IONA

Orbix®

Administrator's Guide

Version 6.1, December 2003

Making Software Work Together™

M 3 1 3 0

Updated: 18-Jun-2004

Contents

Part II Managing an Orbix Environment

Part IV Appendices

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 2.4 of the OMG'S CORBA specification. This guide explains how to configure and manage the components of an Orbix environment.

If you need help with this or any other IONA product, contact IONA at `support@iona.com`. Comments on IONA documentation can be sent to `docs-support@iona.com`.

Audience

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

Organization

This guide is divided into the following parts:

- Getting Started introduces the Orbix environment, and the basic concepts required to understand how it works. It also describes how to run the Orbix configuration tool, which creates a configuration domain and deploys applications.
- Managing an Orbix Environment explains how to manage each component of an Orbix environment. It provides task-based information and examples.
- 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*

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 I

Getting Started

In this part

This part contains the following chapters:

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.

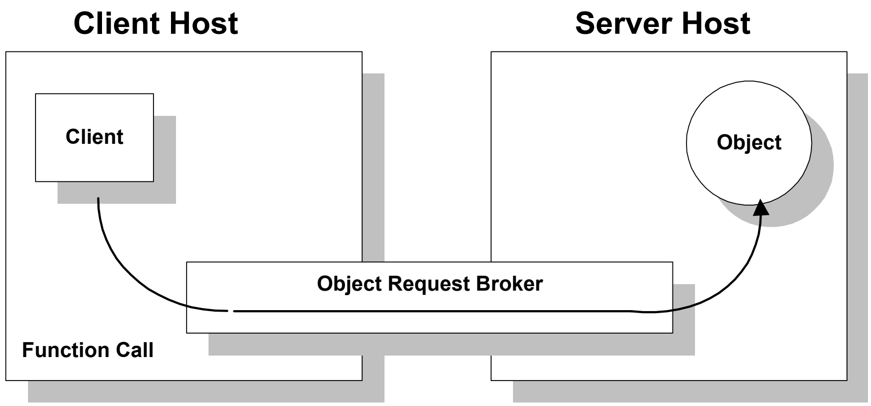

Figure 1: *Basic CORBA Model*

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.

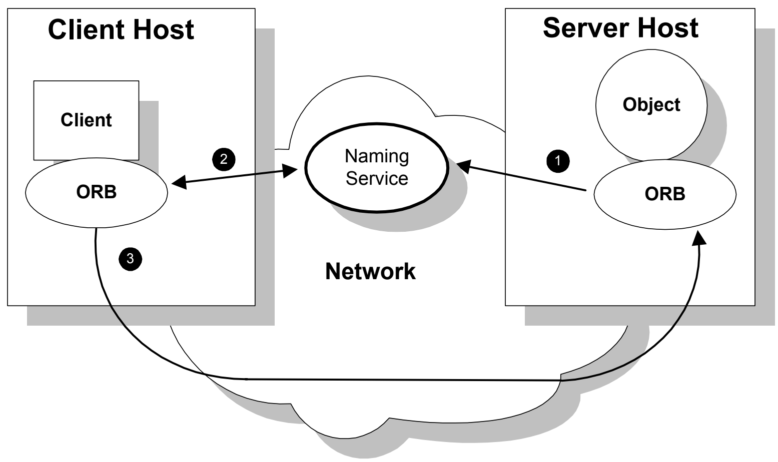

Figure 2: *Overview of a Simple Orbix Application*

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.

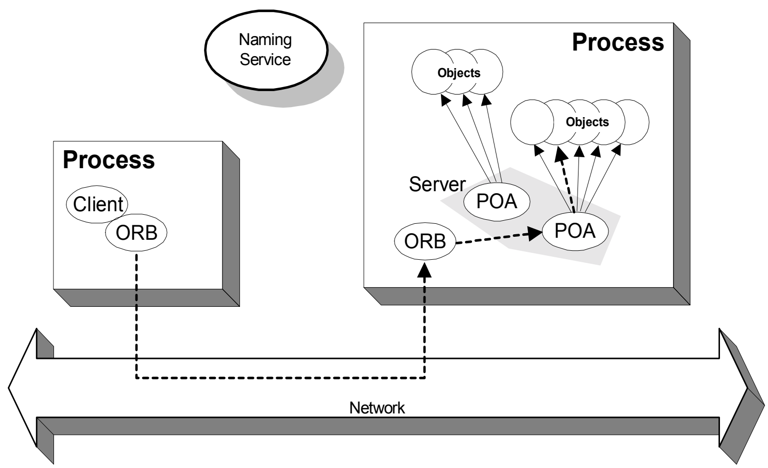

Figure 3: *A POA's Role in Client–Object Communication*

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 i*nterface 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 an 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.

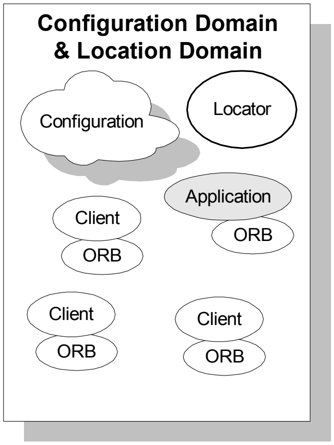

Figure 4: *Simple Configuration Domain and Location Domain*

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.

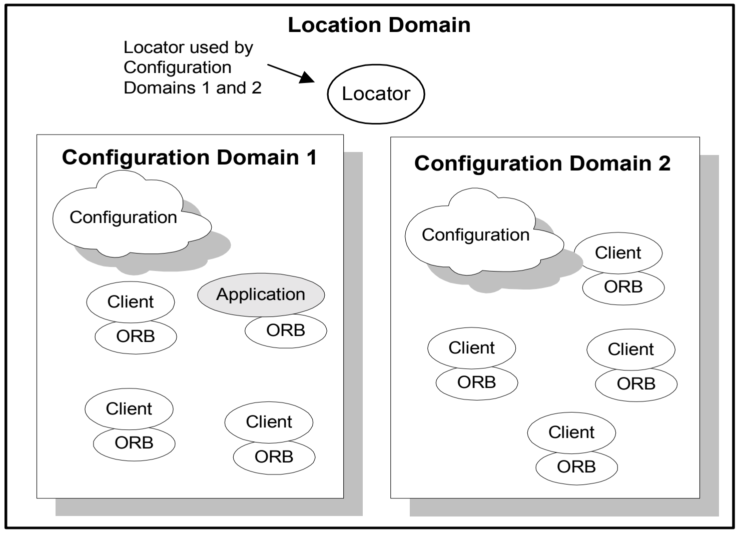

Figure 5: *Multiple Configuration Domains*

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 8.

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 II 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 4.

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 11.

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 III of this manual.

CHAPTER 2

Selecting a Configuration Model

This chapter shows how the Orbix can be configured in a network environment.

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. This chapter 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 configuration, see Chapter 3.

In this chapter

This chapter contains the following sections:

Orbix Development Environment Models

Overview

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.

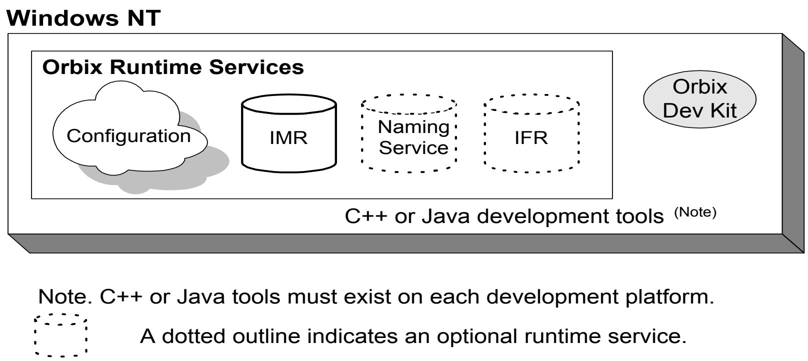

Figure 6: *An Independent Development and Test Environment*

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 20. 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.

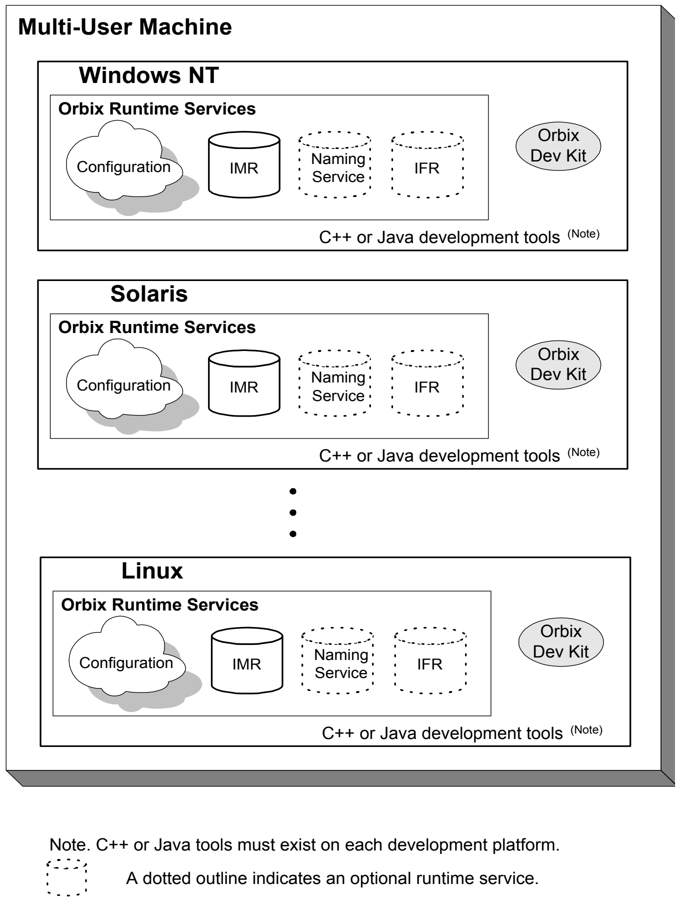

Figure 7: *Multiple Independent Development and Test Environments*

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 Orbix runtime services in your environment (for example, a locator daemon), see Chapter 3.

Distributed Development and Test Environments

Overview

Figure 8 on page 23 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 Orbix runtime services such as the interface repository in your environment, see Chapter 3.

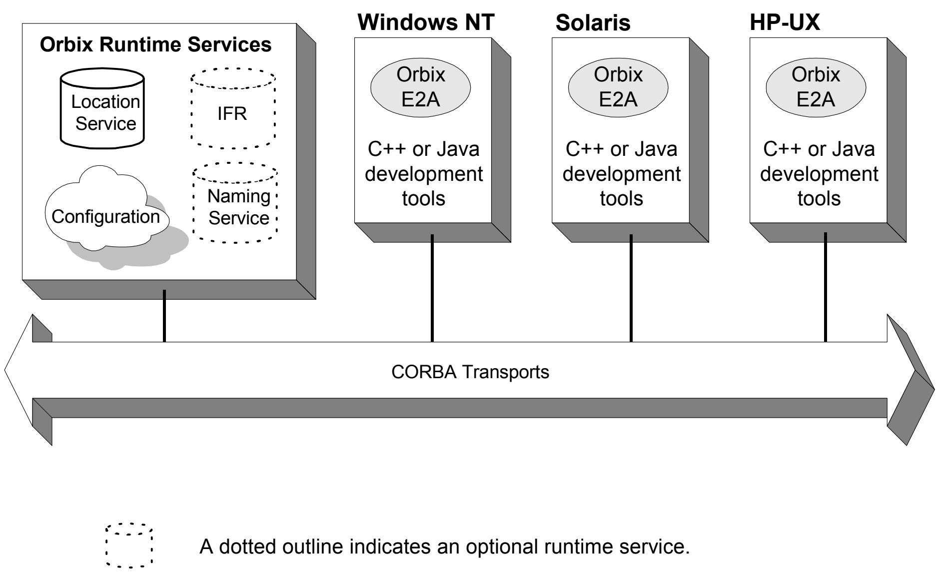

Figure 8: *A Distributed Development and Test Environment*

Configuration Models

Overview

Orbix provides two configuration mechanisms:

- Local file-based configuration
- Configuration repository

For information on managing Orbix configuration domains, see Chapter 4.

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 25 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 25 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 25.

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.

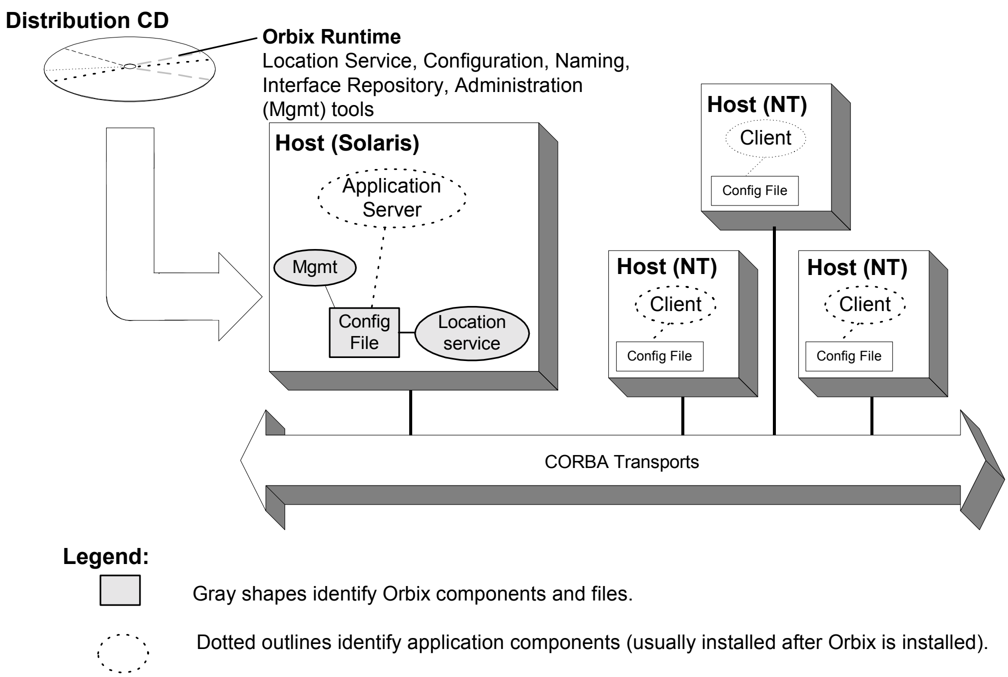

Figure 9: *Orbix Environment with Local Configuration*

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 26 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.

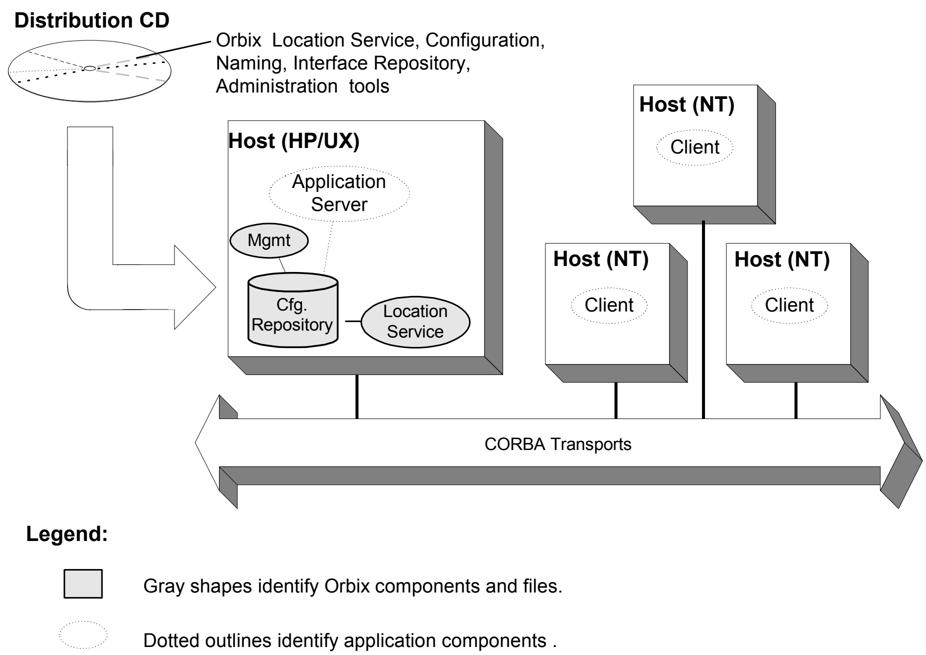

Figure 10: *Orbix Environment with Centralized Configuration*

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 28
- "Using the Right Data Storage Mechanism" on page 30

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.

Configuring an Orbix Domain

Orbix provides a GUI based configuration tool to guide you through generating an environment into which applications can be deployed.

In this chapter

This chapter discusses the following topics:

Running the Orbix Configuration Tool

Overview

The Orbix configuration tool guides you through licensing and configuring Orbix components in your environment. You can use it to perform the following tasks:

- Install or update your licenses.
- Create a configuration domain.
- Deploy services into a configuration domain.
- Link to existing configuration domains.
- Create configuration replicas for clustering.

The Orbix configuration tool analyzes your installation and provides you with the options available for your system.

Prerequisites

Before you run the configuration tool, check the following requirements and constraints:

- Set `JAVA_HOME` so it points to your current Java installation.
- Set UNIX access permissions to account for the following contingencies:
    - The configuration tool must have write access to directories `/var/opt/iona` and `/etc/opt/iona`. These directories are usually restricted to accounts with `superuser` privileges.
    - The configuration tool prompts you to designate a user to run domain services, and sets ownership of files and directories accordingly.
- Set the `IT_PRODUCT_DIR` environment variable to point to the latest Orbix installation on your system.

Syntax

To run the configuration tool, use the following command:

```
itconfigure
              [-ORBproduct_dir install_dir]
              [-ORBlicense_file license_file]
              [-nogui]
              [-gui]
              [-load, -l domain_descriptor]
              [-compatible]
              [-entities file]
              [-save, -s file]
              [-localize]
              [-name domain_name]
              [-file, -f]
              [-cfr, -c]
              [-link, -i hostname]
              [-expert, -e]
              [-host, -h hostname]
              [-multihome hostname]
              [-etc config_dir]
              [-var var_dir]
              [-range, -r base_port]
              [-port iiop_port]
              [-tlsport tls_port]
              [-credentials credentials]
              [-hostnamePolicy policy]
              [-libs, -L path]
              [-help, -?]
              [-demos]
```

The configuration tool options are described as follows:

`-ORBproduct_dir` *`install_dir`*	Specifies your installation directory when Orbix is installed in a non-default location and the `IT_PRODUCT_DIR` environment variable is not set.
`-ORBlicense_file` *`license_file`*	Specifies your license directory when the Orbix license file is not stored in the default location and the `IT_LICENSE_FILE` environment variable is not set. For more details, see "Licensing" on page 39.

`-nogui`	Runs the configuration tool silently. This option can be used with `-load`, `-localize`, `-save`, and `-range`. For example, see "Configuring a Machine with no GUI" on page 63.
`-gui`	Runs the configuration tool GUI. This is the default.
`-load, -l` *`domain_descriptor`*	Loads a preconfigured domain descriptor file. When used in conjunction with `-nogui`, silently deploys the local pieces of the configuration defined in the deployment descriptor. For more details, see "Localizing a Preconfigured Domain" on page 58, and "Deploying a Domain on the Remaining Hosts" on page 61.
`-compatible`	For interoperability with previous versions, this loads the specified file with the `-load` option as an Orbix 5.x driver file. For more details, see the *Deployment Guide*.
`-entities` *`filename`*	Uses the specified entities file when loading the driver file specified with the `-load` option. For more details, see the *Deployment Guide*.
`-save, -s` *`filename`*	Saves a deployment descriptor in the specified file. When used with `-nongui`, this option will not deploy the saved configuration.
`-localize`	Replaces all deployment nodes in a descriptor with the local host. For more details, see "Localizing a Preconfigured Domain" on page 58.
`-name` *`domain_name`*	Specifies the name of the domain. The specified name will override the name in a loaded domain descriptor.
`-file, -f`	Creates a file based configuration. This option will override the setting in a loaded domain descriptor.
`-cfr, -c`	Creates a repository-based configuration. This option overrides the setting in a loaded domain descriptor.

`-link` *`cfr_host`*	Specifies the machine which hosts the domain's configuration repository.
`-expert, -e`	Causes the GUI to skip straight to **Expert** mode. For more details, see "Using Expert Mode" on page 53.
`-host, -h` *`hostname`*	Specifies the name of the domain's host machine. This setting overrides the setting in a loaded domain descriptor.
`-multihome` *`hostname`*	Denotes that the specified host is virtual on a multi-homed host. For more details, see "Configuring a Multihomed Machine" on page 64.
`-etc` *`etc_dir`*	Specifies the directory where configuration information is stored.
`-var` *`var_dir`*	Specifies the directory where database files are stored.
`-range` *`base_port`*	Specifies the base port number from which to begin allocating port numbers. This option is only used in conjunction with `-nogui`.
`-port` *`iiop_port`*	Overrides the default CFR IIOP port when used with `-link`.
`-tlsport` *`tls_port`*	Overrides the default CFR TLS port when used with `-link`.
`-libs, -L` *`path`*	Prefixes the library path to the built-in path used when preparing and running Orbix services. For more details, see the *Deployment Guide*.
`-credentials`	Specifies credentials in the following format: `"username=<name>, \` `password_file=<file>,domain=<domain>"`

`-hostnamePolicy` *`policy`*	Specifies the address mode policy for IORS. Value can be one of the following: • `ip` (IP addresses) • `localhost` ('localhost'), • `localhost_ip` ('`127.0.0.1`'), • `long` (fully qualified hostname), • `short` (unqualified hostname—the default). For more details, see the *Deployment Guide*.
`-help, -?`	Displays an explanation of the command flags.
`-demos`	Specifies the configuration needed to run the Orbix demos in the domain.

Configuration tool screen

When the configuration tool first runs you should see the following screen:

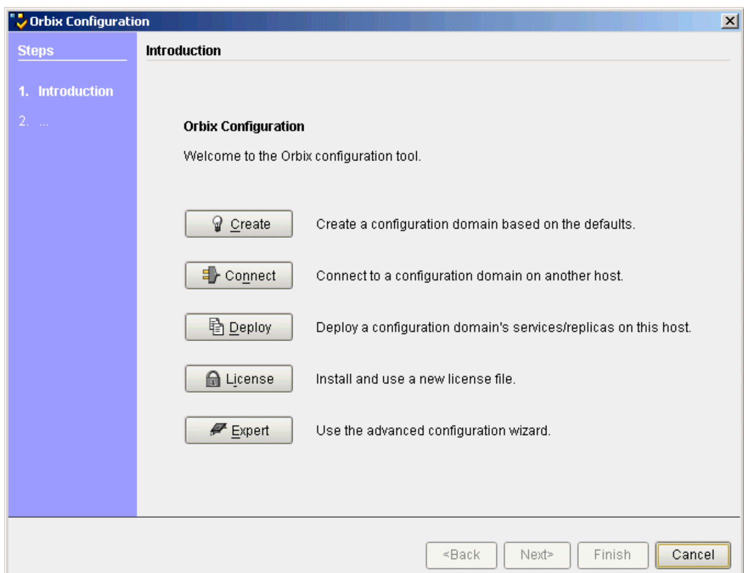

Figure 11: *Main Configuration Window*

From this screen you can chose to perform any of the configuration tasks.

Licensing

Overview

The Orbix configuration tool enables you to specify a location for your license file.

Specifying a license file

To specify a license file:

1. From the Orbix configuration tool's main screen, click **License**.
2. This displays a dialog similar to the one shown in Figure 12:

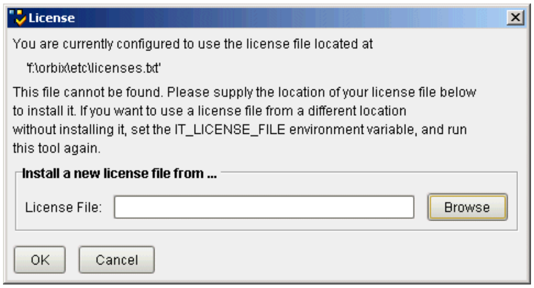

Figure 12: *Entering the License File*

The default locations are as follows:

Windows	<*install-dir*>\etc\licenses.txt
UNIX	<*install-dir*>/etc/opt/iona/licenses.txt

3. Enter the location of license file you wish to install in the **License File** text box. Alternatively, use the **Browse** to navigate to the file. You should have received this file from your IONA representative and stored it in a secure location.
4. Click **OK** to return to the main screen.

The `licenses.txt` file is copied from your specified location. Any existing license files are overwritten. When you have specified a license file, you will not need to perform these steps again.

Configuring an Orbix Domain

Overview

A configuration domain contains all the configuration information used by Orbix ORBs, services, and applications. The Orbix configuration tool configures and deploys Orbix components into a configuration domain. It can also link a machine to an existing configuration domain.

Centralized domain design

The Orbix configuration tool provides a centralized mechanism for designing a distributed configuration domain. While designing your domain, you specify all of the machines that are to host services in your domain, which services are run on each machine, and which machines, if any, host replicas. You can also deploy location services onto machines that will host custom servers.

Once you have designed your configuration, you must then go to each machine in the domain and deploy the configuration. This populates each machines configuration databases and properly deploys the services on each machine.

Configuration options

- **Create**: This enables you to create a new configuration domain from scratch. It is used to determine the type of configuration being created, what ports the core services use, and what services will be deployed into the domain.
- **Deploy**: This enables you to deploy replicated services into a domain. It is also used to deploy services on the host machines in a domain. For more information, see "Replicating Services in a Domain" on page 67.
- **Connect**: This enables you to connect a machine to an existing configuration domain. The new machine will link to the existing configuration repository to retrieve its configuration information.

> **Note:** This option is will fail to create a domain if the configuration repository is not running or if the domain is file based.

- **Expert**: This enables you to create a new configuration domain from scratch. It is similar to using **Create**, but it provides access to advanced configuration options. This option is only recommended if you are familiar with Orbix administration.

In this section

This section discusses the following topics:

Creating a New Domain

Overview

The Orbix configuration tool's **Create** option allows you to create a new configuration domain, or modify an existing one, by walking you through the procedure and providing basic configuration options.

For more advanced configuration options use the **Expert** option.

Procedure

To create a configuration domain, follow these steps:

1. Start the Orbix configuration tool.
2. Select **Create**.
3. You will see a screen similar to Figure 13.

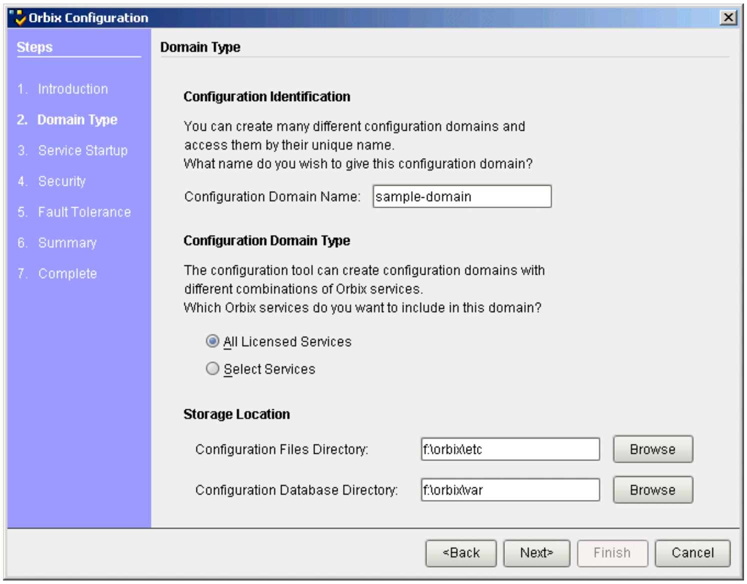

Figure 13: *Domain Type Screen*

4. Specify the domain name. If you are creating a new domain, this name should be unique among any pre-deployed configuration domains. If it is not, the existing domain will be overwritten.

5. Set the level of services to deploy into the domain by selecting one of the following options:
    - **All Licensed Services** automatically deploys all services for which you have purchased licenses.
    - **Select Services** allows you to select which services you wish to deploy into the domain on the particular machine.
6. Specify the directories where you would like configuration data stored on this system. In most cases the defaults are sufficient.
7. Click **Next** to select how your services will start. You should see a screen similar to the one in Figure 14.

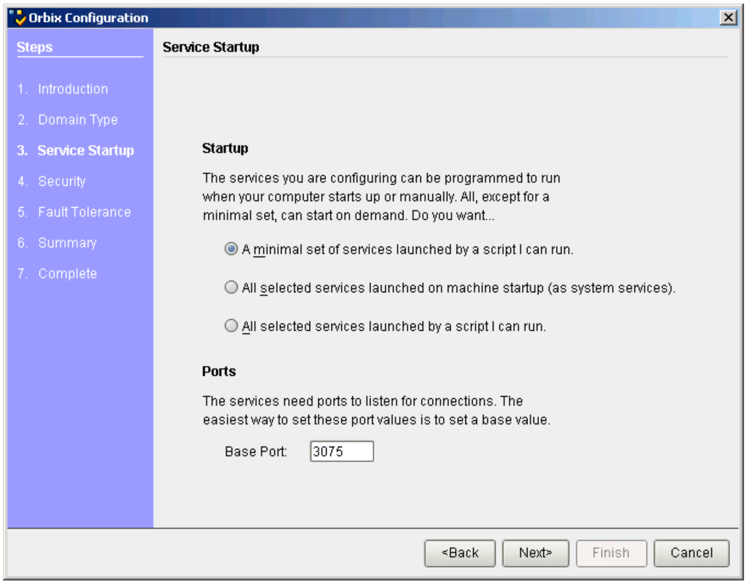

Figure 14: *Startup Mode and Base Port*

8. Choose one of the following options:
    - **A minimal set of services launched by a script I can run** generates a script that to start the location service and, if it selected, the configuration repository. All other deployed services will be started on demand.

- **A minimal set of services launched at machine startup** configures the location service and, if selected, the configuration repository to start up when the machine is booted. All other deployed services will be started on demand.

Note: When the proceeding options are selected, the location service is deployed by default. You will not be able to unselect it.

- **All services launched by a script I can run** generates a script that will start all deployed services.

9. Enter a number for the **Base Port**. This is the number from which Orbix will begin sequentially assigning listener ports for its services.
10. Click **Next** to configure your domains security features. You should see a dialog similar to Figure 15.

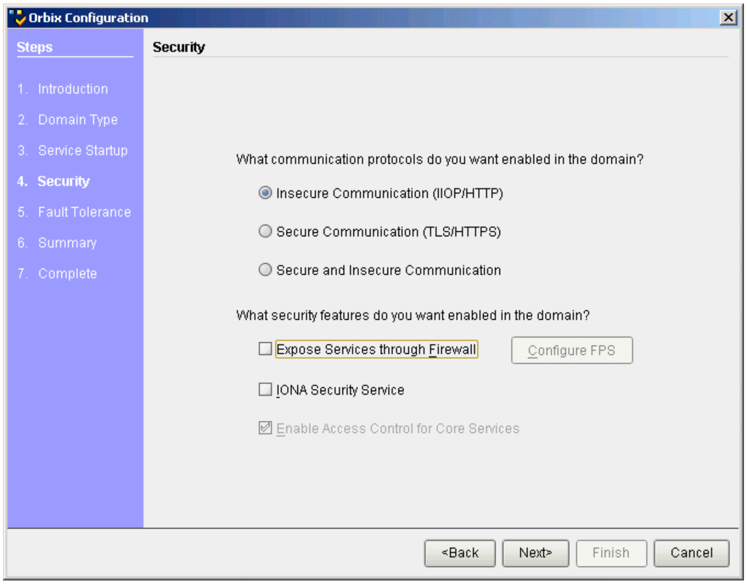

Figure 15: *Setting Security Features*

11. Select what security protocols you wish enabled.

- **Insecure communication (IIOP/HTTP)** configures your domain so that it will not use TLS or HTTPS protocols. It will reject any attempts to make a secure connection.

Note: This is the only mode in which the Firewall Proxy Service will run.

- **Secure communication (TLS/HTTPS)** configures your system so that all communication is done securely. Any attempts to make a connection using a protocol other than TLS or HTTPS will be rejected.
- **Secure and insecure communication** configures your system so that it can use IIOP, TLS, HTTP, and HTPS protocols.

Note: This option is automatically selected if you configure the IS2 Security Infastructure. You will only be able to select secure communication.

12. Select the security features you wish to enable in the domain:
    - **Expose services through Firewall** configures your domain to use the firewall proxy service.

Note: This option is only available for insecure domains.

- **IONA Security Service** configures your domain to take advantages of the IONA security platform. For more information read the *Security Guide*.

Note: This option forces you to use TLS and HTTPS. Therefore the firewall proxy service is unavailable.

- **Enable access control for core services** is only available for use when the IS2 security infrastructure is configured, For more information read the *Security Guide*.

13. Click **Next** to configure any replicas you wish to include in your domain. You should see a dialog similar to Figure 16.

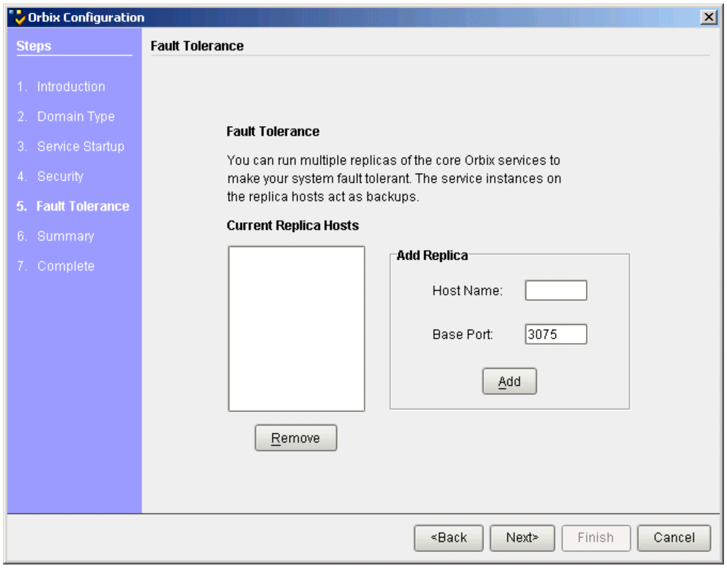

Figure 16: *Replica Configuration*

14. To add a replica to the domain, enter the machine's host name and a listener port, then click **Add**. To remove a replica from the list, highlight its hostname and click **Remove**.
15. When you have specified all of the replicas for your domain, click **Next**.

16. If you chose to deploy only selected services, you will see a dialog similar to Figure 17.

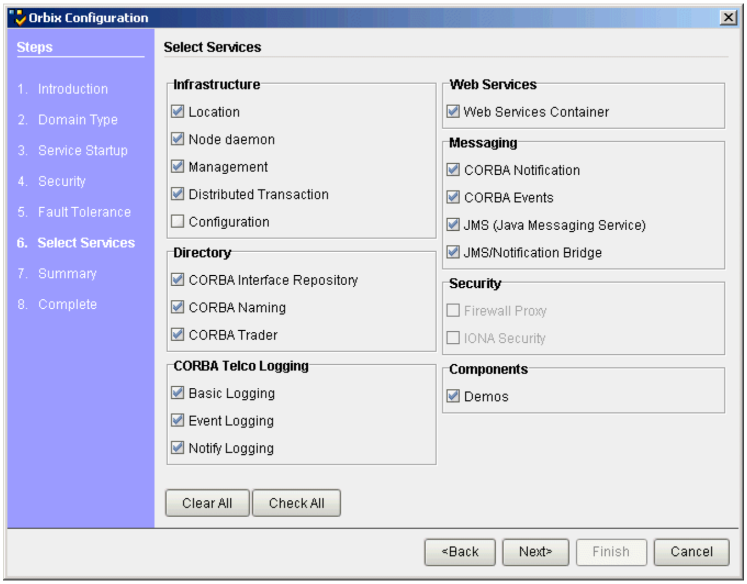

Figure 17: *Selecting Services to Deploy*

Note: If you do not check **Demos**, the demo programs included with the installation will not run properly.

If you chose to deploy all licensed services, go to step 17.

17. Select the services you wish deployed into your configuration. When you have selected the desired services, click **Next** to see a summary of the configuration options you have chosen. A screen similar to Figure 18 on page 48 should be displayed.

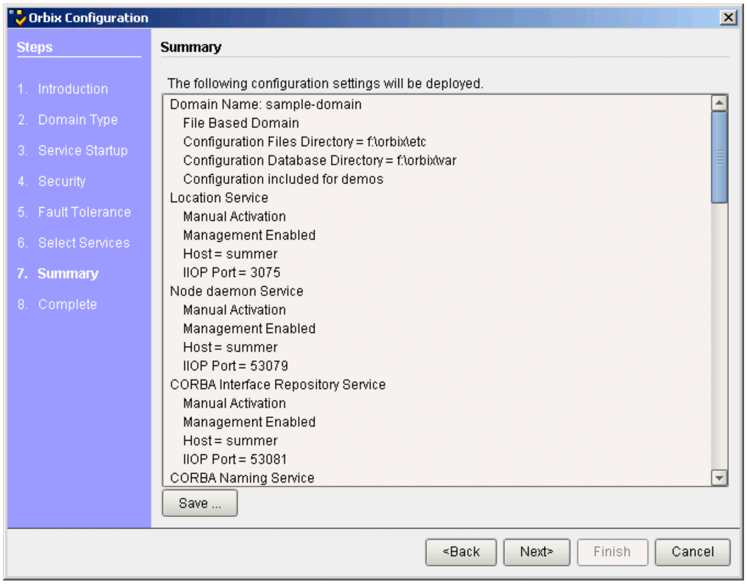

Figure 18: *Configuration Domain Summary.*

18. If you have configured replicas for this domain or have configured services to run on a different host you will need to save a domain descriptor. To save a domain descriptor for this domain, click **Save**.
19. If the summary looks correct, click **Next** to create the domain and deploy the local services.
20. When the domain is successfully created, the **Finish** button becomes available. Click it to close the tool.

Connecting a Client Machine to a Domain

Overview

You may often need to configure machines into a domain that will only run client programs. These client programs do not need any of the CORBA services running, but they do need access to the domain's configuration information. `itconfigure` enables you to connect a new machine into an existing configuration domain. The new machine will retrieve and store all of its configuration information in the configuration repository on the existing host machine.

Note: This option does not allow you to deploy additional services onto a machine. It only generates scripts allowing the current machine to join an existing configuration.

There are two approaches to connecting a client machine to an existing domain:

- "Connecting with a deployment descriptor".
- "Connecting without a deployment descriptor".

Connecting with a deployment descriptor

To connect a new machine to an existing domain using the deployment descriptor, perform the following steps:

1. Select **Connect** from the configuration tool main dialog, shown in Figure 11 on page 38.
2. If you have access to a deployment descriptor, select **Yes** in the **Target CFR Setting** dialog, shown in Figure 19.

Figure 19: *Target CFR Setting Dialog.*

3. Select the deployment descriptor using the **Load Deployment Descriptor** dialog, shown in Figure 20.

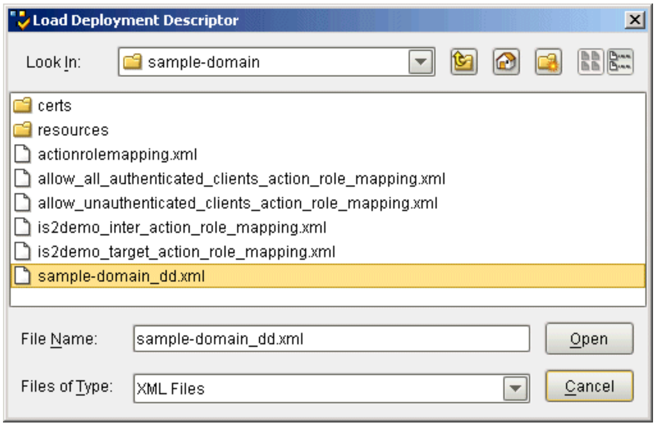

Figure 20: *Load Deployment Descriptor Dialog*

4. In the **Connect** dialog, enter the hostname and port of the CFR to which you wish to connect the new machine. Also enter a location for the configuration files, and **Node Daemon** or **Security** details, if applicable. The **Connect** dialog is shown in Figure 21.
5. Click **OK** to confirm your input and view a summary of the configuration.
6. Click **Next** to create the local files needed to connect the machine to the configuration domain and deploy the local services.
7. When the machine is successfully connected to the domain, select **Finish** to close the dialog.

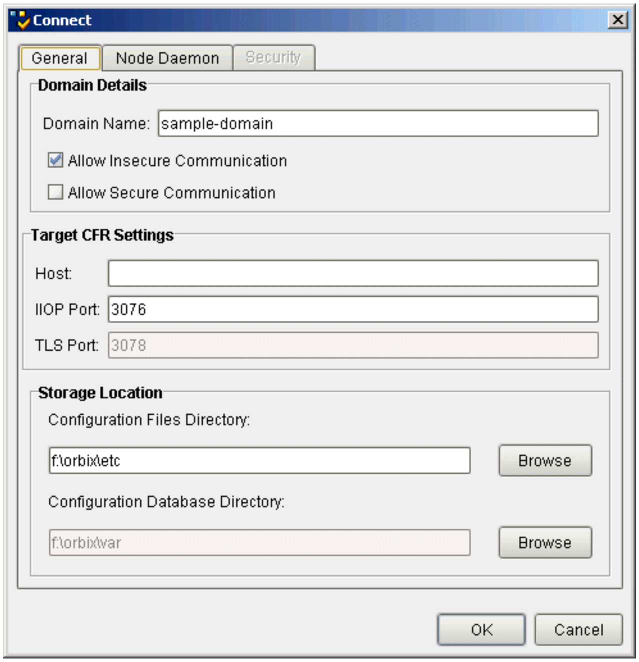

Figure 21: *Connecting to a CFR*

Connecting without a deployment descriptor

Users without access to a a deployment descriptor can also connect a new machine using the **Connect** dialog. To connect this way, perform the following steps:

1. Select **Connect** from the configuration tool main dialog, shown in Figure 11 on page 38.
2. Because you do not have access to a deployment descriptor, select **No** in the **Target CFR Setting** dialog, shown in Figure 19.
3. In the **Connect** dialog, enter the hostname and port of the CFR to which you wish to connect the new machine. Also enter a location for the configuration files, and **Node Daemon** or **Security** details, if applicable. The **Connect** dialog is shown in Figure 21.
4. Click **OK** to confirm your input and view a summary of the configuration.
5. Click **Next** to create the local files needed to connect the machine to the configuration domain and deploy the local services.
6. When the machine is successfully connected to the domain, select **Finish** to close the dialog.

Using Expert Mode

Overview

Expert mode allows the advanced user a greater amount of flexibility in creating and modifying configuration domains. It provides the ability to specify well-known addresses for Orbix services, and also enables the user to configure the services to run in using direct or indirect persistence.

Procedure

To create or modify a configuration domain using expert mode complete the following steps:

1. Select **Expert** from the main dialog, shown in Figure 11 on page 38.
2. In the **Domain Settings** screen enter a name for the domain and specify if the domain is to be file based or CFR based, shown in Figure 22.

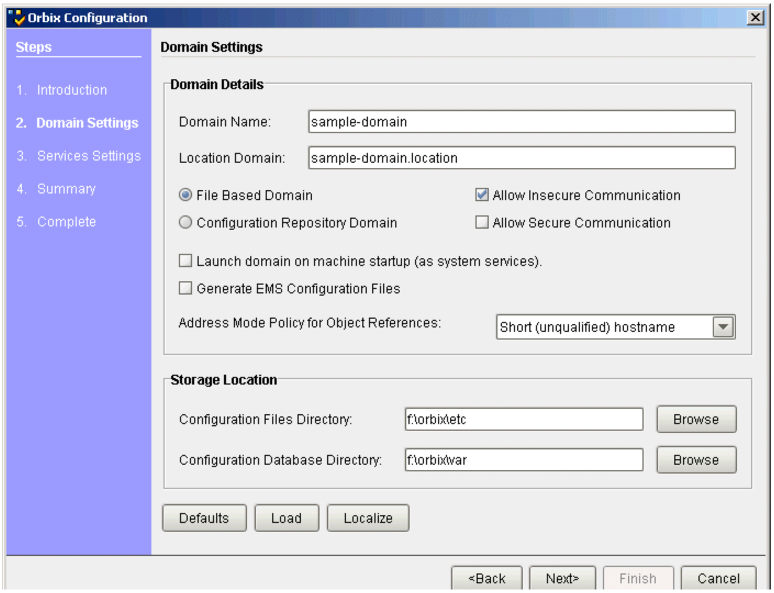

Figure 22: *Domain Settings Screen*

3. Select the level of security for your domain:
   - **Allow insecure communication** configures your domain to allows communication over insecure protocols such as HTTP.
   - **Allow secure communication** configures your system to allow secure communication using TLS or HTTPS.
4. To have the domain be started on system start-up place a check next to **Launch domain on machine startup (as system services).**
5. If you wish to integrate your domain into an IBM Tivoli Enterprise Management System, check **Generate Tivoli Agent Configuration Files.** For full details, see the *IONA Tivoli Integration Guide*.
6. Use the **Address Mode Policy for Object References** drop-down box to select how you wish to have services deployed in the domain (for example, using a hostname or IP address). Select one of the following options:

```
Short (unqualified) hostname
Fully qualified hostname
localhost (the default)
IP Addresss
127.0.0.1
```

Note: To use fully qualified hostnames, you must ensure that `itconfigure` knows your fully qualified host name. You can specify this in the **Default Settings** dialog, shown in Figure 23.

7. If you wish to store the configuration information in a non-default location, you can specify where the files are located by changing the locations specified in **Storage Location**.
8. To configure the default activation modes, replication settings, and optional services for the domain, click **Defaults**. This displays the **Default Settings** dialog, shown in Figure 23.

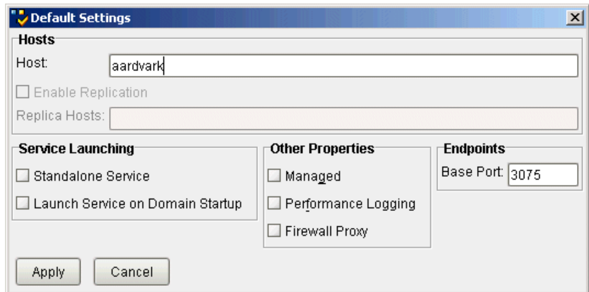

Figure 23: *Default Domain Settings*

These settings are used as the defaults for all services deployed in this domain. You can edit the settings for each service when you select which services to deploy. Click **Apply** to accept the settings.

9. Click **Next** to select the services to deploy into the domain. The **Service Settings** dialog is displayed, shown in Figure 24.
10. In the **Service Settings** dialog, check the services that you wish to deploy into the domain.

Note: If you do not check **Demos**, the demo programs included with the installation will not run properly.

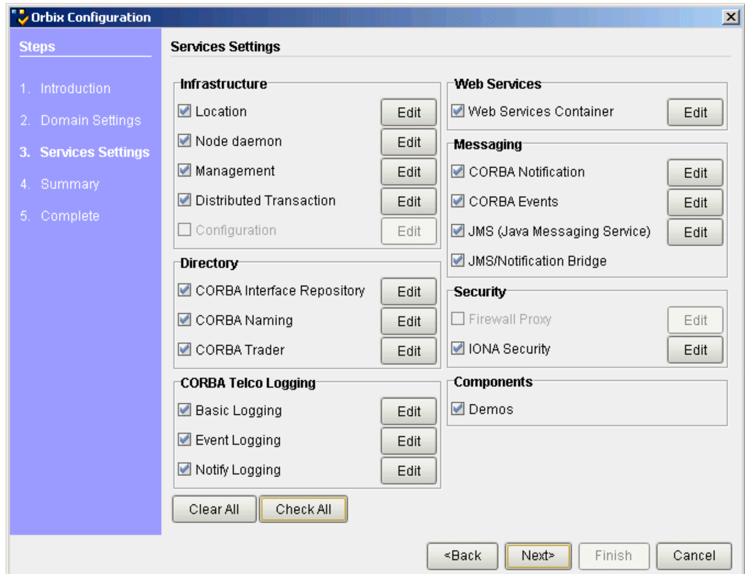

Figure 24: *Services Settings*

11. If you wish to deploy a service using non-default settings, click the **Edit** button next to the service's name. A dialog similar to Figure 25 will appear.

    This dialog allows you to configure activation modes, replication settings, and optional services for the individual service. Once you have selected the desired settings, click **OK** to return to the service selection dialog.

Note: Some options may not be available for all services.

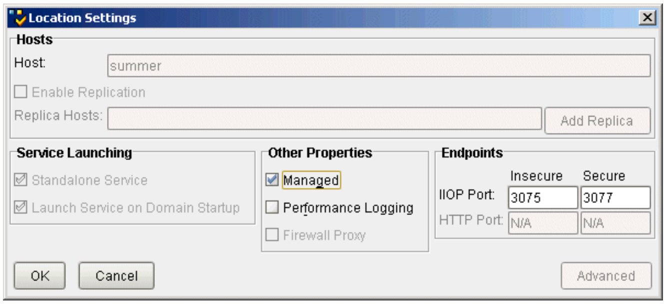

Figure 25: *Configure Advanced Service Settings*

12. After selecting and configuring the desired services, click **Next** to view a summary of the configuration options you have chosen.
13. If you have configured replicas for this domain or have configured services to be run on different hosts you will need to save a domain descriptor. To save a domain descriptor for this domain, click **Save**.
14. If the configuration summary appears correct, click **Next** to create the domain and deploy any local services.
15. Once the domain is successfully created, the **Finish** button becomes available. Click **Finish** to exit.

Localizing a Preconfigured Domain

Overview

There are times when you need to create a duplicate configuration, one that is identical except in terms of the hosts on which it runs. Some reasons for doing this are:

- Creating test and production configurations that are identical in everything but the host on which they run.
- Migrating a system from one machine to another.
- Packaging an Orbix installation with a software distribution. You can then ship a configuration template that can be run on each destination machine, with the services localized for that host, rather than the host on which the configuration was created.

If you wish to deploy a preconfigured domain, the Orbix configuration tool provides two options:

- Use expert mode and select **Localize**.
- Run `itconfigure` using the `-localize` and `-nogui` options.

Using the GUI

To use the Orbix configuration tool's GUI interface to deploy a localized domain complete the following steps:

1. Select **Expert** in the main dialog, shown in Figure 11 on page 38.
2. In the dialog shown in Figure 22 on page 53, click **Localize** (under **Storage Location**).
3. Select the preconfigured domain descriptor using the file selection dialog.
4. Enter a name for the domain and specify if the domain is to be file based or CFR based.
5. If you wish to store the configuration information in a non-default location, you can specify where the files are located by changing the locations specified in **Storage Location**.
6. Select the level of security for your domain.
   - **Allow insecure communication** configures your domain to allows communication over insecure protocols such as HTTP.

- **Allow secure communication** configures your system to allow secure communication using TLS or HTTPS.

7. To have the domain be started on system start-up place a check next to **Launch domain on machine startup (as system services).**
8. Use the **Address Mode Policy for Object References** drop-down box to select how you wish to have services deployed in the domain (for example, using a hostname or IP address).

Note: To use fully qualified hostnames, you must ensure that `itconfigure` knows your fully qualified host name. You can specify this in the **Default Settings** dialog, shown in Figure 23 on page 55.

9. To configure the default activation modes, replication settings, and optional services for the domain, click **Defaults**. A dialog similar to Figure 23 will be displayed.

   These settings will be used as the defaults for all services deployed in this domain. You will be able to edit the settings for each service when you select which services to deploy. Click **Apply** to accept the settings.
10. Click **Next** to select the services to deploy into the domain. A dialog similar to Figure 24 on page 56 will be displayed.
11. In this dialog, check the services that you wish to deploy into the domain.

Note: If you do check off **Demos**, the demo programs included with the installation will not run properly.

12. If you wish to deploy a service using non-default settings, click the **Edit** button next to the service's name. A dialog similar to Figure 25 on page 57 will appear.

    This dialog allows you to configure activation modes, replication settings, and optional services for the individual service. Once you have selected the desired settings, click **OK** to return to the service selection dialog.

Note: Some options may not be available for all services.

13. After selecting and configuring the desired services, click **Next** to view a summary of the configuration options you have chosen.
14. If you have configured replicas for this domain you will need to save a domain descriptor. To save a domain descriptor for this domain, click **Save**.
15. If the configuration summary appears correct, click **Next** to create the domain and deploy any local services.
16. When the domain is successfully created, the **Finish** button becomes available. Click **Finish** to exit.

Using the command line

If you cannot or do not want to run the GUI, you can deploy a localized domain from the command line by running

```
itconfigure -nogui -localize -load deployment-descriptor
```

Running this command replaces the name of the host used to define the original configuration with the name of the local hostname. It then deploys an exact replica of the specified domain on the local host.

You can specify other changes to the deployed domain by using other command line options.

Deploying a Domain on the Remaining Hosts

Overview

When you have designed a distributed domain, you must deploy the domain on all of the hosts that will make up the domain. To do this, you must take the deployment descriptor created when you designed the domain and migrate it to each host machine.

The Orbix configuration tool provides three options for deploying your domain on the remaining hosts:

- Use the **Deploy** option from the GUI main dialog.
- Use the **Load** option in expert mode.
- Use the `-load` option in conjunction with the `-nogui` option.

Using the Deploy option

The easiest way to deploy the local portion of a domain is to use the **Deploy** option. To use this option, perform the following steps:

1. Select **Deploy** in the main dialog, shown in Figure 11 on page 38.
2. Select the deployment descriptor from the file selection dialog.
3. A dialog similar to Figure 26 should appear. Enter the location for the configuration databases to be stored, and verify the domain name. Click **OK**.

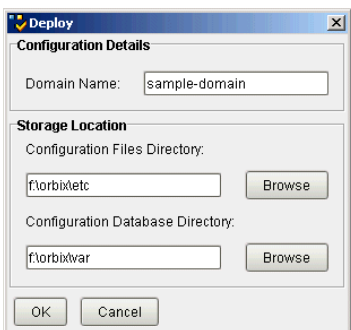

Figure 26: *Deployment details*

4. Verify that the configuration summary is accurate. If so, click **Next** to deploy the local services.
5. When the domain has successfully deployed, click **Finish** to exit.

Using Load

If you need to add services to a host in a predesigned domain you can modify the services deployed on a local host in expert mode. To do so complete the following steps:

1. Select **Expert** in the main dialog, shown in Figure 11 on page 38.
2. In the dialog shown in Figure 22 on page 53 click **Load**, located under **Storage Location**.
3. Select the domain descriptor from the file selection dialog.
4. A dialog similar to Figure 24 on page 56 will be displayed. Select the additional services to deploy on this host and click **Next**.
5. Verify that the configuration summary is accurate. If so, click **Next** to deploy the local services.
6. Once the domain is successfully deployed, the **Finish** button becomes available. Click it to close the tool.

Using the command line

If you cannot or do not want to run the GUI, you can deploy a your domain on the local host from the command line by running:

```
itconfigure -nogui -load deployment-descriptor
```

This command deploys the domain and the services specified for the local host.

Configuring a Machine with no GUI

Overview

You may need to occasionally configure and deploy an Orbix domain on a machine with no GUI capabilities, for example a server on a remote site. The configuration tool supports this by allowing you to create a configuration on one host and then apply it on another host.

Procedure

To configure and deploy a domain on a machine with no GUI capabilities complete the following steps:

1. On a machine with GUI capabilities, run the configuration GUI and select **Expert** mode.
2. Click **Defaults**. This displays a dialog box similar to the one shown in Figure 23 on page 55. In the **Host** text box, enter the name of the remote host, and then click **Apply**.
3. To design the domain, follow the steps outlined in "Using Expert Mode" on page 53.
4. When you reach the **Summary** screen, click **Save** to save the configuration deployment descriptor.
5. Copy the deployment descriptor file to the remote host.
6. Run the configuration tool on the remote host with the following command:

```
itconfigure -nogui -load domain-descriptor
```

7. Repeat this process on any other hosts on which you have configured services.

For more details on deploying on a machine without using the **Orbix Configuration** GUI, see the *Orbix Deployment Guide*.

Configuring a Multihomed Machine

Overview

You may need to occasionally configure and deploy an Orbix domain on a multihomed machine. This is a machine that has more than one IP address and corresponding hostname. A current limitation in the Java VM (Java Bug #4327220) requires that information about alternate hostnames be supplied to Java tools.

The IONA configuration tool supports this by providing a `-multihome` command line option, where the alternate hostname can be specified.

Procedure

To configure and deploy a domain on a multihomed machine complete the following steps:

1. On a multihomed machine, run the configuration tool with the added command line parameter `-multihome`, specifying the alternate hostname. For example, for a multihomed machine with primary hostname `mercury`, and an alternate hostname `hermes`, the `itconfigure` command line would be:

   ```
   itconfigure -multihome hermes
   ```

2. Select **Expert** mode, and then click **Defaults**. This displays a dialog box similar to the one Figure 23 on page 55. In the host box, enter the name of the alternate hostname. This should match the hostname specified by the multihome parameter on the command line.
3. Click **Apply**.

Because the configuration tool has been informed of the alternate hostname, deployment can then proceed as normal.

Configuring Orbix to Listen on a Fixed Port

Overview

The `simple_persistent` server from the Orbix demos, available in `$IT_PRODUCT_DIR/asp/6.1/demos/corba/standard/simple_persistent/`, is used here to demonstrate this feature. The server in this demo creates an indirect persistent POA. An indirect persistent POA exports object references containing the endpoint information (host and port) of the locator.

In order for a server to listen on a fixed port, you typically need to do two things:

1. Configure your server to listen on a well known address.
2. Configure your server to export object references that contain this direct endpoint information.

Orbix provides IONA proprietary POA policies that allow you to instruct a POA to listen on a fixed port. These polices are called `WellKnownAddressing` policy and `DirectPersistent` policy. You can set these polices programmatically when you create your POA, and you can also set them via configuration. For detailed information on how to set these policies programmatically, see the *CORBA Programmers Guide*.

Configuration

It is possible to set these POA policies via configuration. If you want to run a server using direct persistence (and well known addressing), you must add the following entries 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 is now direct persistent containing a well known address (IIOP port 5555). If your POA name is different, then the configuration variables need to be modified. The following schema is used:

```
poa:<FQPN>:direct_persistent = boolean;
poa:<FQPN>:well_known_address = <address_prefix>;
<address_prefix>:iiop:port = long;
```

`<FQPN>` is the Fully Qualified POA Name. The Orbix configuration scheme introduces a restriction in how you name your POA. The name can only contain printable characters; white space and null characters are not permitted in Orbix configuration variables.

`<address_prefix>` is simply the string that gets passed to the well known addressing POA policy. You specify the actual port used via the variable `<address_prefix>:iiop:port`.

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

Replicating Services in a Domain

Overview

You can use the Orbix configuration tool to configure a machine to use replicas of an existing configuration repository and location daemon. A machine configured to host replicas can also host services as part of an existing configuration domain.

Note: To configure a machine to host replicas, you must have already specified that the domain include replicas when you created it (see page 46).

Replicating a secure file-based domain must be performed manually (see "Replicating Services in a File-Based Domain" on page 141).

Deploying a CFR-based replica

To deploy a CFR-based replica, perform the following steps:

1. Copy the generated deployment descriptor from the host machine to the replica that you wish to configure. The deployment descriptor will have the name `domain_name_dd.xml`. For example, the domain descriptor for a domain named `Cuculain` will be `Cuculain_dd.xml`.
2. Run the Orbix configuration tool. From the main screen, click **Deploy**.
3. In the **Load Deployment Descriptor** dialog, select the domain descriptor that you wish to replicate. Click **Open**.
4. Provide information that services require about this machine, including storage locations for their configuration files and service databases and the ports for the local `node_daemon`, if one is in use. Click **Next**.
5. You will be shown a summary of what is being deployed. If the summary is correct, click **Next** to deploy the configuration.
6. When the replica is successfully deployed, click **Finish**.

Starting and Stopping Orbix Services

Overview

The configuration tool automatically generates start and stop scripts, which let you manually activate and deactivate all services deployed on the configured host. You can also manually start and stop services individually (see "Starting Orbix Services Manually" on page 215).

Starting Orbix services

You can start all Orbix services that are deployed on this machine with the following command:

`config-dir/bin/start_domain-name_services`

Stopping Orbix services

You can stop Orbix services manually with the following command:

`config-dir/bin/stop_domain-name_services`

Setting an Orbix environment

In order to use access any of Orbix utilities in a given domain, you will need to set the system environment properly. To set your environment to recognize an Orbix configuration domain use the following command:

`config-dir/bin/domain-name_env`

Setting Java ORB Classes

Overview

In order to run Java applications, Orbix must use its own ORB classes instead of Sun ORB classes. To ensure that Orbix finds the correct classes, perform one of these actions:

- Create an `iona.properties` file.
- Use Java system properties when invoking the Java interpreter.

Using an iona.properties file

Create the `iona.properties` file in the *`JAVA_HOME`*`/jre/lib` directory. This file must contain the following settings:

```
org.omg.CORBA.ORBCLASS=com.iona.corba.art.artimpl.ORBImpl
org.omg.CORBA.ORBSingletonClass=com.iona.corba.art.artimpl.ORBSi
   ngleton
```

Using Java system properties

Invoke the Java interpreter with the `-D` options as follows:

```
java -Dorg.omg.CORBA.ORBCLASS=com.iona.corba.art.artimpl.ORBImpl
-Dorg.omg.CORBA.ORBSingletonClass=com.iona.corba.art.artimpl.ORB
   Singleton app-name
```

Running Sample Applications

Overview

If you install the development environment, you can run the sample applications. These are located in:

install-dir/asp/*version*/demos

Each sample application contains a makefile to compile and link the client and server programs.

> **Note:** You must specify that the demo programs' configuration information is to be included when you run the configuration tool.

The `typetest` application used in this section moves common data types between a client and a server.

C++

To run the `typetest` demo, enter the following:

UNIX

```
cd install-dir/asp/version/demos/corba/orb/typetest
make cxx
cd cxx_server
./server &
cd ../cxx_client
client
```

Windows

```
cd install-dir\asp\version\demos\corba\orb\typetest
nmake cxx
cd cxx_server
server
cd ..\cxx_client
client
```

> **Note:** After running each sample application, you must manually kill the server process.

Java

To run the `typetest` demo, enter the following:

UNIX

```
cd install-dir/asp/version/demos/corba/orb/typetest
make java
cd java_server
java -classpath ./java/classes:$CLASSPATH typetest.Server
cd ../java_client
java -classpath ./java/classes:$CLASSPATH typetest.Client
```

Windows

```
cd install-dir\asp\version\demos\corba\orb\typetest
build all
cd java_server
java -classpath .\java\classes;"%CLASSPATH%" typetest.Server
cd ..\java_client
java -classpath .\java\classes;"%CLASSPATH%" typetest.Client
```

Note: After running each sample application, you must manually kill the server process.

Part II

Managing an Orbix Environment

In this part

This part contains the following chapters:

Configuring Orbix Applications

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.

The 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.

In this chapter

This chapter contains the following sections:

How an ORB Gets its Configuration

Every ORB runs within a configuration domain, which contains variable settings that determine the ORB's runtime behavior. Figure 28 summarizes how an initializing ORB obtains its configuration information in a repository-based system, where services are distributed among various hosts.

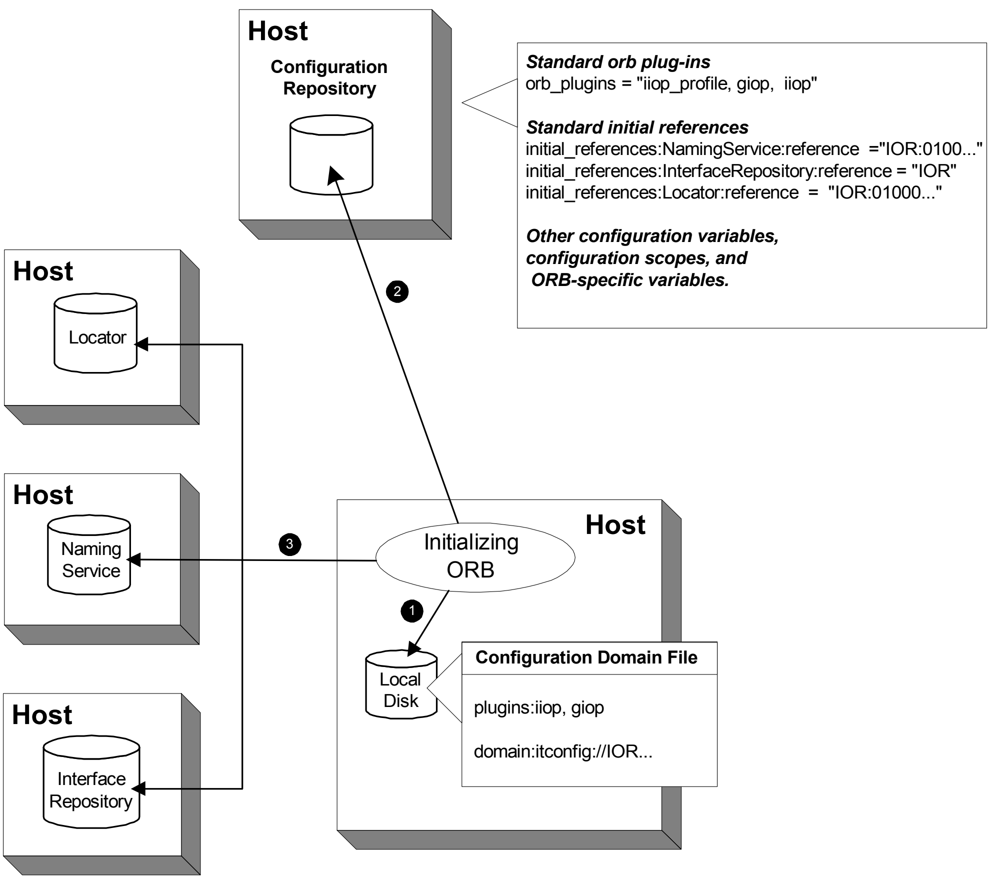

Figure 27: *How an Orbix Application Obtains its Configurations*

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/iona
   ```

   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 397 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 28 shows how a configuration domain might be organized into several scopes:

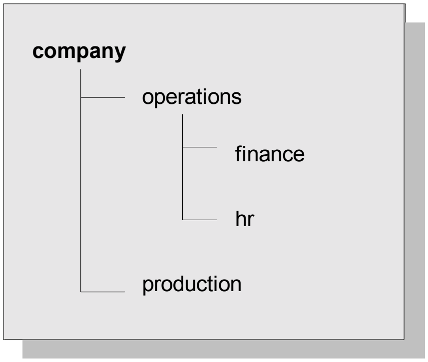

Figure 28: *Hierarchy of Configuration Scopes*

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 28 on page 80:

```
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 28 on page 80:

Example 1: *A 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"
        }
4       hr
        {
           # hr-specific settings

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

     } # close operations scope
```

Example 1: *A File-Based Configuration*

```
5     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`.

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. The following describes how to set the buffer size in the C++ and Java CORBA ORBs.

C++ configuration value

`policies:{iiop¦iiop_tls}:buffer_sizes_policy:default_buffer_size`

This value 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 values

`policies:{iiop¦iiop_tls}:buffer_sizes_policy:default_buffer_size`

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

`policies:{iiop¦iiop_tls}: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 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). For a CORBA ORB to be considered compliant with the OMG GIOP 1.1 specification, the ORB implementation must support fragmentation. Some CORBA ORB implementations do not support fragmentation but claim GIOP 1.1 compliance. Orbix ORBs support fragmentation and are fully compliant with the GIOP 1.1 specification.

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: • The `itconfigure`-generated `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`

Table 1: *Configuration Domain Management Tasks*

Perform this task...	By running...
View variable	`itadmin variable show`
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 5

Managing Persistent CORBA Servers

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

CORBA servers that export persistent objects must be registered with a locator daemon through 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 it transparently. A configured location domain typically requires very little outside management. 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.

In this chapter

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

Registering Persistent Servers

A persistent 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.

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.

The following sections show how to use `itadmin` commands to perform these tasks. These commands can be entered either at the command line, or through a script.

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 111.

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 133.

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 simply 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 Orbix, see Chapter 3 on page 33.

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: *itadmin 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 16 on page 247.

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: *itadmin 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: *itadmin 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 130).

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>;
<address_prefix>:iiop:port=<LONG>;
```

<*FQPN*> is the fully qualified POA name. Obviously this introduces the restriction that your POA name can only contain printable characters, and may not contain white space.

<*address_prefix*> is the string that gets passed to the well-known addressing POA policy. Specify the actual port used using the variable <*address_prefix*>`: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:<*service_name*>**:direct_persistence** Indicates whether the service uses direct or indirect persistence. The default value is FALSE, which indicates indirect persistence.

plugins:<*service_name*>**: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:<*service_reference_string*>**:reference** specifies the IOR of the service.

If the service is clustered, `plugins:`<*service_name*>`: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:`<*service_name*>`:direct_persistence` to `TRUE` within the service's configuration scope.
3. Within the same configuration scope, set `plugins:`<*service_name*>`: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:`<*service_reference_string*>`:reference` with the IOR returned in step **4**.
6. Restart the service.

CHAPTER 6

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 or partitioned by an excavator that accidentally slices through phone lines. Operating systems can encounter fatal errors and fail to reboot automatically. 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 so it can 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. Furthermore, running all web servers on the same host makes the entire web site dependent on that machine; if it fails, it brings down with it the entire web 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 12 for information on additional features that are designed to enhance scalability and performance (for example, Java new I/O and shared memory).

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
```

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 the 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 the 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 their 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. The 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`. As described elsewhere (see "Managing Object Availability" on page 8), Orbix locates persistent objects through the locator daemon, which maintains their addresses on a physical server. 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.

The 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, the 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 134.
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.

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

In order to ensure failover within a replicated POA with direct persistence, the following requirements apply:

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

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

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.

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 that 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 changing replicas within a replicated server.

Sample code

Examples are based on several demos in the distribution's `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 serves 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 come up and connect to the object get and display the server name, thus 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:\iona\asp\version\demos\enterprise\clustering
$ nmake
```

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

```
$ itadmin
% process create \
       -pathname
   /opt/iona/asp/version/demos/enterprise/clustering/cxx_se
   rver/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 via the -args argument. These arguments 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 2; 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, 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 the system. The following commands add a server replica to the set already registered in the `clustering` demo:

Example 2:

```
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 lets you dynamically change the load-balancing algorithm used for a replicated POA. 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.

Replicating Domain Services

Overview

Various IONA services within a configuration domain can be replicated (for example, the locator daemon, naming service, and configuration repository). Clients that use these services are automatically routed to the first server available. If a server fails, clients are transparently rerouted to another one.

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 start or be activated on demand.
- `itadmin` commands that read the implementation repository (such as poa list, process show, and so on) always work.
- Clients can always obtain object references from the naming service.

Master and slave hosts

Replicated services are configured as follows:

- One service host is designated as the master. This machine contains the only writable copy of the service data—the implementation repository for the locator daemon, the naming service graph for the naming service.
- Other hosts of the same service are designated as slaves. Each slave contains a local read-only copy of the service data.

A client that tries to update service data—for example, by calling `NameContext::bind()`—is automatically directed to the service's master. The master, in turn, propagates these changes to its slaves. A client that tries to read service data—for example, by calling `NameContext::resolve()`—is directed to one of the slaves.

Note: All replicas within a replicated service must run on the same operating system.

Because instances of a replicated service run on different host machines, an application is insulated from system failures. Only one host needs to run to ensure that applications can obtain service-related data.

Deploying CFR and file-based replicas

For details of how to use the Orbix configuration tool to replicate services in a configuration repository domain, see "Replicating Services in a Domain" on page 67. Manually replicating services in a file-based domain is described in the next section.

Replicating Services in a File-Based Domain

Overview

You can manually replicate Orbix services in a file-based domain by editing your configuration files and using the command line. This section includes the following:

- "Recommendations".
- "Replication IORs".
- "Replication configuration variables".
- "Replicating in file-based domain".

Recommendations

Before you begin, check the following:

- All configuration files, database files, log files should be stored on separate disks, and not stored on a shared disk or NFS.
- The services are generated in the configuration file with a hostname in lowercase. This does not cause problems if the hostnames are in uppercase. The security service is an exception because its scoped name is case sensitive (`iona_services.security.blackadder` is different to `iona_services.security.Blackadder`).
- When preparing IORs for services, you should prepare on the machine in question. Do not generate a slave IOR on a master machine. In a best case scenario, this will not cause problems; however, if there are any slave configuration issues, they may not be apparent on the master machine during preparation.

Replication IORs

A replicated Orbix service produces two IORs when prepared: one prefixed with the `IT_Single` string and one that is not (for example, `IT_SingleLocator` and `IT_Locator`).

The `IT_Single`<*service_name*> IORs are used to allow individual addressing of one replica in the cluster, either master or slave. A call to an `IT_Single`<*service_name*> IOR never results in a load-balanced request. It goes to one replica, and one replica only.

Replication configuration variables

There are three important types of configuration variables for replicated services that use the replication IORs:

The initial_references variable This variable should contain the non-single IOR (for example, `IT_Locator`) produced by any instance of that service. Each service replica in a cluster should always produce the same value for this IOR.

The non-single IOR should contain addresses of all replicas in the cluster (for example, in the case of direct persistent services like the locator). Alternatively, it will point to the locator cluster itself (for example, in the case of indirect persistent services like naming).

The IT_<*Service*>Replicas variable This variable is a list of all the `IT_Single`<*service_name*> IORs produced by all the replicas in a service cluster (for example, the `IT_LocatorReplicas` variable).

This list is used by `itadmin` to enable administration of a particular replica. The format of each entry in the list is *replica_name*=`IOR`, where *replica_name* is a simple string used to identify the replica. This is displayed, for example, when you use the `itadmin locator list_servers` command.

The IT_Master<*Service*> variable This variable identifies the master replica in the cluster (for example, the `IT_MasterLocator` variable). This is used by slaves to contact the master for delegation purposes. The IOR comes from one place only—the `IT_Single`<*service_name*> IOR produced by the master during its prepare stage. To generate these IORs, the configuration file must first be edited to include slave and master configuration.

Replicating in file-based domain

In the example that follows, assume that you have a master machine called `blackadder` (IP address: `10.139.54.30`) and a slave machine called `baldrick` (IP address: `10.139.54.38`).

To replicate a file-based domain, perform the following steps:

1. Use the Orbix configuration tool (`itconfigure`) to create a normal insecure master domain selecting the locator, node daemon and naming service. In this example, the domain is called `fileCluster`.
2. After creating a file-based domain on the master machine, open the `fileCluster.cfg` file, and ensure that the following entries are present in the master `iona_services.locator.blackadder` scope:

```
is_master = "true";
plugins:pss_db:envs:it_locator:push_all_updates = "true";
plugins:locator_cluster:iiop:addr_list = ["blackadder:3075",
   "+10.139.54.30:3075"];
```

And create an `iona_services.locator.baldrick` slave scope as:

```
baldrick
{
is_master = "false";
orb_plugins = ["local_log_stream", "iiop_profile", "giop",
   "iiop", "ots"];
binding:server_binding_list = ["OTS", ""];
pss_plugins = ["locator_store", "named_key_store"];
plugins:pss_db:envs:it_locator:replication_model = "push";
plugins:pss_db:envs:it_locator:db_home =
   "<install-dir>\var\fileCluster\dbs\locator";
plgins:pss_db:envs:it_locator_priv:db_home =
   "<install-dir>\var/fileCluster\dbs\locator_priv";
plugins:it_mgmt:managed_server_id:name =
   "iona_services.locator.baldrick";
policies:iiop:server_address_mode_policy:local_hostname =
   "baldrick";
plugins:locator:iiop:port = "3075";
plugins:locator:iiop:host = "baldrick";
plugins:locator_cluster:iiop:addr_list = ["+blackadder:3075",
"+10.139.54.30:3075"];
event_log:filters = ["*=*"];
plugins:local_log_stream:filename =
   "<install-dir>\var\fileCluster\logs\locator.log";
};
```

The slave scope must contain all of these variables, except the `plugins:local_log_stream:filename` variable. This is optional, but is advised to enable logging for the slave locator for diagnosis, if any exceptions occur.

3. Source the `fileCluster_env` script to set up your environment. You must do this on the slave by copying the script to the slave machine, and modifying any host-specific information such as paths.
4. Copy the `fileCluster.cfg` file to the slave before preparing the locator. Make sure the `<install-dir>/var/fileCluster/dbs` and`<install-dir>/var/fileCluster/logs` directories are created on the slave machine. The `dbs` directory is created for you, but `logs` is not.
5. Prepare the locators on their respective machines, as follows:

```
<install-dir>/asp/6.1/bin/itlocator -ORBdomain_name fileCluster
   -ORBname iona_services.locator.blackadder prepare

<install-dir>/asp/6.1/bin/itlocator -ORBdomain_name fileCluster
   -ORBname iona_services.locator.baldrick prepare
```

   As explained in"Replication IORs" on page 141, each of these commands will generate two IORs (`IT_Locator` and `IT_SingleLocator`).
6. Update the values of the following variables in both configuration files, as explained in "Replication configuration variables" on page 141:

```
IT_LocatorReplicas
initial_references:IT_Locator:reference
initial_references:IT_MasterLocator:reference
```

   The first IOR, `IT_Locator` contains the extra component for the slave locator `host:port`. If you run the `iordump` utility on this IOR, you see a tagged component called `TAG_ALTERNATE_IIOP_ADDRESS`. This details the `host:port` of the slave locator, and as a result the `IT_Locator` generated for both the master and the slave are the same. Substitute this IOR for the current IOR in the `initial_references:IT_Locator:reference`.
7. Start the master locator on the master machine, and start the slave locator on the slave machine.
8. You cannot replicate the node daemon. When it is started on the slave machine, it automatically registers itself with the slave locator. Create a `iona_services.naming.baldrick` scope for the node daemon on the `baldrick` slave host. The contents of this scope can be the same as the

master, provided that you change all host-specific values for the slave machine, such as paths. Each node daemon is an independent instance.

Note: Before preparing the node daemons, the `IT_LocatorReplicas` configuration variable must be updated with each locator in the cluster.

9. Prepare the node daemons, for example:

```
<install-dir>/asp/6.1/bin/itnode_daemon -ORBdomain_name
   fileCluster -ORBname iona_services.node_daemon.blackadder
   prepare

<install-dir>/asp/6.1/bin/itnode_daemon -ORBdomain_name
   fileCluster -ORBname iona_services.node_daemon.baldrick
   prepare
```

   After preparing the slave node daemon, modify the `LOCAL_NODE_DAEMON_REFERENCE` in the slave's configuration file only. This avoids problems later when preparing the naming service.

10. Start the node daemon on the master machine, and start the node daemon on the slave machine.
11. On the master machine, add the following configuration to the `iona_services.naming.blackadder` scope:

```
plugins:naming:is_master = "true";
plugins:pss_db:envs:it_naming_store:push_all_updates = "true";
plugins:naming_cluster:iiop:addr_list = ["blackadder:0",
   "+10.139.54.30:0"];
```

And update the `iona_services.naming` scope to include the new scope for the slave machine:

```
baldrick
{
plugins:naming:is_master = "false";
orb_plugins = ["local_log_stream", "iiop_profile", "giop",
   "iiop"];
plugins:pss_db:envs:it_naming_store:db_home =
   "<install-dir>\var\fileCluster\dbs\naming";
policies:iiop:server_address_mode_policy:local_hostname =
   "baldrick";
plugins:naming:iiop:port = "0";
plugins:naming:iiop:host = "baldrick";
plugins:naming_cluster:iiop:addr_list = ["+blackadder:0",
   "+10.139.54.30:0"];
event_log:filters = ["*=*"];
plugins:local_log_stream:filename =
   "<install-dir>\var\fileCluster\logs\naming.log";
plugins:orb:is_managed = "false";
plugins:naming:is_managed = "false";
plugins:naming:direct_persistence = "false";
plugins:pss_db:envs:it_naming_store:replication_model = "push";
plugins:it_mgmt:managed_server_id:name =
   "iona_services.naming.baldrick";
};
```

12. Prepare the master naming service as follows:

```
<install-dir>/asp/6.1/bin/itnaming –ORBdomain_name fileCluster –
   ORBname iona_services.locator.blackadder prepare
```

    Update the IOR values in the configuration file.

13. The preparation of the naming service would have registered POAs in the locator. These need to be modified for all the naming service to be replicated.

    You can view registered POAs using the `itadmin poa list` command as follows:

```
<install-dir>\asp\6.1\bin>itadmin poa list
IT_ObjectGroup
IT_ObjectGroupFactory
IT_NamingServiceAdmin_iona_services.naming.blackadder
IT_NamingContextExt
```

You can modify these POAs using the following commands:

```
itadmin poa modify -allowdynreplicas  yes
   IT_ObjectGroup
itadmin poa modify -allowdynreplicas  yes
   IT_ObjectGroupFactory
itadmin poa modify -allowdynreplicas  yes
   IT_MasterNamingContextExt
itadmin poa modify -allowdynreplicas  yes
   IT_MasterObjectGroupFactory
itadmin poa modify -allowdynreplicas  yes
   IT_MasterObjectGroup
itadmin poa modify -allowdynreplicas  yes
   IT_NamingContextExt
```

14. Prepare the naming service slave as follows:

```
<install-dir>/asp/6.1/bin/itnaming –ORBdomain_name fileCluster –
   ORBname iona_services.locator.baldrick prepare
```

And update the IORs accordingly. If you experience any problems with this step, see “Incorrect node daemon” on page 148.

15. Launch both naming services. To test if the services have been replicated successfully, shutdown the services on the master, and run a some `itadmin` commands from your slave machine. These commands should all still work. As always, you will be unable to write to the slave services because these are read-only.

Troubleshooting Replication Errors

Overview

Running the naming service in prepare mode might generate some errors because POAs might be registered under different ORBs in the locator. This can be rectified using the `itadmin` command. This section explains how to troubleshoot the following errors:

- "Incorrect node daemon".
- "Replicas have not been run".

Incorrect node daemon

The following is an example error that you might encounter:

```
<install-dir>\asp\6.1\bin\itnaming -ORBdomain_name fileCluster
   -ORBconfig_domains_dir
   T:\user\margo\cluster\fileCluster_ant\domains -ORBname
   iona_services.naming.blackadder prepare
ERROR: Process ('') does not exist
ERROR: exception occurred during prepare:
   IDL:omg.org/CORBA/OBJ_ADAPTER:1.0: minor = 0x4954050D
   (IT_POA:PROCESS_NOT_EXIST), completion status = NO
<install-dir>\asp\6.1\bin\itadmin -ORBdomain_name fileCluster
   -ORBconfig_domains_dir
   T:\user\margo\cluster\fileCluster_ant\domains process list
iona_services.naming.blackadder
<install-dir>\asp\6.1\bin\itadmin -ORBdomain_name fileCluster
   -ORBconfig_domains_dir
   T:\user\margo\cluster\fileCluster_ant\domains process show
   iona_services.naming.blackadder
Process Name: iona_services.naming.blackadder
Active: no
Description:
Startup Mode: disable
Usually Monitored By: iona_services.node_daemon.baldrick
Can not be started on-demand
```

In this example, the `blackadder` master naming service is being monitored by an incorrect `baldrick` slave node daemon. The following `itadmin` command fixes this behavior:

```
<install-dir>\asp\6.1\bin\itadmin -ORBdomain_name fileCluster
    -ORBconfig_domains_dir
    T:\user\margo\cluster\fileCluster_ant\domains process modify
    -node_daemon iona_services.node_daemon.blackadder -pathname
    <install-dir>\asp\6.1\bin\itnaming
    iona_services.naming.blackadder
```

Replicas have not been run

The following is another example error that might appear:

```
>itnaming -ORBdomain_name fileCluster -ORBconfig_domains_dir
    T:\user\margo\cluster\fileCluster_ant\domains -ORBname
    iona_services.naming.baldrick prepare
ERROR: could not register persistent POA IT_NamingContextExt: POA
    is not registered with ORB iona_services.naming.baldrick in
    the implementation repository
ERROR: exception occurred during prepare:
    IDL:omg.org/CORBA/OBJ_ADAPTER:1.0: minor = 0x49540509
    (IT_POA:POA_CONFIGURED_IN_DIFFERENT_ORB), completion status =
    NO
```

This error implies allow dynamic replicas have not been run. This can be solved by running the following command:

```
itadmin poa modify -allowdynreplicas  yes IT_NamingContextExt
```

You can view all registered POAs using the `itadmin poa list` command:

```
>itadmin poa list
IT_ObjectGroup
IT_ObjectGroupFactory
IT_NamingServiceAdmin_iona_services.naming.baldrick
IT_NamingContextExt
```

You can view a single registered POA using the `itadmin poa show` command:

```
>itadmin poa show IT_ ObjectGroup
  FQPN: IT_ObjectGroup
  Active: no
  Lifespan: persistent
  ORB Names:
    iona_services.naming.baldrick
  Allow Replicas outside this list: no
  Load Balancing Algorithm:  <NONE>
  Allow Dynamic Registration: yes
  Parent FQPN: <NONE>
  Children FQPN: <NONE>
```

You can then remove the POA using the `itadmin poa remove` command:

```
C:\orbix\asp\6.1\bin>itadmin poa remove IT_ ObjectGroup
```

If POAs seem to be completely corrupt, prepare the services again to register new POAs.

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 279. 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 29.

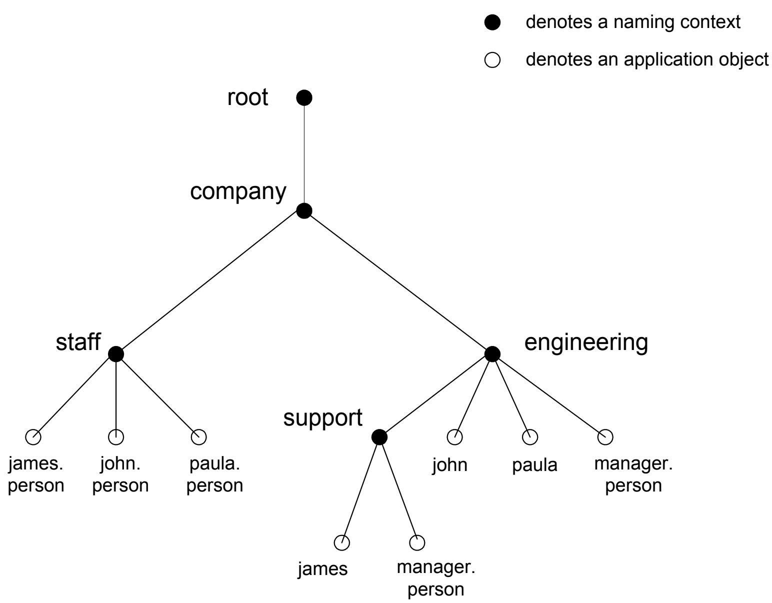

Figure 29: *Naming Context Graph*

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 29, 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 29 on page 157, 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 29 on page 157.

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 163).
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 284 describes the `itadmin` commands that you can use to create and administer object groups.

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 293.

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 `typedef` 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)
```

Orbix Firewall Proxy Service

The 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:

Firewall Proxy Service Functionality

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.

Using the Firewall Proxy Service in a Deployed Environment

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 Iona's IS2 security infastructure or any other systems that use TLS.
- The J2EE portion of your systems cannot be hidden behind a proxy.

Managing CORBA Service Databases

This chapter explains how to manage databases that store persistent information 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, and naming service). By default, these Orbix services use an embedded database management system, Berkeley DB. 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:shmiop` 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 305.

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 305.

WARNING: Deleting old transaction log files can make recovery from a catastrophic failure impossible. See "Database Recovery for Orbix Services" on page 186.

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 IONA 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 186) 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 188. 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 187).
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/.

Setting Up 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.

Note: For information on logging Orbix Windows NT Services, refer to "Logging Orbix Windows Services" on page 374.

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 194). The `severity` field indicates the severity levels that are logged by that subsystem (see Table 7 on page 195).

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
```

Logging Subsystem

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_NODE_DAEMON	Node daemon.
IT_CONFIG_REP	Configuration repository.
IT_CORE	Core ORB.
IT_GIOP	General Inter-Orb Protocol (transport layer).
IT_IFR	Interface repository.
IT_IIOP	Internet inter-orb protocol (transport layer).
IT_IIOP_PROFILE	Internet inter-orb protocol profile (transport layer).
IT_LOCATOR	Server locator daemon.
IT_NAMING	Naming service.
IT_NOTIFICATION	Event service.
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_TS	Threading/synchronization package.

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 the 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 the Orbix encounters an error. The 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 the 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, the 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. 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`.

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",];
```

Configuring Advanced Features

This chapter explains some how to configure advanced features such as Java new I/O, shared memory, and bidirectional GIOP.

In this chapter

This chapter includes the following topics

Configuring Java NIO

Overview

Java's new I/O (NIO) provides enhanced connection scalability. It enables you to manage more connections with fewer resources (specifically, fewer threads). This section includes the following:

- "ATLI2/Java NIO".
- "Requirements".
- "Enabling Java NIO".
- "Further information".

ATLI2/Java NIO

IONA's current transport layer implementation is called the Abstract Transport Layer Interface, version 2 (ATLI2). Orbix offers an ATLI2 implementation based on Java NIO. The default ATLI2 plugin is based on Java classic I/O (CIO).

In addition to allowing more connections to be managed with fewer threads, ATLI2/Java NIO also performs better than ATLI2/Java CIO in the presence of many incoming connections.

Requirements

To use ATLI2/Java NIO, you must have JDK version 1.4.x installed.

> **Note:** Applications that use either Transport Layer Security (TLS) or Endpoint Granularity Multicast Inter-ORB Protocol (EGMIOP) must use the default Java CIO. Java NIO does not support Java Secure Socket Extensions (JSSE) or multicast sockets.

Enabling Java NIO

To enable Java NIO, change the `plugins:atli2_ip:ClassName` configuration variable setting from the following:

```
plugins:atli2_ip:ClassName
=com.iona.corba.atli2.ip.cio.ORBPlugInImpl
```

to the following:

```
plugins:atli2_ip:ClassName
=com.iona.corba.atli2.ip.nio.ORBPlugInImpl
```

Further information

For more information about Java NIO, see the Sun web site:

http://java.sun.com/j2se/1.4.1/docs/guide/nio/

Configuring Shared Memory

Overview

Shared memory is an inter-process communication mechanism, available on certain operating systems. It provides an efficient means of passing data between programs that are executing on the same host. One process creates a memory portion that other processes can access.

When the client and server are located on the same host, using shared memory to communicate is usually faster than using network calls. This section includes the following:

- "Shared memory segment size".
- "Enabling shared memory".
- "Shared memory logging".
- "Shared memory segment size".

Platform availability

The shared memory plug-in is available for C++ ORBs on the following platforms:

- Solaris
- HP-UX
- Windows

> **Note:** Java ORBs can not read their `orb_plugins` list if it specifies the shared memory plug-in. For this reason, a shared memory configuration domain should not be shared between C++ and Java ORBs.

Enabling shared memory

Orbix provides the `shmiop` transport plugin, which uses shared memory as its underlying communication mechanism.

To use shared memory with Orbix, perform the following steps:

1. Modify the `orb_plugins` list in your configuration to include the SHMIOP plugin. For example:

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

2. On the client side, add the shmiop plugin to the client_binding_list, for example:

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

When the client_binding_list is set, Orbix first attempts to bind to the server using the faster shared memory transport. If this is unsuccessful—for example, if the server is not on the same host as the client—Orbix then uses the standard IIOP transport as normal.

Shared memory logging

To enable logging output from the shared memory plugin, turn on the log stream, and add the following filter in your configuration:

```
event_log:filters = ["IT_ATLI2_SHM=*"];
```

IONA's transport layer implementation is referred to as the Abstract Transport Layer Interface, version 2 (ATLI2).

Shared memory segment size

You can configure the size of the shared memory segment created (for example, in the call to mmap on Solaris). You can set this using the following configuration variable:

```
plugin:atli2_shm:shared_memory_size
```

The default value is 8*1024*1024. This size should be larger than the largest data payload passed between a client and server. If the setting is too small, the shared memory transport will run out of memory, and will be unable to marshal the data. If there is danger of this occurring, add GIOP+IIOP to your client_binding_list setting. This enables the ORB to use the normal network transport if a large payload can not make it through shared memory.

Further information

For information on additional shared memory configuration variables, see the plugin:atli2_shm and policies:shmiop namespaces in the *Configuration Reference*. The default configuration settings are sufficient for most cases.

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 205.
- "Migration and Interoperability Issues" on page 208.

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 208 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 Appendix E on page 405.

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 6.1 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 205. 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 6.1 includes support for interoperability with Orbix 3.x (Generation 3). This enables an Orbix 6.1 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 6.1 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 206.

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 6.1 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 6.1 bidirectional invocations may be made after the connection is reaped by Active Connective Management and re-established.

The Orbix 6.1 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 6.1 or Orbix 3.x.

Orbix 6.1 restrictions

Orbix 6.1 includes the following restrictions:

- Orbix 6.1 support for Orbix 3 bidirectional GIOP is asymmetric. An Orbix 6.1 server can invoke on a Orbix 3 callback reference using bidirectional GIOP. However, an Orbix 6.1 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.

Part III
Reference

In this part

This part contains the following chapters:

Starting Orbix Services

This chapter describes commands that start Orbix services. For information on starting Orbix services as Windows NT services, see *Appendix A on page 365**.*

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: *itadmin 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 14

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 the 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 232.

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.

The 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 317.-

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.

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:

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 213).

Configuration Repository

Overview

The following commands enable you to manage the configuration repository (CFR):

Table 10:

`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...723a312e300000000000001000000
   00001a00"

initial_references:POACurrent:plugin = "poa"

initial_references:NameService:reference =
   "IOR:010000002f00000049444c3a696f6e61...2e6362f49545f4e616d69
   6e6060000000010000003500"

initial_references:DynAnyFactory:plugin = "it_dynany"

initial_references:ConfigRepository:reference =
   "IOR:010000002000000049444c3a495000002000...000060000000010000
   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 11:

`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 12:

`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 13:

`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 89.

`-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 229 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:0043324342352342359333..."
   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 229 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
```

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 14:

`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 15:

`named_key create`	Creates an association between a specified well-known object key and a specified object reference.

Table 15:

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 16:

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 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 17:

`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 268).

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 18:

`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 19:

`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.

`-description` A brief description of the target process. Enclose the description in double quotes.

`-startupmode` Specifies whether to enable automatic startup of the target process:

- `on_demand` (default) starts the process when requested by a 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 98.
`-group`	Group name that starts the target process. The default is nobody. For more information, see page 100.

`-user`	User name that starts the target process. The default is nobody. For more information, see page 100.
`-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`***`]` ***`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`.

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`	

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.
`-description`	A brief description of the target process.
`-startupmode`	Specifies when to start the target process using one of these arguments: • `on_demand` (default) starts the process when requested by a client. • `disable` disables the process from starting.
`-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 98.
`-group`	Group name that starts the target process. The default is nobody. For more information, see page 100.

`-user`	User name that starts the target process. The default is nobody. For more information, see "File access permissions" on page 100.
`-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.

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 20:

`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 21:

`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 164.

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 163).

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 164).
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 287).

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 288). 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
```

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 22:

`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.

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 23:

`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 213.

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 24:

`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.

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 with the following commands

Table 25:

`pss_db archive_old_logs`	Archives old log files for the IOR specified.
`pss_db delete_old_logs`	Deletes old log files for IOR specified.
`pss_db checkpoint`	Performs checkpoint operations on the database referenced in the specified file.
`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.
`pss_db pre_backup`	Performs pre-backup operations on the database referenced in the specified file.

pss_db archive_old_logs

Synopsis

`pss_db archive_old_logs` *`IOR-file`*

Description

Archives old log files for the IOR specified.

pss_db delete_old_logs

Synopsis

`pss_db delete_old_logs` *`IOR-file`*

Description

Deletes old log files for IOR specified.

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:shmiop` namespace in the *Configuration 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.

Notification Service

Overview

The notification service is a CORBA service that enables applications to send events to any number of objects. For more about the CORBA notification service, see the *CORBA Notification Service Guide*.

`itadmin` commands let you manage the following components of a notification service:

Notification Service Management

The following commands let you manage an notification service instance.

Table 26:

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 213.

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 let you manage a notification service's event channel:

Table 27:

`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 create

Synopsis

`nc create` *`channel-name`*

Description

Creates an untyped event channel, in the default notification service, with the specified name.

Examples

The following command creates an untyped event channel named `my_channel`:

```
itadmin nc create 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 my_channel2
```

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 *Configuration Reference*.

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 28:

`tx begin`	Starts a transaction.
`tx commit`	Commits a transaction.
`tx resume`	Resumes a transaction.
`tx rollback`	Rolls back a transaction.
`tx suspend`	Suspends 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
```

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.

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 29:

`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 30:

`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.
%
```

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 31:

`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:0000000000000036494….

> 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 32:

`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 33:

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 34:

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:00000000000000224…
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 35:

`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
```

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 36:

`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.
`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 37:

`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.

Part IV

Appendices

In this part

This part contains the following:

Orbix Windows Services

During configuration, Orbix services are installed as Windows services that start up automatically at system startup.

This appendix describes how you can manage Orbix services as Windows services, and offers solution to typical problems. These services include:

- Configuration repository
- Locator daemon
- Node daemon
- Naming service
- Interface repository
- Event and notification services
- JMS
- Object transaction service

In this appendix

This appendix discusses the following topics:

Managing Orbix Services on Windows

Overview

If you choose to install Orbix services as Windows services, you can use the control panel's **Services** dialog to start, pause, continue, and stop any of the installed services. Equivalent functionality is provided through Orbix commands (see "Orbix Windows Service Commands").

> **Note:** In order to install and uninstall Orbix services as Windows services, you must execute the `install` and `uninstall` commands.

Identifying Orbix services as Windows services

Each installed Orbix service executable name has a Windows service name. This is a unique identifier for each service used by the Windows Service control manager. By default, a Windows service name has the following format:

```
IT ORB-name domain-name
```

Each service can create sub-keys under the following registry key:

```
HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Services
```

A Windows service name is used internally and must be unique. A Windows display name is shown in the Services dialog only. By default, the Windows service name and display name are the same.

Orbix Windows Service Commands

Overview

You can manage Orbix services from the command-line. Service commands have the following syntax:

```
exec-name [ORB-arguments] [exec-arguments] Win-service-command
    [Win-service-arguments]
```

ORB-arguments can be any of the ORB initialization parameters that are documented in Appendix C on page 397. In general, *ORB-arguments* is required only for the configuration repository. Because the configuration repository has its own domain, any service command that applies to the configuration repository must supply the `-ORBname` argument.

For example, the following command installs the configuration repository as a Windows service in the `cfr-AcmeProducts` configuration repository domain:

```
itconfig_rep -ORBname iona_services.config_rep -ORBdomain_name
    cfr-AcmeProducts install
```

You can execute the following commands on any Orbix Windows service:

`continue`
`help`
`install`
`pause`
`prepare`
`query`
`run`
`stop`
`uninstall`

continue

Synopsis: *executable-name* `continue`

Description: Resumes execution of the background service from its paused state.

help

Synopsis `executable-name help`

Description Prints a description message for the specified service.

install

Synopsis `executable-name install [-description=service-description]`

Description Installs the specified Orbix service as a Windows service. Because the Orbix configuration tool automatically installs the desired services as Windows services, you should rarely need to use this command to install a service manually.

The Windows service control manager starts installed Orbix services automatically during system startup. The `install` command specifies a Windows 32-bit service that runs in its own process.

Use the `-description` argument to change a display name for each service used by the Windows Service control manager. This leaves unchanged the internal service name used in the Windows registry key.

> **Note:** In general, it is recommended that you always install Orbix Windows services by running the Orbix configuration tool.

pause

Synopsis `executable-name pause`

Description Pauses execution of the specified background service.

prepare

Synopsis `executable-name prepare [-publish_to_file=name]`

Description Prepares the specified Orbix service for running, creating databases and initial object references. Use the `-publish_to_file` argument to write object references to a specified file; otherwise, `stdout` is used. This command is implicitly performed when Orbix is configured.

query

Synopsis `*executable-name* query`

Description For the specified service, outputs current status, configuration parameters, and dependencies on other services.

run

Synopsis `*executable-name* run -service`

Description Runs the specified Orbix service as a Windows service. The specified service must already be installed.

stop

Synopsis `*executable-name* stop`

Description Stops execution of the specified service. You must stop a service before you can uninstall it.

uninstall

Synopsis `*executable-name* uninstall`

Description Uninstalls the specified Orbix service as a Windows service. See "Uninstalling Orbix Windows Services" on page 375 for more details.

Orbix Windows Service Accounts

Overview

By default, Orbix installs services on Windows under a `LocalSystem` account that has no interaction with the desktop. You can change the `domain/user/passwd` with the Windows service control manager.

To change this password, use the **Services** options in the Windows **Control Panel**. You can also enable interaction with the desktop for a `LocalSystem` account only. Figure 30 shows details displayed for the locator service on Windows 2000.

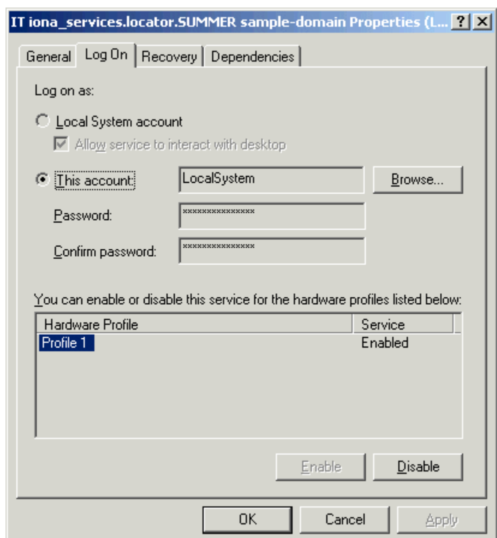

Figure 30: *Locator Service Details*

Setting service security

A service running under the `LocalSystem` account has no user account information associated with it. As a result, the service might have limited access to network resources. If this is not desired, use the **Services** options available in the Windows **Control Panel** to change the `user/group` and `passwd` for the service.

Orbix node daemons run under the `LocalSystem` account and activate other processes as the `LocalSystem` account. If this is not desired, use the **Services** options available in the Windows **Control Panel** to change the `user/group` and `passwd` for this service.

Running Orbix Windows Services

Overview

Before you can run an Orbix Windows service, the specified service must already be installed. You must supply the -service parameter to run as a Windows service.

When Orbix Windows services are installed, the order in which they must be run depends on whether your configuration domain is configuration repository-based or file-based.

Running in a configuration repository domain

When running Orbix Windows services in a configuration repository domain, run the services in the following order:

1. Configuration repository. For example:

```
itconfig_rep -ORBdomain_name cfr-AcmeProducts run -service
```

2. Locator daemon. For example:

```
itlocator run -service
```

3. Any other persistent service—interface repository, node daemon, naming service. For example:

```
itifr run -service
```

Running in a file-based domain

When running Orbix services as Windows services in a file-based domain, run Orbix services in the following order:

1. Locator daemon. For example:

```
itlocator run -service
```

2. Any other persistent service—interface repository, node daemon, naming service. For example:

```
itnode_daemon run -service
```

Logging Orbix Windows Services

Overview

In a configuration domain, logging is written to a file located in the same directory as the services, by default. By default, logging shows all informational messages, warnings, errors, and fatal errors.

The default log file name has the following format:

```
service-name.log.timestamp
```

For example, the locator's log file might have the following name:

```
locator.log.18012000
```

Setting user-defined logging

To change the logging output stream to a different file, set the following configuration variable in the configuration scope for each service:

```
plugins:local_log_stream:filename=filename
```

To add this variable to your configuration domain, use the `itadmin variable create` command. You must set this variable in the configuration scope for each service; for example, in the `locator` configuration scope:

```
itadmin variable create -scope iona_services.locator
   -type string -value "c:temp\it_locator.log"
   plugins:local_log_stream:filename
```

If your configuration domain is file based, you can manually add variables to your configuration file in the appropriate configuration scope. For example, to set logging for the node daemon, add the following in the `node_daemon` scope:

```
plugins:local_log_stream:filename="c:\temp\it_node_daemon.log";
```

See Chapter 11 on page 191 for more information on Orbix logging.

Uninstalling Orbix Windows Services

Overview

In order to cleanly remove any version of Orbix from your system, you should first uninstall all Orbix services from the Windows host.

In a configuration repository-based domain, complete the following procedure:

1. Stop and uninstall all services while the configuration repository and locator daemon are still running.
2. Stop and uninstall the locator daemon.
3. Stop and uninstall the configuration repository.

Commands for uninstalling services

The following series of commands show how you should stop and uninstall Orbix Windows services:

```
itnode_daemon stop
itnode_daemon uninstall

itifr stop
itifr uninstall

itnaming stop
itnaming uninstall

itevent stop
itevent uninstall

itlocator stop
itlocator uninstall

itconfig_rep -ORBdomain_name cfr-AcmeProducts stop

itconfig_rep -ORBdomain_name cfr-AcmeProducts uninstall
```

Troubleshooting Orbix/Windows Services

The following sections describe several common problems related to Orbix/Windows services, and how to resolve them.

Handling log-off events in activated servers

A node daemon that is installed as a Windows service continues to run in the background after users log off. It also activates server processes under the `LocalSystem` account. In order to shield these processes from log-off events (`CTRL_LOGOFF_EVENT`), the activated processes must have control handlers; otherwise, the logoff causes them to shut down.

Configuring for slow service startup

Occasionally, Windows services might require extra time to restart after system reboot. This might be due to a slow system, or to recovery of service-related databases.

Two changes in the configuration can help resolve this problem:

- Reduce the value set for `max_binding_iterations`, as in the following example:

  ```
  policies:binding_establishment:max_binding_iterations = "1";
  ```

- Increase the wait time for a service's pending operations (for example, start, pause, resume). The default wait time for all services is set to 900 seconds (15 minutes):

  ```
  plugins:plugin-name:nt_service_pending_op_wait = "900";
  ```

  Reset this variable for services, as necessary. For example, the following variable increases the locator's wait time to 20 minutes:

  ```
  plugins:locator:nt_service_pending_op_wait = "1200";
  ```

Run Control Scripts for Unix Platforms

Orbix services can be configured to start when the operating system enters the default run level and to shut down when the operating system leaves the default run level.

Overview

This appendix provides details on how Orbix registers its services with the operating system for automated startup and shutdown. Procedures for disabling, enabling and removal of automated startup registration are also covered.

Sometimes UNIX system administrators choose to customize run levels and run control scripts of their operating systems. If your run levels are customized, the details in this appendix will help you manually register your Orbix services for automated startup and shutdown or to use run control scripts generated by Orbix as a starting point for customization.

Note: For reliable startup and shutdown of Orbix services, it is recommended that you install the Java runtime, the Orbix components, the license file, the domain configuration files, the service databases and the log files on locally mounted filesystems.

You must have root privileges to perform tasks described in this appendix.

Operating Systems

Follow the links below for details on your operating system:

Solaris	page 379
AIX	page 382
HP-UX	page 386
IRIX	page 390
Red Hat Linux	page 393

For additional details on run levels and run control scripts refer to your operating system's documentation.

Solaris

Run level

The default run level is 3; this includes all services from run level 2.

Run control scripts

For a domain, <*domain*>, the following run control scripts are generated:

```
/etc/init.d/itsvs_<domain>
/etc/rc0.d/K27itsvs_<domain> -> /etc/init.d/itsvs_<domain>
/etc/rc1.d/K27itsvs_<domain> -> /etc/init.d/itsvs_<domain>
/etc/rc2.d/S97itsvs_<domain> -> /etc/init.d/itsvs_<domain>
/etc/rcS.d/K27itsvs_<domain> -> /etc/init.d/itsvs_<domain>
```

`/etc/init.d/itsvs_<domain>` contains the following:

```
#!/bin/sh
#
#         Copyright (c) 1993-2002 IONA Technologies PLC.
#         All Rights Reserved.
#
# <deployment-specific portion>
DOMAIN=boot
DOMAINS_ETC_DIR=/etc/opt/iona
DOMAINS_VAR_DIR=/var/opt/iona
# </deployment-specific portion>

DOMAIN_START_SCRIPT=
   ${DOMAINS_ETC_DIR}/bin/tart_${DOMAIN}_services
DOMAIN_STOP_SCRIPT=
   ${DOMAINS_ETC_DIR}/bin/stop_${DOMAIN}_services
```

```
rval=0
case "$1" in
   'start')
if [ -x ${DOMAIN_START_SCRIPT} ]; then
         echo "Starting IONA Orbix services for domain ${DOMAIN}"
         ${DOMAIN_START_SCRIPT}
else
echo “ERROR: Failed to start IONA Orbix services for domain
   ${DOMAIN} - \
          domain start script ${DOMAIN_START_SCRIPT} does not
   exist or is not executable"
rval=1
fi
   ;;
   'stop')

if [ -x ${DOMAIN_STOP_SCRIPT} ]; then
         echo "Stopping IONA Orbix services for domain ${DOMAIN}"
         ${DOMAIN_STOP_SCRIPT}
else
echo “ERROR: Failed to stop IONA Orbix services for domain
   ${DOMAIN} - \
         domain stop script ${DOMAIN_STOP_SCRIPT} does not exist
   or is not executable"
rval=1
fi
   ;;
   *)
   echo "IONA Orbix run control script for domain ${DOMAIN}”
echo “Usage: $0 { start | stop }"
   rval=1
   ;;
   esac
   exit $rval
```

Disabling automatic services

To temporarily disable automatic startup and shutdown for domain *<domain>*:

1. Stop *<domain>* services by running

```
> stop_<domain>_services
```

2. Rename the following symbolic links by prepending a _ to their names:

```
/etc/rc0.d/K27itsvs_<domain>
/etc/rc1.d/K27itsvs_<domain>
/etc/rc2.d/S97itsvs_<domain>
/etc/rcS.d/K27itsvs_<domain>
```

Enabling automatic service

To enable automatic startup and shutdown for *<domain>*:

1. Rename the following symbolic links by removing leading _ from their names:

```
/etc/rc0.d/_K27itsvs_<domain>
/etc/rc1.d/_K27itsvs_<domain>
/etc/rc2.d/_S97itsvs_<domain>
/etc/rcS.d/_K27itsvs_<domain>
```

2. Start domain services by running:

```
> start_<domain>_services
```

Unregistering automatic services

To unregister automatic startup and shutdown for *<domain>*:

1. Stop *<domain>* services by running:

```
> stop_<domain>_services
```

2. Remove the following files:

```
/etc/rc0.d/K27itsvs_<domain>
/etc/rc1.d/K27itsvs_<domain>
/etc/rc2.d/S97itsvs_<domain>
/etc/rcS.d/K27itsvs_<domain>
/etc/init.d/itsvs_<domain>
```

AIX

Run level

The default run level is 2.

Actions

For a domain named <*domain*>, Orbix performs the following actions:

- Makes an entry in `/etc/inittab` with `/usr/sbin/mkitab`:

```
itsvs_<domain>:2:wait:/etc/rc.itsvs_<domain> start >/dev/console
   2>&1 # IONA Orbix services for domain <domain>
```

- Creates a run control script `/etc/rc.itsvs_<domain>` that contains the following:

```
#!/bin/sh
#
#         Copyright (c) 1993-2002 IONA Technologies PLC.
#         All Rights Reserved.
#

<deployment-specific portion>
DOMAIN=boot
DOMAINS_ETC_DIR=/etc/opt/iona
DOMAINS_VAR_DIR=/var/opt/iona
# </deployment-specific portion>

#
DOMAIN_START_SCRIPT=
   ${DOMAINS_ETC_DIR}/bin/start_${DOMAIN}_services
DOMAIN_STOP_SCRIPT=
   ${DOMAINS_ETC_DIR}/bin/stop_${DOMAIN}_services
```

```
rval=0
case "$1" in
'start')
if [ -x ${DOMAIN_START_SCRIPT} ] ; then
echo "Starting IONA Orbix services for domain ${DOMAIN}"
${DOMAIN_START_SCRIPT}
else
echo " ERROR: Failed to start IONA Orbix services for domain
   ${DOMAIN} - \
                    domain start script ${DOMAIN_START_SCRIPT}
   does not exist or is not executable"
rval=1
fi
;;
'stop')
if [ -x ${DOMAIN_STOP_SCRIPT} ] ; then
echo "Stopping IONA Orbix services for domain <domain>"
${DOMAIN_STOP_SCRIPT}
else
echo "Can not stop IONA Orbix servies for domain <domain> - \
         domain stop script ${DOMAIN_STOP_SCRIPT} does not exist
   or is not executable"
rval=1
fi
;;
*)
echo "IONA Orbix run control script for domain ${DOMAIN}”
echo "Usage: $0 { start | stop }"
rval=1
;;
esac
exit $rval
```

- Creates `/etc/rc.shutdown` if it does not exist, and adds the following code:

```
#<IONA Orbix <domain> >
if [ -x /etc/rc.itsvs_<domain> ]; then
/etc/rc.itsvs_<domain> stop
else
echo "ERROR: Failed to stop IONA Orbix services for domain
   <domain> - \
         /etc/rc.itsvs_<domain> does not exist or is not
   executable"
fi
#</IONA Orbix <domain> >

exit 0
```

Note: `/etc/rc.shutdown` *must* return 0, otherwise the AIX shutdown sequence is interrupted.

Disable automatic services

To temporarily disable automatic startup and shutdown for *<domain>*:

1. Stop domain services by running

```
> stop_<domain>_services
```

2. Comment out the `itsvs_<domain>` entry in `/etc/inittab`.
3. Comment out the code between `<IONA Orbix <domain> >` and `</IONA Orbix <domain> >` tags in `/etc/rc.shutdown`.

Enable automatic services

To enable automatic startup and shutdown for *<domain>*:

1. Uncomment the code between `<IONA Orbix <domain> >` and `</IONA Orbix <domain> >` tags in `/etc/rc.shutdown`.
2. Uncomment the `itsvs_<domain>` entry in `/etc/inittab`.
3. Start domain services by running

```
> start_<domain>_services
```

Unregister automatic services

To unregister automatic startup and shutdown for <*domain*>:

1. Remove the itsvs_<*domain*> entry from /etc/inittab by running

```
> rmitab itsvs_<domain>
```

2. If <*domain*> is the only Orbix domain registered for automatic startup and shutdown, remove file /etc/rc.shutdown. Otherwise, remove the code between <IONA Orbix <*domain*> > and </IONA Orbix <*domain*> > tags in/etc/rc.shutdown.
3. Remove /etc/rc.itsvs_<*domain*>.

HP-UX

Run level

The default run level is 3. See the output of run control scripts for the last boot of the machine in /etc/rc.log. The previous boot log is in /etc/rc.log.old.

Run control scripts

For a domain, <*domain*>, the following files are generated:

```
/sbin/rc2.d/K270itsvs_<domain> -> /sbin/init.d/itsvs_<domain>
/sbin/rc3.d/S970itsvs_<domain> -> /sbin/init.d/itsvs_<domain>
/sbin/init.d/itsvs_<domain>
/etc/rc.config.d/itsvs_<domain>
```

The contents of /sbin/init.d/itsvs_<*domain*> is as follows:

```
#!/bin/sh
#
#         Copyright (c) 1993-2002 IONA Technologies PLC.
#         All Rights Reserved
#
# <deployment-specific portion>
DOMAIN=boot
DOMAINS_ETC_DIR=/etc/opt/iona
DOMAINS_VAR_DIR=/var/opt/iona
# </deployment-specific portion>

DOMAIN_START_SCRIPT=
   ${DOMAINS_ETC_DIR}/bin/start_${DOMAIN}_services
DOMAIN_STOP_SCRIPT=
   ${DOMAINS_ETC_DIR}/bin/stop_${DOMAIN}_services

if [ -r /etc/rc.config.d/itsvs_ ${DOMAIN} ] ;
  then . /etc/rc.config.d/itsvs_${DOMAIN}
else
  echo "WARNING: /etc/rc.config.d/itsvs_${DOMAIN} configuration
   file is missing or is not readable"
fi
```

```
rval=0

case "$1" in
  'start_msg')
    echo "Starting IONA Orbix services for domain ${DOMAIN}"
  ;;

  'stop_msg')
    echo "Stopping IONA Orbix services for domain ${DOMAIN}"
  ;;

   'start')
if [ "ITSVS_${DOMAIN}" -eq 1 ]; then
  if [ -x ${DOMAIN_START_SCRIPT} ]; then
    echo "Starting IONA Orbix services for domain ${DOMAIN}"
    ${DOMAIN_START_SCRIPT}
    rval=4
  else
    echo "ERROR: Failed to start IONA Orbix services for domain
   ${DOMAIN} - \ domain start script ${DOMAIN_START_SCRIPT} does
   not exist or is not executable"
    rval=1
  fi
else
  # domain is disabled
  rval=2
fi
   ;;

   'stop')
if [ "ITSVS_${DOMAIN}" -eq 1 ]; then
  if [ -x ${DOMAIN_STOP_SCRIPT} ]; then
    echo "Stopping Orbix services for the ${DOMAIN} domain"
   ${DOMAIN_STOP_SCRIPT}
    rval=4
  else
    echo "ERROR: Failed to start IONA Orbix services for domain
   ${DOMAIN} - \ domain stop script ${DOMAIN_STOP_SCRIPT} does
   not exist or is not executable"
    rval=1
  fi
else
  # domain is disabled
  rval=2
fi
   ;;
```

```
*)
   echo "IONA Orbix run control script for domain ${DOMAIN}"
   echo "Usage: $0 { start | stop }"
   rval=1
  ;;
   esac
   exit $rval
```

`/etc/rc.config.d/itsvs_<domain>` contains the following:

```
#
#         Copyright (c) 1993-2002 IONA Technologies PLC.
#         All Rights Reserved
#
#         IONA Orbix services, domain <domain> configuration
#         ITSVS_<DOMAIN>: set to 1 to enable Orbix services for
   domain <domain>

ITSVS_<DOMAIN>=1
```

Disable automatic services

To temporarily disable automatic startup and shutdown for <*domain*>:

1. Stop domain services by running

```
> stop_<domain>_services
```

2. Set `ITSVS_<DOMAIN>` to 0 in `/etc/rc.config.d/itsvs_<domain>`.

Enable automatic services

To enable automatic startup and shutdown for <*domain*>:

1. Set `ITSVS_<DOMAIN>` to 1 in `/etc/rc.config.d/itsvs_<domain>`.
2. Start domain services by running

```
> start_<domain>_services
```

Unregister automatic services

To unregister automatic startup and shutdown for <*domain*>:

1. Stop domain services by running

```
> stop_<domain>_services
```

2. Remove the following files:

```
/sbin/rc2.d/K270itsvs_<domain>
/sbin/rc3.d/S970itsvs_<domain>
/sbin/init.d/itsvs_<domain>
/etc/rc.config.d/itsvs_<domain>
```

IRIX

Run level

The default run level is 2.

Run control scripts

For a domain, <*domain*>, the following files are generated:

```
/etc/init.d/itsvs_<domain>
/etc/r0.d/K27itsvs_<domain> -> /etc/init.d/itsvs_<domain>
/etc/r2.d/S97itsvs_<domain> -> /etc/init.d/itsvs_<domain>
/var/config/itsvs_<domain>
```

`/etc/init.d/itsvs_<domain>` contains the following:

```
#!/bin/sh
#
#         Copyright (c) 1993-2002 IONA Technologies PLC.
#         All Rights Reserved.
#

# <deployment-specific portion>
DOMAIN=boot
DOMAINS_ETC_DIR=/etc/opt/iona
DOMAINS_VAR_DIR=/var/opt/iona
# </deployment-specific portion>

DOMAIN_START_SCRIPT=
   ${DOMAINS_ETC_DIR}/bin/start_${DOMAIN}_services
DOMAIN_STOP_SCRIPT=
   ${DOMAINS_ETC_DIR}/bin/stop_${DOMAIN}_services

rval=0

if [ ! /sbin/chkconfig itsvs_${DOMAIN} ]; then
# domain is disabled
  exit $rval
fi
```

```
case "$1" in
   'start')
if [ -x ${DOMAIN_START_SCRIPT} ]; then
  echo "Starting Orbix services for domain ${DOMAIN}"
  ${DOMAIN_START_SCRIPT}
else
  echo "ERROR: Failed to start IONA Orbix services for domain
   ${DOMAIN} - "
  echo "domain start script ${DOMAIN_START_SCRIPT} does not exist
   or is not executable"
  rval=1
fi
;;

   'stop')
if [ -x ${DOMAIN_STOP_SCRIPT} ] ; then
  echo "Stopping IONA Orbix services for domain ${DOMAIN}"
  ${DOMAIN_STOP_SCRIPT}
else
  echo "ERROR: Failed to stop IONA Orbix servies for domain
   ${DOMAIN} - "
  echo "domain stop script ${DOMAIN_STOP_SCRIPT} does not exist
   or is not executable"
  rval=1
fi
;;

   *)
   echo "IONA Orbix run control script for domain ${DOMAIN}”
   echo “Usage: $0 { start | stop }"
   rval=1
   ;;
   esac
   exit $rval
```

Disable automatic services

To temporarily disable automatic startup and shutdown for <*domain*>:

1. Stop domain services by running

```
> stop_<domain>_services
```

2. Run

```
> /sbin/chkconfig itsvs_<domain> off
```

Enable automatic services

To enable automatic startup and shutdown for <*domain*>:

1. Run

```
> /sbin/chkconfig itsvs_<domain> on
```

2. Start domain services by running

```
> start_<domain>_services
```

Unregister automatic services

To unregister automatic startup and shutdown for <*domain*>:

1. Stop domain services by running

```
> stop_<domain>_services
```

2. Remove the following files:

```
/var/config/itsvs_<domain>
/etc/r0.d/K27itsvs_<domain>
/etc/r2.d/S97itsvs_<domain>
/etc/init.d/itsvs_<domain>
```

Red Hat Linux

Run level

The default run level is either 3 or 5. Orbix determines the default run level.

Run control scripts

Run control scripts generated by the Orbix configuration tool are compatible with `chkconfig(8)` and `linuxconf`.

For a domain named <*domain*>, the following files are generated by the Orbix configuration tool:

```
/etc/rc0.d/K27itsvs_<domain> -> /etc/rc.d/init.d/itsvs_<domain>
/etc/rc1.d/K27itsvs_<domain> -> /etc/rc.d/init.d/itsvs_<domain>
/etc/rc2.d/K27itsvs_<domain> -> /etc/rc.d/init.d/itsvs_<domain>
/etc/rc[3|5].d/S97itsvs_<domain> ->
   /etc/rc.d/init.d/itsvs_<domain>
/etc/rc6.d/K27itsvs_<domain> -> /etc/rc.d/init.d/itsvs_<domain>
```

`/etc/rc.d/init.d/itsvs_<domain>` contains the following:

```
#!/bin/bash
#
#         Copyright (c) 1993-2002 IONA Technologies PLC.
#         All Rights Reserved
#
# chkconfig:   [3|5] 27 97
# description: IONA Orbix services, domain <domain>
#

# <deployment-specific portion>
DOMAIN=boot
DOMAINS_ETC_DIR=/etc/opt/iona
DOMAINS_VAR_DIR=/var/opt/iona
# </deployment-specific portion>

DOMAIN_START_SCRIPT=
   ${DOMAINS_ETC_DIR}/bin/start_${DOMAIN}_services
DOMAIN_STOP_SCRIPT=
   ${DOMAINS_ETC_DIR}/bin/stop_${DOMAIN}_services
DOMAIN_LOCK_FILE=/var/lock/subsys/itsvs_${DOMAIN}
```

```
rval=0
case "$1" in
   'start')
# check if the domain is running
[ -f "${DOMAIN_LOCK_FILE}" ] && exit $rval
if [ -x ${DOMAIN_START_SCRIPT} ]; then
  echo "Starting IONA Orbix services for domain <domain>"
  ${DOMAIN_START_SCRIPT}
  touch ${DOMAIN_LOCK_FILE}
else
  echo "ERROR: Failed to start IONA Orbix services for domain
   <domain> - "
  echo "domain start script ${DOMAIN_START_SCRIPT} does not exist
   or is not executable"
  rval=1
fi
   ;;

   'stop')
# check if the domain is not running
[ ! -f "${DOMAIN_LOCK_FILE}" ] && exit $rval
if [ -x ${DOMAIN_STOP_SCRIPT} ]; then
  echo "Stopping IONA Orbix services for domain <domain>"
  ${DOMAIN_STOP_SCRIPT}
else
  echo "ERROR: Failed to stop IONA Orbix services for domain
   <domain> - "
  echo "domain stop script ${DOMAIN_STOP_SCRIPT} does not exist
   or is not executable"
fi
rm -f ${DOMAIN_LOCK_FILE}
   ;;

*)
   echo "IONA Orbix run control script for domain ${DOMAIN}”
   echo “Usage: $0 { start | stop }"
   rval=1
;;
   esac
   exit $rval
```

Disable automatic services

To temporarily disable automatic startup and shutdown for <*domain*>:

1. Stop domain services by running

```
> stop_<domain>_services
```

2. Run

```
> chkconfig –del itsvs_<domain>
```

Enable automatic services

To enable automatic startup and shutdown for <*domain*>:

1. Run

```
> chkconfig –add itsvs_<domain>
```

2. Start domain services by running

```
> start_<domain>_services
```

Unregister automatic services

To unregister automatic startup and shutdown for <*domain*>:

1. Stop domain services by running

```
> stop_<domain>_services
```

2. Run

```
> chkconfig –del itsvs_<domain>
```

3. Remove the following files:

```
/etc/rc.d/init.d/itsvs_<domain>
/var/lock/subsys/itsvs_<domain>
```

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:\iona\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/iona` 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 IONA 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:\iona
```

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/iona` on UNIX, and `%SystemDrive%\Program Files\IONA` 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 E

Debugging IOR Data

Orbix includes a tool for analyzing IOR data and finding possible causes for malformed IORs.

In this appendix

This appendix 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 it's 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 will decode CORBA inter-operable object reference (IOR) content and present it in readable format through `stdout`. The utility's output also includes debugging information to assist in analyzing the cause of malformed IOR data.

Synopsis

`iordump [-no_host_check] {`*`file`*` | -}`

`iordump [-no_host_check] `*`IOR:...`*

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 specificed 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.

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.
`-`	Specifies that the IOR data is to be read from `stdin`.
`IOR:...`	Specifies the IOR to decode on the command line.

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.....
```

Notes

Data other than a single IOR in a file will result in the whole data being analyzed as a single IOR. Only in the case of stringified IORs are trailing newlines, carriage returns and nulls removed.

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 417.

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
   563743a312e300001000000000000006a000000010102000e00000036332e
   36352e3133332e32353000a70f1b0000003a3e0231310c00000000ec09000
   08d200000008000000000000000000000002000000010000001800000001000000
   010001000000000000010100010000000090101000600000006000000001000
   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*

```
                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
```

In this section

This section discusses the following topics:

Stringified Data Output

All output begins with the stringified IOR such as:

```
Stringified IOR is: ([string/coded data] length: 312 / 154 bytes)

>>
   IOR:010000001c00000049444c3a53696d706c652f53696d706c654f626a6
   563743a312e30000100000000000000006a000000010102000e00000036332e
   36352e3133332e32353000a70f1b0000003a3e0231310c00000000ec09000
   08d20000008000000000000000000000200000001000000180000000100000
   0100010000000000000101000100000009010100060000000600000001000
   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.

Index

D

E

F

G

O

P