs:env_name:checkpoint_deletes_old_logs
```

The following configuration variable specifies the directory to which log files should be moved:

```
plugins:pss_db:envs:env_name:old_log_dir
```

For more details on these configuration variables, see the section discussing the `plugins:pss_db` namespace in the *Orbix Configuration Reference*.

pss_db delete_old_logs

Synopsis

```
pss_db delete_old_logs IOR-file
```

Description

Deletes old log files for specified IOR. The *IOR-file* argument specifies the full pathname to the file that contains the object reference.

pss_db list_replicas

Synopsis

```
pss_db list_replicas [-active] IOR-file
```

Returns the names of all replicas for the database specified in the file containing the object reference.

Arguments

`-active`	List only active replicas.
IOR-file	Specifies the full pathname to file that contains the object reference.

pss_db name

Synopsis

```
pss_db name IOR-file
```

Description

Returns the name of the object reference to the persistent state database.

The *IOR-file* argument specifies the full pathname to the file that contains the object reference.

pss_db post_backup

Synopsis

```
pss_db post_backup IOR-file
```

Description

Performs post-backup operations on the database referenced in the file. The *IOR-file* argument specifies the full pathname to the file that contains the object reference.

When backing up data files, it is important that no checkpoint occurs during the backup. The pre-backup operations force a checkpoint and then suspend checkpointing. The post-backup operations resume checkpointing.

pss_db pre_backup

Synopsis

```
pss_db pre_backup IOR-file
```

Description

Performs pre-backup operations on the database referenced in the file. The *IOR-file* argument specifies the full pathname to file that contains the object reference.

When backing up data files, it is important that no checkpoint occurs during the backup. The pre-backup operations force a checkpoint and then suspend checkpointing. The post-backup operations resume checkpointing.

pss_db remove_replica

Synopsis

```
pss_db remove_replica [-iorfile IOR-file] [-envhome env-dir]
replica-name
```

Description

Removes the replica specified *replica-name* from the replica group. The *-iorfile* or *envhome* argument must be specified, depending on whether the service containing the database is running or not.

The `remove_replica` command should only be used when removing a service's replica. See the *Orbix Deployment Guide* for more details.

Arguments

`-iorfile`	Specifies the path to the file containing the databases reference. This argument is used to remove a replica when the replica group is running.
`-envhome`	Specifies the path to the database root directory. This argument is used when the service containing the database is not running. It only removes the replica from the local database.

pss_db show

Synopsis

```
pss_db show IOR-file
```

Description

Returns information about the specified database. This includes:

- database name
- whether the database is replicated
- database replica name
- whether the database is a master or slave.

The *IOR-file* argument specifies the full pathname to file that contains the object reference.

CHAPTER 30

Security Service

Overview

The `itadmin` tool supports security commands to administer the key distribution management (KDM) database, which is part of SSL/TLS for CORBA. The KDM is a security feature that enables automatic activation of secure Orbix servers—see the *CORBA SSL/TLS Guide* for details.

Key distribution management

Key distribution management (KDM) is a mechanism that distributes pass phrases to a secure server during automatic activation. Without the KDM, it is impossible to activate a secure server automatically because pass phrases must be supplied manually when the server starts up.

The KDM also protects a server's implementation repository (IMR) entry from unauthorized tampering. Whenever a *process IMR entry* is updated, the KDM requires a security checksum to be generated (using the `checksum create` command). The process IMR entry is the part of an IMR record that stores the server executable location. Before activating a secure server, the KDM checks that the stored checksum matches the current checksum for the process IMR entry.

The KDM framework consists of the following elements:

- A *KDM server* provides security attributes to the locator on request.
- A *KDM database* is used by the KDM server to store security attributes.
- A *KDM administration plug-in* provides the security commands described in this section and communicates directly with the KDM server. SSL/TLS installs a secure KDM administration plug-in the `itadmin` utility.

KDM database

The KDM database stores the following kinds of security attributes:

- *Pass phrases* are associated with an ORB name and stored as a security attribute in the KDM database. The pass phrases are supplied to a secure server during automatic activation.
- *Checksums* are associated with a process name and stored as a security attribute in the KDM database. The checksum is tested against the current process IMR record before a server is automatically activated.

The process IMR record used by the checksum algorithm includes all of the fields associated with the `itadmin process` command except the process description.

The security commands are mainly concerned with managing the entries in the KDM database—creating, updating, and removing security attributes.

All of these commands require a secure connection to the KDM database. It is therefore necessary to log on to the KDM server, using `admin_logon`, prior to issuing any of the security commands.

Commands

`itadmin` commands let you manage the following security service activities:

Logging On

Overview

You log on to the KDM server with the `itadmin admin_logon` command.

admin_logon

Synopsis

`admin_logon login [-password` *`pass-phrase`*`] identity`

Description

Logs an administrator on to the KDM server. This command must be issued prior to any of the other secure commands (`kdm_adm` or `checksum`).

Arguments

`login`	This argument specifies the name of an X.509 certificate that identifies the administrator. The *`identity`* parameter specifies the name of a PKCS#12 certificate file, *`identity`*`.p12`, located in the directory specified by the `itadmin_x509_cert_root` configuration variable.
`-password`	This argument lets you specify the pass phrase for the *`identity`*`.p12` certificate on the same line as the command, instead of being prompted for it. This argument is provided for scripting in a development environment and should not be used in a live system.

Examples

To log on to the KDM server, before issuing any secure commands, enter the following at the command line:

```
itadmin
% admin_logon login my_admin_id
Please enter password for identity my_admin_id:
%
```

The `Enter password` prompt lets you enter the pass phrase for the `my_admin_id.p12` certificate without echoing to the screen.

Managing Checksum Entries

Overview

The following `itadmin` commands let you manage checksum entries:

Table 33: *Checksum Entry Commands*

`checksum confirm`	Confirms that the process IMR entry for the specified process has not been changed since the checksum was created.
`checksum create`	Creates a checksum for the specified process IMR entry and store the checksum in the KDM database.
`checksum list`	Lists process names that have security checksum information in the KDM database.
`checksum remove`	Removes a security checksum entry from the KDM database.

checksum confirm

Synopsis

`checksum confirm -process` *`process-name`*

Description

Confirms that the process IMR entry for *`process-name`* has not been modified since the checksum entry in the KDM database was created.

Arguments

`-process` Specifies the name, *`process-name`*, of a process IMR entry.

Examples

To confirm that the checksum previously stored for the `my_process_name` process agrees with the checksum for the current `my_process_name` IMR entry, enter the following at the command line:

```
itadmin
% admin_logon login my_admin_id
Please enter password for identity my_admin_id:
% checksum confirm -process my_process_name
The checksum is valid.
%
```

checksum create

Synopsis

`checksum create -process` *`process-name`*

Description

Creates a checksum entry in the KDM database for the process *`process-name`*. The checksum must be recreated whenever the process IMR entry for the specified process is modified.

Arguments

`-process` Specifies the name, *`process-name`*, of a process IMR entry.

Examples

To create a checksum entry in the KDM database for `my_process_name`, enter the following at the command line:

```
itadmin
% admin_logon login my_admin_id
Please enter password for identity my_admin_id:
% checksum create -process my_process_name
%
```

checksum list

Synopsis

`checksum list [-count]`

Description

Lists the names of all processes that have checksum entries in the KDM database.

Arguments

`-count` Returns a count of the number of checksum entries, instead of listing them.

Examples

To list all process names with checksum entries in the KDM database, enter the following at the command line:

```
itadmin
% admin_logon login my_admin_id
Please enter password for identity my_admin_id:
% checksum list
simple_process
%
```

checksum new_pw

Synopsis

```
checksum new_pw
```

Description

Password protects the checksum entry in the KDM database.

checksum remove

Synopsis

```
checksum remove -process process-name
```

Description

Removes the checksum entry associated with the *process-name* process name from the KDM database.

Arguments

`-process` Specifies the name, *process-name*, of a process IMR entry.

Examples

To remove the checksum entry associated with `my_process_name` from the KDM database, enter the following at the command line:

```
itadmin
% admin_logon login my_admin_id
Please enter password for identity my_admin_id:
% checksum remove -process my_process_name
Security checksum associated with process my_process_name has
   been removed.
%
```

Managing Pass Phrases

Overview

The following `itadmin` commands let you manage pass phrases:

Table 34: *Pass Phrase Commands*

`kdm_adm change_pw`	Changes the pass phrase for encrypting the KDM database.
`kdm_adm confirm`	Confirms that the pass phrase associated with the specified ORB name has the value you expect.
`kdm_adm create`	Creates an entry in the KDM database that associates a pass phrase with the specified ORB name.
`kdm_adm list`	Lists the ORB names that have pass phrase information in the KDM database.
`kdm_adm new_pw`	Creates a new pass phrase for encrypting the KDM database.
`kdm_adm remove`	Removes an entry from the KDM database associated with the specified ORB name.

kdm_adm change_pw

Synopsis

```
kdm_adm change_pw
```

Description

Changes the pass phrase used to encrypt the KDM database. The command prompts you for the current pass phrase and then prompts you twice for the new pass phrase (to ensure it was entered correctly).

Examples

To change the KDM database pass phrase, enter the following at the command line:

```
itadmin
% admin_logon login my_admin_id
Please enter password for identity my_admin_id:
% kdm_adm change_pw
Please enter the current KDM password:
Please enter the new KDM password:
Please confirm the new KDM password:
%
```

After entering the `admin_logon` command, you are prompted for the `my_admin_id.p12` certificate pass phrase.

After entering the `kdm_adm change_pw` command, you are prompted three times for pass phrases. In response to the first `Enter password` prompt, enter the current KDM database pass phrase. In response to the second and third `Enter password` prompts, enter the new KDM database pass phrase.

kdm_adm confirm

Synopsis

`kdm_adm confirm -orbname` *`ORB-name`*

Description

Confirms the pass phrase associated with the specified ORB name, *`ORB-name`*. The command prompts you for the pass phrase associated with *`ORB-name`* and tells you whether or not you entered the correct pass phrase.

Examples

To confirm the pass phrase associated with the `my_orb_name` ORB name, enter the following at the command line:

```
itadmin
% admin_logon login my_admin_id
Please enter password for identity my_admin_id:
% kdm_adm confirm -orbname my_orb_name
Please enter password for orb my_orb_name :
The password is correct.
%
```

kdm_adm create

Synopsis

`kdm_adm create -orbname` *`ORB-name`* `[-password` *`pass-phrase`*`]`

Description

Creates an entry in the KDM database to associate a pass phrase with the specified ORB name, *ORB-name*. Just one pass phrase can be associated with an ORB name. If the -password argument is omitted, the command prompts you for a pass phrase which is not echoed to the screen.

Arguments

-orbname	Specifies the ORB name, *ORB-name*, with which the new pass phrase is associated.
-password	Lets you specify a new pass phrase. This argument is provided for scripting purposes during development and should not be used in a live system.

Examples

To associate a pass phrase with the my_orb_name ORB name and store the association in the KDM database, enter the following at the command line:

```
itadmin
% admin_logon login my_admin_id
Please enter password for identity my_admin_id:
% kdm_adm create -orbname my_orb_name
Please enter password for orb my_orb_name :
%
```

kdm_adm list

Synopsis

```
kdm_adm list [-count]
```

Lists all ORB names that have associated pass phrases stored in the KDM database.

Arguments

-count	Returns a count of the number of ORB name entries instead of listing them.

Examples

To list all ORB names that have associated pass phrases, enter the following at the command line:

```
itadmin
% admin_logon login my_admin_id
Please enter password for identity my_admin_id:
% kdm_adm list
my_orb_name
%
```

kdm_adm new_pw

Synopsis

```
kdm_adm new_pw
```

Description

Creates a new pass phrase for encrypting the KDM database.

kdm_adm remove

Synopsis

```
kdm_adm remove -orbname ORB-name
```

Description

Removes the security entry in the KDM database associated with the *ORB-name* ORB name.

Examples

To remove the security entry associated with the `my_orb_name` ORB name, enter the following at the command line:

```
itadmin
% admin_logon login my_admin_id
Please enter password for identity my_admin_id:
% kdm_adm remove -orbname my_orb_name
Security attributes associated with orbname my_orb_name have been
   removed.
%
```

CHAPTER 31

Trading Service

Overview

`itadmin` provides a set of commands for managing the following trading service components:

Trading Service Administrative Settings

Overview

The following commands let you mange trading service administrative settings:

Table 35: *Trading Service Commands*

`trd_admin get`	Displays administrative settings.
`trd_admin set`	Modifies administrative settings.
`trd_admin stop`	Stops the trading service.

trd_admin get

Synopsis

`trd_admin get` *arg*

Description

Displays administrative settings.

Arguments

Supply one of the following arguments:

`-request_id_stem`	Displays the request id stem assigned to this instance of the trading service.
`-def_search_card`	Displays the default search cardinality-the default upper bound of offers to be searched.
`-max_search_card`	Displays the maximum search cardinality-maximum upper bound of offers to be searched.
`-def_match_card`	Displays the default match cardinality-default upper bound of matched offers to be ordered.
`-max_match_card`	Displays the maximum match cardinality-maximum upper bound of matched offers to be ordered.
`-def_return_card`	Displays the default return cardinality-default upper bound of ordered offers to be returned.
`-max_return_card`	Displays the maximum return cardinality-maximum upper bound of ordered offers to be returned.

-max_list	Displays the upper bound on the size of any list returned by the trading service, namely the returned offers parameter in query, and the next_n operations in `OfferIterator` and `OfferIdIterator`.
-modifiable_properties	Displays whether the trading service supports properties modification.
-dynamic_properties	Displays whether the trading service supports dynamic properties.
-proxy_offers	Displays whether the trading service supports proxy offers.
-def_hop_count	Displays the default hop count-default upper bound of depth of links to be traversed in a federated query.
-max_hop_count	Displays the maximum hop count-maximum upper bound of depth of links to be traversed in a federated query.
-def_follow_policy	Displays the default federation link follow policy.
-max_follow_policy	Displays the limiting link follow policy for all links of the trader. This setting overrides both link and importer policies.
-max_link_follow_policy	Displays the most permissive follow policy allowed when creating new links.
-type_repos	Displays the stringified IOR of the service type type repository.

Examples

```
>itadmin trd_admin get -type_repos
IOR:000000000000036494….

> itadmin trd_admin get -proxy_offers
yes

>itadmin trd_admin get -def_follow_policy
always

>itadmin trd_admin get -max_list
2147483647
```

trd_admin set

Synopsis

```
trd_admin set arg
```

Description

Modifies administrative settings.

Arguments

Supply one of the following arguments:

`-request_id_stem id_stem`	Modifies the request id stem of this instance of the trading service.
`-def_search_card value`	Modifies the default search cardinality-the default upper bound of offers to be searched. The value must be a positive integer.
`-max_search_card value`	Modifies the maximum search cardinality-the maximum upper bound of offers to be searched. The value must be a positive integer.
`-def_match_card value`	Modifies the default match cardinality-the default upper bound of matched offers to be ordered. The value must be a positive integer.
`-max_match_card value`	Modifies the maximum match cardinality-the maximum upper bound of matched offers to be ordered. The value must be a positive integer.
`-def_return_card value`	Modifies the default return cardinality-the default upper bound of ordered offers to be returned. The value must be a positive integer.
`-max_return_card value`	Modifies the maximum return cardinality-the maximum upper bound of ordered offers to be returned. The value must be a positive integer.
`-max_list value`	Modifies the upper bound on the size of any list returned by the trading service, namely the returned offers parameter in query, and the next_n operations in `OfferIterator` and `OfferIdIterator`. The value must be a positive integer.

`-modifiable_properties` *boolean-value*	Specifies whether to enable support of modifiable properties.
`-dynamic_properties` *boolean-value*	Specifies whether to enable support of dynamic properties.
`-proxy_offers` *boolean-value*	Specifies whether to enable support of proxy offers.
`-def_hop_count` *value*	Sets the default hop count-the default upper bound of depth of links to be traversed in a federated query. The value must be a positive integer.
`-max_hop_count`	Sets the maximum hop count-the maximum upper bound of depth of links to be traversed in a federated query.
`-def_follow_policy` *policy*	Sets the default federation link follow policy with one of the following values: • `local_only` • `if_no_local` • `always`
`-max_follow_policy` *policy*	Sets the limiting link follow policy for all links of the trader. This setting overrides both link and importer policies. Supply one of the following values: • `local_only` • `if_no_local` • `always`
`-max_link_follow_policy` *policy*	Specifies the most permissive follow policy allowed when creating new links with one of the following values: • `local_only` • `if_no_local` • `always`
`-type_repos` *IOR*	Sets the IOR, in string format, of the service type repository.

Examples

```
>itadmin trd_admin set -def_search_card 12
def_search_card set to 12
```

trd_admin stop

Stops the trading service.

Federation Links

Overview

The following commands let you mange federation links:

Table 36: *Federation Link Commands*

`trd_link create`	Creates a federation link.
`trd_link list`	Lists all federation links.
`trd_link modify`	Modifies a federation link.
`trd_link remove`	Removes a federation link.
`trd_link show`	Displays the details on a federation link.

trd_link create

Synopsis

```
trd_link create
   -target IOR
   -def_pass_on_follow_rule rule
   -limiting_follow_rule rule
   link-name
```

Description

Creates a federation link.

Arguments

`-target` *IOR*

Defines the trading service instance the link points to. An IOR to a `CosTrading::Lookup` interface is expected.

`-def_pass_on_follow_rule` *rule*

Defines default link-follow behavior to pass on for a particular link, if an importer does not specify its `link_follow_rule`; it must not exceed `limiting_follow_rule`. Supply one of the following values for *rule*:

- `local_only`
- `if_no_local`
- `always`

`-limiting_follow_rule` *rule*	Defines limiting link follow behavior for a particular link. Supply one of the following values for *rule*: • `local_only` • `if_no_local` • `always`
link-name	A string that uniquely identifies the new link in the trading service instance.

Examples

```
>itadmin trd_link create -target 'cat ./trader_B_lookup.ior'
   -def_pass_on_follow_rule always -limiting_follow_rule always
   Link_to_Trader_B
created link Link_to_Trader_B
```

trd_link list

Synopsis

```
trd_link list
```

Description

Lists names of all federation links in the trading service instance.

Examples

```
>itadmin trd_link list
Link_to_Trader_B
```

trd_link modify

Synopsis

```
trd_link modify
   -def_pass_on_follow_rule rule
   -limiting_follow_rule rule
   link-name
```

Description

Modifies an existing federation link.

Arguments

`-def_pass_on_follow_rule` *rule*	Defines the default link-follow behavior to be passed on for a particular link if an importer does not specify its `link_follow_rule`; it must not exceed `limiting_follow_rule`. Supply one of the following values for *rule*: • `local_only` • `if_no_local` • `always`
`-limiting_follow_rule` *rule*	Defines limiting link follow behavior for a particular link. Supply one of the following values for *rule*: • `local_only` • `if_no_local` • `always`
link-name	A string that uniquely identifies the new link in the trading service instance.

Examples

```
>itadmin trd_link modify -def_pass_on_follow_rule if_no_local
   -limiting_follow_rule always Link_to_Trader_B
modified link Link_to_Trader_B
```

trd_link remove

Synopsis

`trd_link remove` *link-name*

Description

Removes the specified federation link.

Arguments

link-name	A string that uniquely identifies the link to be removed from the trading service instance.

Examples

```
>itadmin trd_link remove Link_to_Trader_B
removed link Link_to_Trader_B
```

trd_link show

Synopsis

`trd_link show link-name`

Description

Displays details on the specified federation link.

Arguments

link-name A string that uniquely identifies the link whose details are to be displayed.

Examples

```
>itadmin trd_link show Link_to_Trader_B
name:
        Link_to_Trader_B
def_pass_on_follow_rule:
        if_no_local
limiting_follow_rule:
        always
target:
limiting_follow_rule:
        IOR:000000000000002249…
```

Regular Offers

Overview

The following commands let you mange regular offers:

Table 37: *Regular Offer Commands*

`trd_offer list`	Lists all regular offers.
`trd_offer remove`	Removes a regular offer.
`trd_offer show`	Displays details on a regular offer.

trd_offer list

Synopsis

`trd_offer list`

Description

Lists the offer IDs of all regular (non-proxy) offers.

Examples

```
>itadmin trd_offer list
Printer~1~0
```

trd_offer remove

Synopsis

`trd_offer remove` *`offer-id`*

Description

Removes (withdraws) the specified offer.

`Arguments`

`offer-id` Offer ID of an existing offer.

Examples

```
>itadmin trd_offer remove Printer~1~0
offer Printer~1~0 removed
```

trd_offer show

Synopsis

`trd_offer show` *offer-id*

Description

Displays details on the specified offer.

Arguments

offer-id Offer ID of an existing offer.

Examples

```
>itadmin trd_offer show Printer~1~0
offer id:
        Printer~1~0
object:
        IOR:0000000000000224…
service type:
        Printer
properties:
        boolean color TRUE
        long dpi 3200
        short ppm 30
```

Proxy Offers

Overview

The following commands let you manage proxy offers:

Table 38: *Proxy Offer Commands*

`trd_proxy list`	Lists all proxy offers.
`trd_proxy remove`	Removes a proxy offer.
`trd_proxy show`	Displays details on a proxy offer.

trd_proxy list

Synopsis

`trd_proxy list`

Description

Lists the offer IDs of all proxy offers

Examples

```
>itadmin trd_proxy list
Printer~2~0
```

trd_proxy remove

Synopsis

`trd_proxy remove` *`offer-id`*

Description

Removes (withdraws) the specified proxy offer.

`Arguments`

`offer-id`	Offer ID of an existing proxy offer

Examples

```
>itadmin trd_proxy remove Printer~2~0
proxy offer Printer~2~0 removed
```

trd_proxy show

Parameters

trd_proxy show *offer-id*

Description

Displays details on the specified proxy offer.

Arguments

offer-id	Offer ID of an existing proxy offer

Examples

```
>itadmin trd_proxy show Printer~2~0
offer id:
        Printer~2~0
service type:
        Printer
target:
        IOR:0000000000000224…
if match all:
        TRUE
constraint recipe:
        ppm > 20
policies to pass on:
        boolean bool_policy FALSE
properties:
        boolean color FALSE
        long dpi 3200
        short ppm 12
```

Type Repository

Overview

They following commands effect the server type repository:

Table 39: *Server Type Repository Commands*

`trd_type list`	Lists all service types in the service type repository.
`trd_type mask`	Masks a service type.
`trd_type remove`	Removes a service type from the service type repository.
`trd_type show`	Displays details on a given service type.
`trd_type unmask`	Unmasks a service type.

trd_type list

Synopsis

`trd_type list`

Description

Lists all service types in the service type repository.

Examples

```
>itadmin trd_type list
Printer
```

trd_type mask

Synopsis

`trd_type mask` *`service-type-name`*

Description

Masks a service type.

Examples

```
>itadmin trd_type mask Printer
service type Printer masked
```

trd_type remove

Synopsis

`trd_type remove` *service-type-name*

Description

Removes a service type from the service type repository.

Examples

```
>itadmin trd_type remove Printer
service type Printer removed
```

trd_type show

Synopsis

`trd_type show` *service-type-name*

Description

Displays details on a given service type.

Examples

```
>itadmin trd_type show Printer
name:
        Printer
interface:
        IDL:PrintServer:1.0
masked:
        no
incarnation number:
        {0,1}
super types:
        none
properties:
        mandatory read-only boolean color
        mandatory long dpi
        mandatory read-only short ppm
```

trd_type unmask

Synopsis

`trd_type unmask` *`service-type-name`*

Description

Unmasks a service type.

Examples

```
>itadmin trd_type unmask Printer
service type Printer unmasked
```

Part 5

Appendices

In this part

This part contains the following:

ORB Initialization Settings

Initialization settings can be set for an ORB through command-line arguments, which are passed to the initializing ORB.

In most cases, equivalent environment variables or Java properties are available. In the absence of command-line arguments, these are used by the initializing ORB.

Initialization parameters pertain to the immediate requirements of the initializing ORB; for example, the name of its configuration domain and location, and the naming scope in which to find the ORB's configuration. The ORB's behavior is further defined by its configuration, as set by configuration variables. For more information about these, refer to the *Configuration Reference*.

Precedence of settings

Most initialization parameters can be set in one of the following ways, in descending order of precedence:

- Command-line arguments.
- Environment variables or Java properties.
- Default values.

Java properties

Java properties can be set for an initializing ORB in two ways, in descending order of precedence:

- Set as system properties. For example:

```
java -DORBdomain_name finance corporate.finance_app
```

- Set in the properties file `iona.properties`.

An initializing ORB searches for the properties file in the following locations, in this order:

1. Current directory.
2. Directories on the classpath.
3. Jars on the classpath.

Domains directory

The directory that contains the target configuration file; set with:

Command-line argument: `-ORBconfig_domains_dir`

Environment variable: `IT_CONFIG_DOMAINS_DIR`

Java property: `ORBconfig_domains_dir`

This directory typically stores a file for each accessible configuration domain name.

For example:

```
my_app -ORBconfig_domains_dir C:\Program Files\Micro
   Focus\etc\domains
```

Nothing else should be stored in this directory. This enables tools to easily enumerate the list of available domains.

The configuration domains directory defaults to *`ORBconfig_dir`*`/domains` on UNIX, and *`ORBconfig_dir`*`\domains` on Windows.

Domain name

The name of the configuration domain to use; set with:

Command-line argument: `-ORBdomain_name`

Environment variable: `IT_DOMAIN_NAME`

Java property: `ORBdomain_name`

For example:

```
my_app -ORBdomain_name my_domain
```

Configuration directory

The root configuration directory; set with:

Command-line argument: `-ORBconfig_dir`

Environment variable: `IT_CONFIG_DIR`

Java property: `ORBconfig_dir`

Specifies the root configuration directory. The default root configuration directory is `/etc/opt/microfocus` on UNIX, and *`product-dir`*`\etc` on Windows.

ORB name

The ORB name, which specifies the configuration scope for this ORB; set with:

Command-line argument only: `-ORBname`

The following application takes it configuration from the `my_orb` scope:

```
my_app -ORBname my_orb
```

You can also use the `-ORBname` parameter to specify non-default configuration scopes for Orbix services. For example:

```
itconfig_rep -ORBname config_rep.config2 run
```

Initial reference

An initial object reference for a service using the interoperable naming service format; set with:

Command-line argument only: `-ORBInitRef`

For example:

```
-ORBInitRef NameService=IOR00023445AB...
-ORBInitRef
   NotificationService=corbaloc:555objs.com/NotificationService

-ORBInitRef TradingService=corbaname:555objs.com/Dev/Trader
```

Default initial reference

An initial object reference to a service if none is explicitly specified by `-ORBInitRef`; set with:.

Command-line argument only: `-ORBDefaultInitRef`

This parameter takes a URL, which forms a new URL identifying an initial object reference. For example:

```
my_app -ORBDefaultInitRef corbaloc:555objs.com
```

A call to `resolve_initial_references("NotificationService")` with the following argument results in a new URL:

```
corbaloc:555.objs.com/NotificationService
```

The new URL has a '/' character and a stringified object key appended.

Product directory

The directory in which Orbix products are installed, set with:

Command-line argument: `-ORBproduct_dir`

Environment variable: `IT_PRODUCT_DIR`

Java property: `ORBproduct_dir`

For example:

```
my_app -ORBproduct_dir c:\Micro Focus Orbix
```

This directory is read-only and location independent. This enables it to be shared across systems even if mounted at different locations.

The directory in which products are installed defaults to `/opt/microfocus` on UNIX, and `%SystemDrive%\Program Files\Micro Focus` on Windows.

Development Environment Variables

For C++ installations, you can specify several environment variables that pertain to development environments only.

IT_IDL_CONFIG_FILE

Specifies the configuration file for the IDL compiler.

UNIX

Defaults to `$IT_INSTALL_DIR/asp/version/etc/idl.cfg`.

Windows

Defaults to `%IT_INSTALL_DIR%\asp\version\etc\idl.cfg`.

Note: Do not modify the default IDL configuration file. This affects demo programs and other applications. Instead, use this variable to point the IDL compiler to a customized file if necessary.

IT_IDLGEN_CONFIG_FILE

Specifies the configuration file for the Orbix code generation toolkit.

UNIX

Defaults to `$IT_INSTALL_DIR/asp/version/etc/idlgen.cfg`.

Windows

Defaults to `%IT_INSTALL_DIR%\asp\version\etc\idlgen.cfg`.

APPENDIX C

Named Keys for Orbix Services

This appendix lists the named keys for the Orbix services and associated configuration variables.

In this appendix

This appendix includes the following sections:

Orbix Service Named Key Strings

Table 40 shows the key strings used by each service.

Table 40: *Orbix Service Key Strings*

Service	Plain Text Forwarder Key	IMR Key	IOR Prefix	Initial Reference
Security	IT_SecurityService	n/a	IT_SecurityService	IT_SecurityService
	IT_Login	n/a	IT_Login	IT_Login
Configuration Repository (CFR)	ConfigRepository	n/a	ConfigRepository	ConfigRepository
	IT_ConfigRepositoryReplica	n/a	IT_SingleConfigRepository	n/a
Firewall Proxy Service (FPS)	IT_FPS_Registry	n/a	IT_FPS_Registry	IT_FPS_Registry
	IT_FPS_Manager	n/a	IT_FPS_Manager	IT_FPS_Manager
Management	IT_ManagementService.User	n/a	IT_MgmtServiceUser	IT_MgmtServiceUser
	IT_ManagementService.Registration	n/a	IT_MgmtService	IT_MgmtService
	IT_ManagementService.Security	n/a	IT_MgmtServiceSec	IT_MgmtServiceSec
Locator	IT_Locator	n/a	IT_Locator	IT_Locator
	IT_LocatorReplica	n/a	IT_SingleLocator	n/a
Node daemon	IT_NodeDaemon	n/a	IT_NodeDaemon	n/a
Transaction monitor	TransactionServiceAdmin	TransactionServiceAdmin	TransactionServiceAdmin	TransactionServiceAdmin

Service	Plain Text Forwarder Key	IMR Key	IOR Prefix	Initial Reference
	TransactionFactory	TransactionFactory	TransactionFactory	TransactionFactory
Interface Repository	InterfaceRepository	InterfaceRepository	InterfaceRepository	InterfaceRepository
Naming	NameService	NameService	NameService	NameService
	IT_NameServiceReplica	n/a	IT_SingleNameService	n/a
Trader	TradingService	TradingService	TradingService	TradingService
	TradingServiceNR	n/a	n/a	n/a
	Replicator	n/a	n/a	n/a
Basic Log	DefaultBasicLogFactory	BasicLoggingService	BasicLoggingService	BasicLoggingService
Event Log	DefaultEventLogFactory	EventLoggingService	EventLoggingService	EventLoggingService
Notification Log	DefaultNotifyLogFactory	NotifyLoggingService	NotifyLoggingService	NotifyLoggingService
Notification	DefaultEventChannelFactory	NotificationService	NotificationService	NotificationService
	DefaultEndpointAdmin	n/a	IT_NotificationEndpointAdmin	IT_NotificationEndpointAdmin
Event	DefaultEventChannelFactory	EventService	EventService	EventService
	DefaultTypesEventChannelFactory	n/a	n/a	n/a
JMS	MessageBroker	IT_JMSMessageBroker	IT_JMSMessageBroker	IT_JMSMessageBroker
	ServerContext	n/a	IT_JMSServerContext	IT_JMSServerContext
	MessagingBridge	n/a	IT_MessagingBridge	IT_MessagingBridge
	EndpointAdmin	n/a	IT_JMSEndpointAdmin	IT_JMSEndpointAdmin

Configuration for Advertising Services

Table 41 shows the configuration variables for each service (where applicable). Setting one of these variables to `true` prevents registration of a key with the plain text key forwarder for that service.

Table 41: *Advertise Service Configuration Variables*

Service	Configuration Variable Name
Firewall Proxy Service (FPS)	`fps:advertise_services`
Transaction monitor	`plugins:ots:advertise_services`
Interface Repository	`plugins:ifr:advertise_services`
Naming	`plugins:naming:advertise_services`
Trader	`trader:advertise_services`
Basic Log	`plugins:basic_log:advertise_services`
Event Log	`plugins:event_log:advertise_services`
Notification Log	`plugins:notify_log:advertise_services`
Notification	`plugins:notify:advertise_services`
Event	`plugins:event:advertise_services`

Glossary

A

administration

All aspects of installing, configuring, deploying, monitoring, and managing a system.

ART

Adaptive Runtime Technology. Micro Focus's modular, distributed object architecture, which supports dynamic deployment and configuration of services and application code. ART provides the foundation for Orbix software products.

ATLI2

Abstract Transpot Layer Interface, version 2. Micro Focus's current transport layer implementation.

C

Certificate Authority

Certificate Authority (CA). A trusted third-party organization or company that issues digital certificates used to create digital signatures and public-private key pairs. The role of the CA in this process is to guarantee that the individual granted the unique certificate is, in fact, who he or she claims to be. CAs are a crucial component in data security and electronic commerce because they guarantee that the two parties exchanging information are really who they claim to be.

CFR

See configuration repository.

client

An application (process) that typically runs on a desktop and requests services from other applications that often run on different machines (known as server processes). In CORBA, a client is a program that requests services from CORBA objects.

configuration

A specific arrangement of system elements and settings.

configuration domain

Contains all the configuration information that Orbix ORBs, services and applications use. Defines a set of common configuration settings that specify available services and control ORB behavior. This information consists of configuration variables and their values. Configuration domain data can be implemented and maintained in a centralized Orbix configuration repository or as a set of files distributed among domain hosts. Configuration domains let you organize ORBs into manageable groups, thereby bringing scalability and ease of use to the largest environments. See also configuration file and configuration repository.

configuration file

A file that contains configuration information for Orbix components within a specific configuration domain. See also configuration domain.

configuration repository

A centralized store of configuration information for all Orbix components within a specific configuration domain. See also configuration domain.

configuration scope

Orbix configuration is divided into scopes. These are typically organized into a root scope and a hierarchy of nested scopes, the fully-qualified names of which map directly to ORB names. By organizing configuration properties into various scopes, different settings can be provided for individual ORBs, or common settings for groups of ORB. Orbix services, such as the naming service, have their own configuration scopes.

CORBA

Common Object Request Broker Architecture. An open standard that enables objects to communicate with one another regardless of what programming language they are written in, or what operating system they run on. The CORBA specification is produced and maintained by the OMG. See also OMG.

CORBA naming service

An implementation of the OMG Naming Service Specification. Describes how applications can map object references to names. Servers can register object references by name with a naming service repository, and can advertise those

names to clients. Clients, in turn, can resolve the desired objects in the naming service by supplying the appropriate name. The Orbix naming service is an example.

CORBA objects

Self-contained software entities that consist of both data and the procedures to manipulate that data. Can be implemented in any programming language that CORBA supports, such as C++ and Java.

CORBA transaction service

An implementation of the OMG Transaction Service Specification. Provides interfaces to manage the demarcation of transactions and the propagation of transaction contexts. Orbix OTS is such as service.

CSIv2

The OMG Common Secure Interoperability protocol v2.0, which can be used to provide the basis for application-level security in both CORBA and J2EE applications. The Orbix Security Framework implements CSIv2 to transmit usernames and passwords, and to assert identities between applications.

D

deployment

The process of distributing a configuration or system element into an environment.

H

HTTP

HyperText Transfer Protocol. The underlying protocol used by the World Wide Web. It defines how files (text, graphic images, video, and other multimedia files) are formatted and transmitted. Also defines what actions Web servers and browsers should take in response to various commands. HTTP runs on top of TCP/IP.

I

IDL
Interface Definition Language. The CORBA standard declarative language that allows a programmer to define interfaces to CORBA objects. An IDL file defines the public API that CORBA objects expose in a server application. Clients use these interfaces to access server objects across a network. IDL interfaces are independent of operating systems and programming languages.

IFR
See interface repository.

IIOP
Internet Inter-ORB Protocol. The CORBA standard messaging protocol, defined by the OMG, for communications between ORBs and distributed applications. IIOP is defined as a protocol layer above the transport layer, TCP/IP.

implementation repository
A database of available servers, it dynamically maps persistent objects to their server's actual address. Keeps track of the servers available in a system and the hosts they run on. Also provides a central forwarding point for client requests. See also location domain and locator daemon.

IMR
See implementation repository.

installation
The placement of software on a computer. Installation does not include configuration unless a default configuration is supplied.

Interface Definition Language
See IDL.

interface repository

Provides centralized persistent storage of IDL interfaces. An Orbix client can query this repository at runtime to determine information about an object's interface, and then use the Dynamic Invocation Interface (DII) to make calls to the object. Enables Orbix clients to call operations on IDL interfaces that are unknown at compile time.

invocation

A request issued on an already active software component.

IOR

Interoperable Object Reference. See object reference.

L

location domain

A collection of servers under the control of a single locator daemon. Can span any number of hosts across a network, and can be dynamically extended with new hosts. See also locator daemon and node daemon.

locator daemon

A server host facility that manages an implementation repository and acts as a control center for a location domain. Orbix clients use the locator daemon, often in conjunction with a naming service, to locate the objects they seek. Together with the implementation repository, it also stores server process data for activating servers and objects. When a client invokes on an object, the client ORB sends this invocation to the locator daemon, and the locator daemon searches the implementation repository for the address of the server object. In addition, enables servers to be moved from one host to another without disrupting client request processing. Redirects requests to the new location and transparently reconnects clients to the new server instance. See also location domain, node daemon, and implementation repository.

N

naming service

See CORBA naming service.

node daemon

Starts, monitors, and manages servers on a host machine. Every machine that runs a server must run a node daemon.

O

object reference

Uniquely identifies a local or remote object instance. Can be stored in a CORBA naming service, in a file or in a URL. The contact details that a client application uses to communicate with a CORBA object. Also known as interoperable object reference (IOR) or proxy.

OMG

Object Management Group. An open membership, not-for-profit consortium that produces and maintains computer industry specifications for interoperable enterprise applications, including CORBA. See www.omg.com.

ORB

Object Request Broker. Manages the interaction between clients and servers, using the Internet Inter-ORB Protocol (IIOP). Enables clients to make requests and receive replies from servers in a distributed computer environment. Key component in CORBA.

OTS

See CORBA transaction service.

P

POA

Portable Object Adapter. Maps object references to their concrete implementations in a server. Creates and manages object references to all objects used by an application, manages object state, and provides the infrastructure to support persistent objects and the portability of object implementations between different ORB products. Can be transient or persistent.

protocol

Format for the layout of messages sent over a network.

S

server

A program that provides services to clients. CORBA servers act as containers for CORBA objects, allowing clients to access those objects using IDL interfaces.

SSL

Secure Sockets Layer protocol. Provides transport layer security—authenticity, integrity, and confidentiality—for authenticated and encrypted communications between clients and servers. Runs above TCP/IP and below application protocols such as HTTP and IIOP.

SSL handshake

An SSL session begins with an exchange of messages known as the SSL handshake. Allows a server to authenticate itself to the client using public-key encryption. Enables the client and the server to co-operate in the creation of symmetric keys that are used for rapid encryption, decryption, and tamper detection during the session that follows. Optionally, the handshake also allows the client to authenticate itself to the server. This is known as mutual authentication.

T

TCP/IP

Transmission Control Protocol/Internet Protocol. The basic suite of protocols used to connect hosts to the Internet, intranets, and extranets.

TLS

Transport Layer Security. An IETF open standard that is based on, and is the successor to, SSL. Provides transport-layer security for secure communications. See also SSL.

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z