



CORBA Administrator's Guide

Version 6.0, August 2003

Making Software Work Together™

IONA, IONA Technologies, the IONA logo, Orbix, Orbix/E, Orbacus, Artix, Orchestrator, Mobile Orchestrator, Enterprise Integrator, Adaptive Runtime Technology, Transparent Enterprise Deployment, and Total Business Integration are trademarks or registered trademarks of IONA Technologies PLC and/or its subsidiaries.

Java and J2EE are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

CORBA is a trademark or registered trademark of the Object Management Group, Inc. in the United States and other countries. All other trademarks that appear herein are the property of their respective owners.

While the information in this publication is believed to be accurate, IONA Technologies PLC makes no warranty of any kind to this material including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. IONA Technologies PLC shall not be liable for errors contained herein, or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

COPYRIGHT NOTICE

No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, photocopying, recording or otherwise, without prior written consent of IONA Technologies PLC. No third party intellectual property right liability is assumed with respect to the use of the information contained herein. IONA Technologies PLC assumes no responsibility for errors or omissions contained in this book. This publication and features described herein are subject to change without notice.

Copyright © 2001, 2003 IONA Technologies PLC. All rights reserved.

All products or services mentioned in this manual are covered by the trademarks, service marks, or product names as designated by the companies who market those products.

Updated: 27-Nov-2003

M 3 0 3 0

Contents

List of Figures	ix
List of Tables	xi
Preface	xiii

Part I Getting Started

Chapter 1 Application Server Platform Environment	3
Basic CORBA Model	4
Simple Application Server Platform Application	6
Portable Object Adapter	8
Broader Application Server Platform Environment	10
Managing Object Availability	11
Scaling Application Server Platform Environments with Configuration Domains	13
Using Dynamic Application Server Platform Applications	16
Application Server Platform Administration	17
Chapter 2 Selecting a Configuration Model	19
Application Server Platform Development Environment Models	21
Independent Development Environments	22
Distributed Development and Test Environments	25
Configuration Models	26
Getting the Most from Your Application Server Platform Environment	29
Using Capabilities of Well-Designed Application Server Platform Applications	30
Using the Right Data Storage Mechanism	32
Getting the Most from Application Server Platform Configuration	33
Chapter 3 Configuring an Application Server Platform Domain	35
Running the Orbix E2A Configure Tool	37
Licensing	42

Configuring an Application Server Platform Domain	44
Create a New Domain	46
Connect a Client Machine to a Domain	54
Use Expert Mode	56
Localize a Preconfigured Domain	60
Deploy a Domain on the Remaining Hosts	63
Configure a Machine with no GUI	66
Configure a Multihomed Machine	67
Configure a Domain without Using the itconfigure GUI Interface	68
Configure ASP to Listen on a Fixed Port	72
Replicating a Services in a Domain	74
Starting and Stopping Application Server Platform Services	75
Setting Java ORB Classes	76
Running Sample Applications	77

Part II Managing an Application Server Platform Environment

Chapter 4 Configuring Application Server Platform Applications	81
How an ORB Gets its Configuration	83
Locating the Configuration Domain	85
Obtaining an ORB's Configuration	87
Setting Buffer Sizes	94
Configuration Variables and Namespaces	96
Managing Configuration Domains	98
Chapter 5 Managing Persistent CORBA Servers	101
Registering Persistent Servers	103
Server Environment Settings	106
Windows Environment Settings	107
UNIX Environment Settings	108
Managing a Location Domain	110
Managing Server Processes	111
Managing the Locator Daemon	112
Managing Node Daemons	114
Listing Location Domain Data	117

Modifying a Location Domain	118
Ensuring Unique POA Names	119
Using Direct Persistence	121
CORBA Application Servers	122
Application Server Platform Services	125
Chapter 6 Configuring Scalable Applications	127
Active Connection Management	129
Fault Tolerance and Replicated Servers	131
About Replicated Servers	132
Automatic Replica Failover	135
Direct Persistence and Replica Failover	136
Building a Replicated Server	139
Example 1: Building a Replicated Server to Start on Demand	140
Example 2: Updating a Replicated Server	143
Example 3: Dynamically Changing the Load Balancing Algorithm	144
Replicating Domain Services	145
Chapter 7 Managing the Naming Service	147
Naming Service Administration	149
Naming Service Commands	151
Controlling the Naming Service	152
Building a Naming Graph	153
Creating Naming Contexts	155
Creating Name Bindings	156
Maintaining a Naming Graph	158
Managing Object Groups	159
Chapter 8 Managing an Interface Repository	161
Controlling the Interface Repository Daemon	163
Managing IDL Definitions	164
Browsing Interface Repository Contents	165
Adding IDL Definitions	167
Removing IDL Definitions	168
Chapter 9 Orbix E2A Firewall Proxy Service	169
Firewall Proxy Service Functionality	170
Using the Firewall Proxy Service in a Deployed Environment	171

Known Restrictions	174
Chapter 10 Managing CORBA Service Databases	175
Berkeley DB Environment	177
Performing Checkpoints	178
Managing Log Files	179
Performing Online Back-Ups	181
Performing a Recovery	182
Chapter 11 Setting Up Application Server Platform Logging	185
Setting Logging Filters	187
Setting Logging for J2EE	189
Registering Log4J with JMX	191
Logging Subsystem	193
Logging Severity Levels	195
Redirecting Log Output	197
 Part III Reference	
Chapter 12 Starting Application Server Platform Services	201
Starting and Stopping Configured Services	202
Starting Application Server Platform Services Manually	203
Stopping Services Manually	212
Chapter 13 Managing Application Server Platform Services With itadmin	213
Using itadmin	214
Command Syntax	217
Services and Commands	220
Configuration Domain	221
Configuration Repository	222
Namespaces	224
Scopes	227
Variables	229
Location Domain	233
Locator Daemon	234
Named Key	237

Node Daemon	240
ORB Name	244
POA	247
Server Process	253
Naming Service	263
Names	264
Object Groups	268
Interface Repository	275
IDL Definitions	276
Repository Management	277
Event Service	281
Event Service Management	282
Event Channel	284
Persistent State Service	289
Notification Service	293
Notification Service Management	294
Event Channel	298
Object Transaction Service	301
Object Transaction Service Encina	305
Security Service	313
Logging On	315
Managing Checksum Entries	316
Managing Pass Phrases	319
Trading Service	323
Trading Service Administrative Settings	324
Federation Links	329
Regular Offers	333
Proxy Offers	335
Type Repository	337
Bridging Service	341
JMS Broker	345

Part IV Appendices

Appendix A Application Server Platform Windows Services	349
Managing Application Server Platform Services on Windows	351
Application Server Platform Windows Service Commands	352

Application Server Platform Windows Service Accounts	355
Running Application Server Platform Windows Services	357
Logging Application Server Platform Windows Services	359
Uninstalling Application Server Platform Windows Services	360
Troubleshooting Application Server Platform/Windows Services	361
Appendix B Run Control Scripts for Unix Platforms	363
Solaris	365
AIX	369
HP-UX	373
IRIX	379
Red Hat Linux	383
Appendix C ORB Initialization Settings	387
Appendix D Internationalization Configuration Variables	393
Description of Configuration Variables	394
Default Values	396
Appendix E Development Environment Variables	399
Appendix F Debugging IOR Data	401
IOR Data Formats	402
Using iordump	405
iordump Output	407
Stringified Data Output	410
ASCII-Hex Data Output	411
Data, Warning, Error and Information Text	412
Errors	413
Warnings	417
Index	419

List of Figures

Figure 1: Basic CORBA Model	5
Figure 2: Overview of a Simple Application Server Platform Application	6
Figure 3: A POA's Role in Client–Object Communication	8
Figure 4: Simple Configuration Domain and Location Domain	14
Figure 5: Multiple Configuration Domains	15
Figure 6: An Independent Development and Test Environment	22
Figure 7: Multiple Independent Development and Test Environments	23
Figure 8: A Distributed Development and Test Environment	25
Figure 9: Application Server Platform Environment with Local Configuration	27
Figure 10: Application Server Platform Environment with Centralized Configuration	28
Figure 11: Main Configuration Window	41
Figure 12: Entering the License File	42
Figure 13: Services to Deploy	46
Figure 14: Startup Mode and Base Port	47
Figure 15: Setting security features	49
Figure 16: Replica Configuration	51
Figure 17: Selecting services to deploy	52
Figure 18: Configuration domain summary.	53
Figure 19: Select a CFR for linking	54
Figure 20: Advanced Domain Specification	56
Figure 21: Configure the default domain settings	57
Figure 22: Advance Service Configuration	58
Figure 23: Configure Advanced Service Settings	59
Figure 24: Deployment details	64
Figure 25: How an Application Server Platform Application Obtains its Configurations	83

LIST OF FIGURES

Figure 26: Hierarchy of Configuration Scopes

87

Figure 27: Naming Context Graph

153

List of Tables

Table 1: Configuration Domain Management Tasks	98
Table 2: itadmin Commands that List Location Domain Data	117
Table 3: itadmin Commands that Modify a Location Domain	118
Table 4: itadmin Commands that Remove Location Domain Components	118
Table 5: Naming Graph Maintenance Commands	158
Table 6: Application Server Platform Logging Subsystems	193
Table 7: Application Server Platform Logging Severity Levels	196
Table 8: Commands to Manually Start Application Server Platform Services.	203
Table 9: itadmin Commands for Stopping Application Server Platform Services	212
Table 10: CORBA Configuration Variables	394
Table 11: J2EE Configuration Variables	395
Table 12: C++ CORBA (non-MVS platforms)	396
Table 13: C++ CORBA MVS Platforms	396
Table 14: Java CORBA (ISO-8859-1/Cp-1252/US-ASCII locale)	397
Table 15: Java CORBA (Shift_JIS locale)	397
Table 16: Java CORBA (EUC-JP locale)	397
Table 17: Java CORBA (other locales)	398

LIST OF TABLES

Preface

Introduction

IONA Orbix E2A Application Server Platform is a software environment for building and integrating distributed object-oriented applications. Application Server Platform provides a full implementation of the Common Object Request Broker Architecture (CORBA) from the Object Management Group (OMG). Application Server Platform is compliant with version 2.4 of the OMG'S CORBA specification. This guide explains how to configure and manage the components of an Application Server Platform 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 Application Server Platform environments, and programmers developing Application Server Platform applications.

Organization

This guide is divided into the following parts:

- [Getting Started](#) introduces the Application Server Platform environment, and the basic concepts required to understand how it works. It also describes how to run Orbix E2A Configure, which creates a configuration domain and deploys applications.
 - [Managing an Application Server Platform Environment](#) explains how to manage each component of an Application Server Platform environment. It provides task-based information and examples.
 - [Reference](#) provides a comprehensive reference for all Application Server Platform configuration variables and administration commands.
 - [Appendices](#) explain how to use Application Server Platform components as Windows NT services. They also provide reference information for initialization parameters and environment variables.
-

Related documentation

Orbix documentation also includes:

- *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:

Application Server Platform Environment	page 3
Selecting a Configuration Model	page 19
Configuring an Application Server Platform Domain	page 35

Application Server Platform Environment

Orbix E2A Application Server Platform is a network software environment that enables programmers to develop and run distributed applications.

Overview

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

In this chapter

This chapter contains the following sections:

Basic CORBA Model	page 4
Simple Application Server Platform Application	page 6
Broader Application Server Platform Environment	page 10
Application Server Platform Administration	page 17

Basic CORBA Model

Overview

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

To establish and use a proper Application Server Platform environment, administrators and programmers need to know how the Application Server Platform components interact, so that applications can find and use them correctly. This chapter starts with a sample application that requires a minimal Application Server Platform 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.

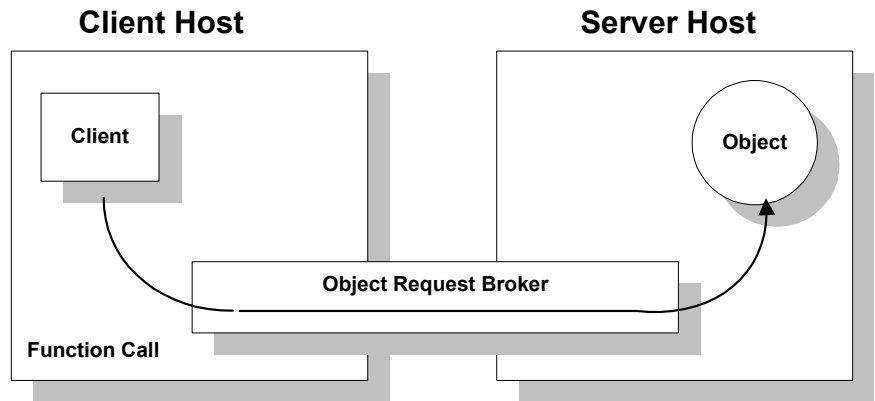


Figure 1: *Basic CORBA Model*

Simple Application Server Platform Application

Overview

A simple Application Server Platform 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.

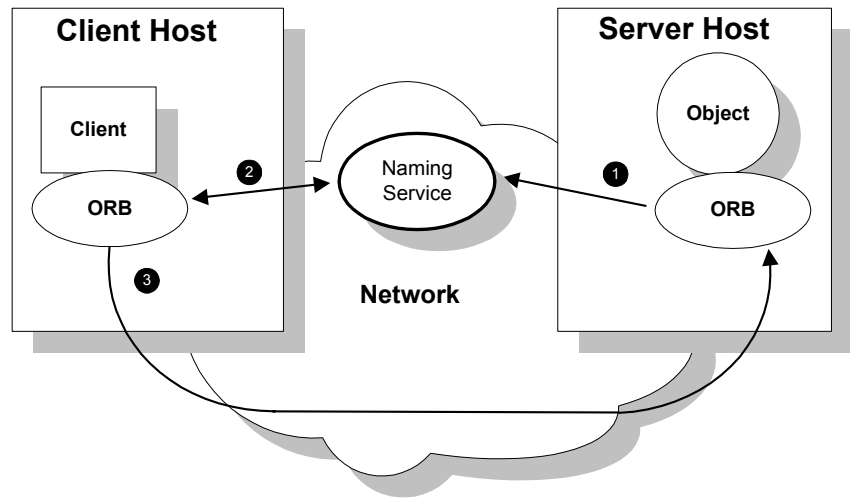


Figure 2: Overview of a Simple Application Server Platform Application

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 6](#) omits details that all applications require. For example, Application Server Platform 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.

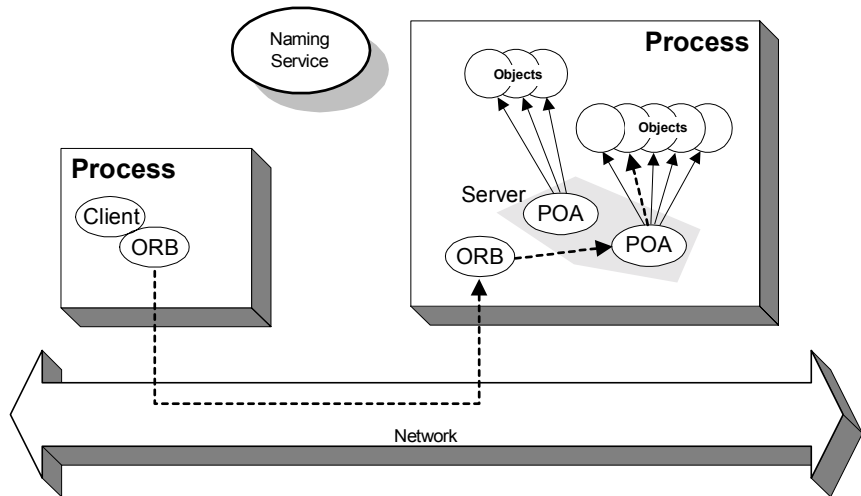


Figure 3: A POA's Role in Client–Object Communication

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 Application Server Platform environment, as described in the next section.

Broader Application Server Platform Environment

Overview

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

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

In this section

This section discusses the following topics:

Managing Object Availability	page 11
Scaling Application Server Platform Environments with Configuration Domains	page 13
Using Dynamic Application Server Platform Applications	page 16

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

Application Server Platform 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 *IORs*. 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, Application Server Platform 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 Application Server Platform location domain

An Application Server Platform 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 application server must run a node daemon. The node daemon starts, monitors, and manages application servers on its machine. The locator daemon relies on node daemons to start processes and tell it when new processes are available.

Scaling Application Server Platform 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 Application Server Platform 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. Application Server Platform 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 Application Server Platform 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.

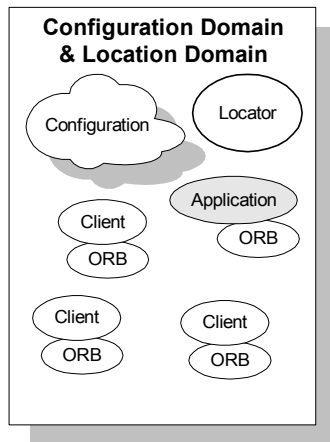


Figure 4: *Simple Configuration Domain and Location Domain*

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.

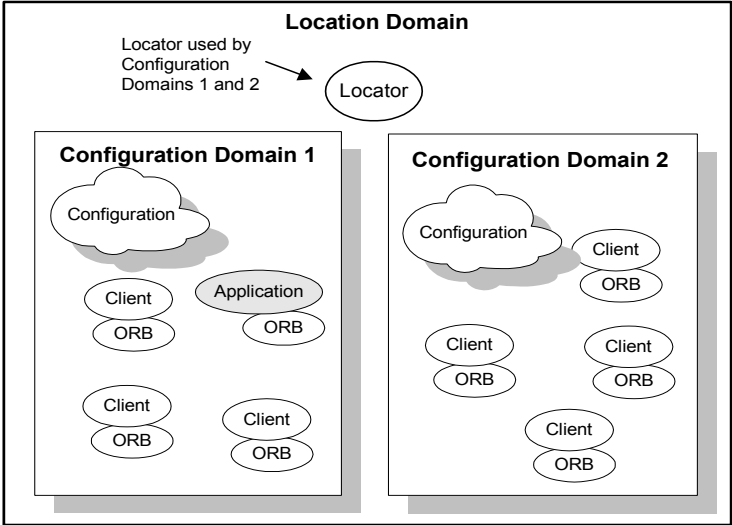


Figure 5: Multiple Configuration Domains

Using Dynamic Application Server Platform 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.

Application Server Platform 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.

Application Server Platform 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. Application Server Platform 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](#).

Application Server Platform Administration

Overview

Application Server Platform services, such as the naming service, and Application Server Platform 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. Application Server Platform 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

Application Server Platform administration tasks include the following:

- [Managing an Application Server Platform environment](#)
- [Application deployment and management](#)
- [Troubleshooting](#)

Managing an Application Server Platform environment

This involves starting up Application Server Platform services, or adding, moving, and removing Application Server Platform components. For example, adding an interface repository to a configuration domain, or modifying configuration settings (for example, initial references to Application Server Platform 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 Application Server Platform 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 Application Server Platform `itadmin` command interface lets you control all aspects of Application Server Platform administration. Administration commands can be executed from any host. For detailed reference information about Application Server Platform administration commands, see [Part III](#) of this manual.

Selecting a Configuration Model

This chapter shows how the Application Server Platform 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. Application Server Platform 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 Application Server Platform environment models that can handle one or many applications. This chapter also explains Application Server Platform configuration mechanisms, and how to scale an Application Server Platform environment to support more applications, more users, and a wider geographical area. For detailed information on how to set up your Application Server Platform configuration, see [Chapter 3](#).

In this chapter

This chapter contains the following sections:

Application Server Platform Development Environment Models	page 21
Configuration Models	page 26
Getting the Most from Your Application Server Platform Environment	page 29
Getting the Most from Application Server Platform Configuration	page 33

Application Server Platform Development Environment Models

Overview

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

Application testing requires deployment of Application Server Platform 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 Application Server Platform 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	page 22
Distributed Development and Test Environments	page 25

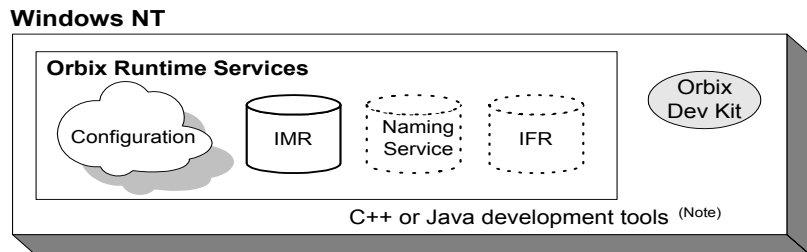
Independent Development Environments

Overview

This section discusses some typical models of Application Server Platform 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.



Note. C++ or Java tools must exist on each development platform.



A dotted outline indicates an optional runtime service.

Figure 6: *An Independent Development and Test Environment*

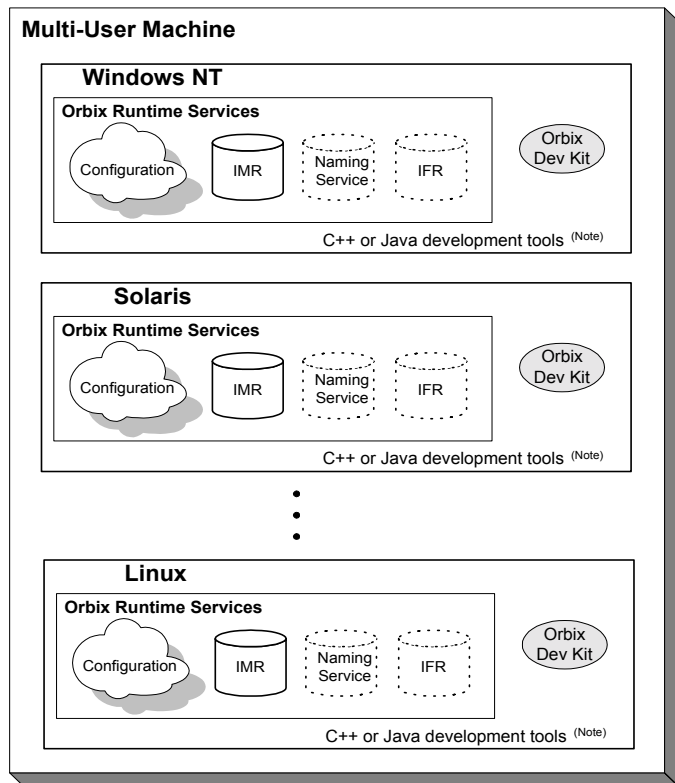
To test an application, it must first be deployed. This involves populating the necessary Application Server Platform repositories (for example, the configuration domain, location domain, and naming service), with appropriate Application Server Platform application data.

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 22. 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.



Note. C++ or Java tools must exist on each development platform.


 A dotted outline indicates an optional runtime service.

Figure 7: Multiple Independent Development and Test Environments

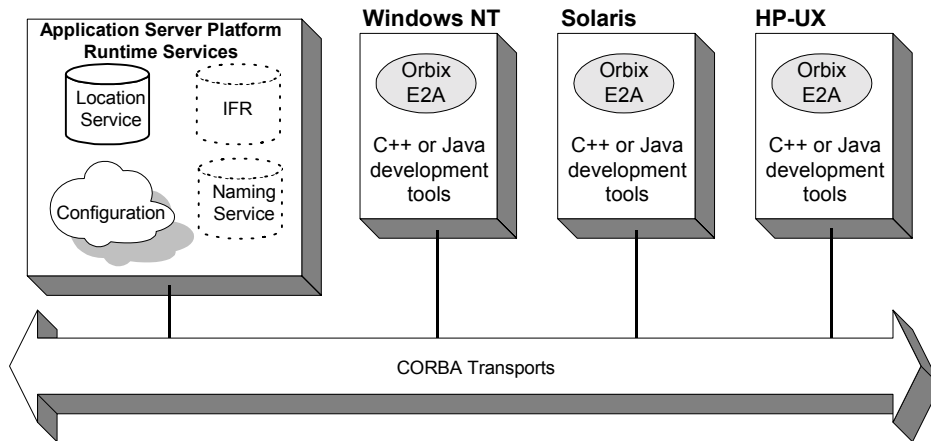
To establish independent development and test environments, first ensure that the appropriate C++ or Java libraries are present. You should then install the Application Server Platform on the desired platforms. For information on how to configure Application Server Platform runtime services in your environment (for example, a locator daemon), see [Chapter 3](#).

Distributed Development and Test Environments

Overview

Figure 8 on page 25 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 Application Server Platform runtime services in your environment. Ensure that the appropriate C++ or Java libraries are present on your development platforms. Then install the Application Server Platform developer's kit on each platform. For information on how to configure Application Server Platform runtime services such as the interface repository in your environment, see [Chapter 3](#).



A dotted outline indicates an optional runtime service.

Figure 8: *A Distributed Development and Test Environment*

Configuration Models

Overview

Application Server Platform provides two configuration mechanisms:

- [Local file-based configuration](#)
- [Configuration repository](#)

For information on managing Application Server Platform 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 Application Server Platform uses local configuration files. [Figure 9 on page 27](#) shows an example Application Server Platform environment where the configuration is implemented in local files on client and server machines.

The Application Server Platform components in [Figure 9 on page 27](#) consist of Application Server Platform management tools, the locator daemon, and configuration files that store the configuration of the Application Server Platform components. When the application server 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 Application Server Platform runtime components, but for simplicity these are not shown in [Figure 9 on page 27](#).

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.

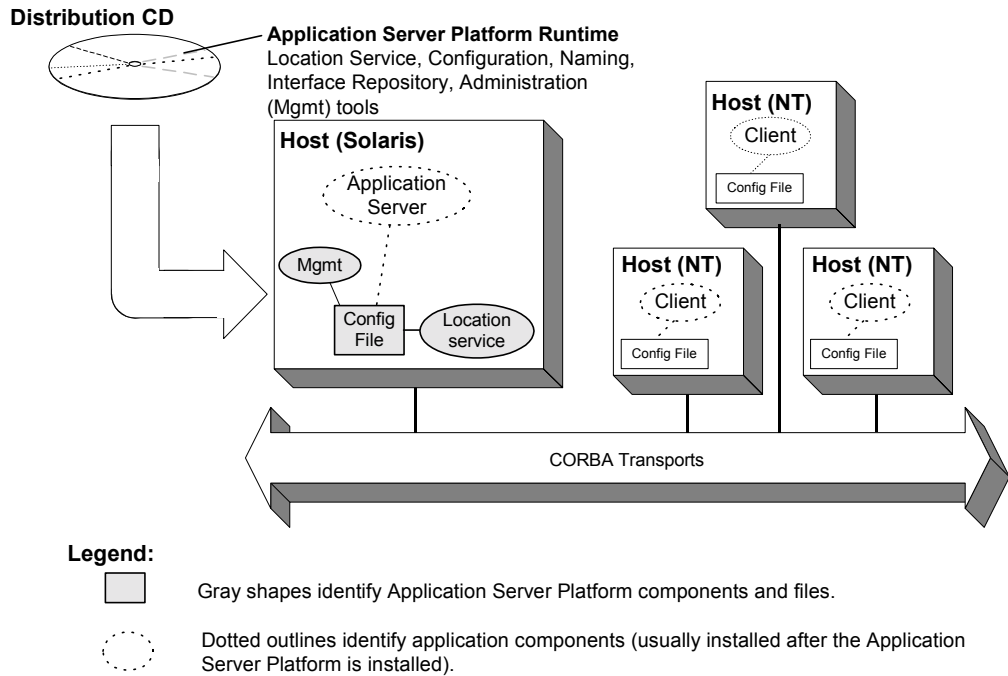


Figure 9: Application Server Platform Environment with Local Configuration

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 Application Server Platform configuration repository provides a centralized database for all configuration information.

The Application Server Platform components in [Figure 10 on page 28](#) consist of the Application Server Platform management tools, the locator daemon, and a configuration repository. The configuration repository stores the configuration for all Application Server Platform components. When the application server and clients are installed, they store their configuration in separate configuration scopes in the configuration repository. Application clients and servers also include their own Application Server Platform 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 the application server, configuration repository, and locator daemon must be highly reliable and always available to all clients and servers.

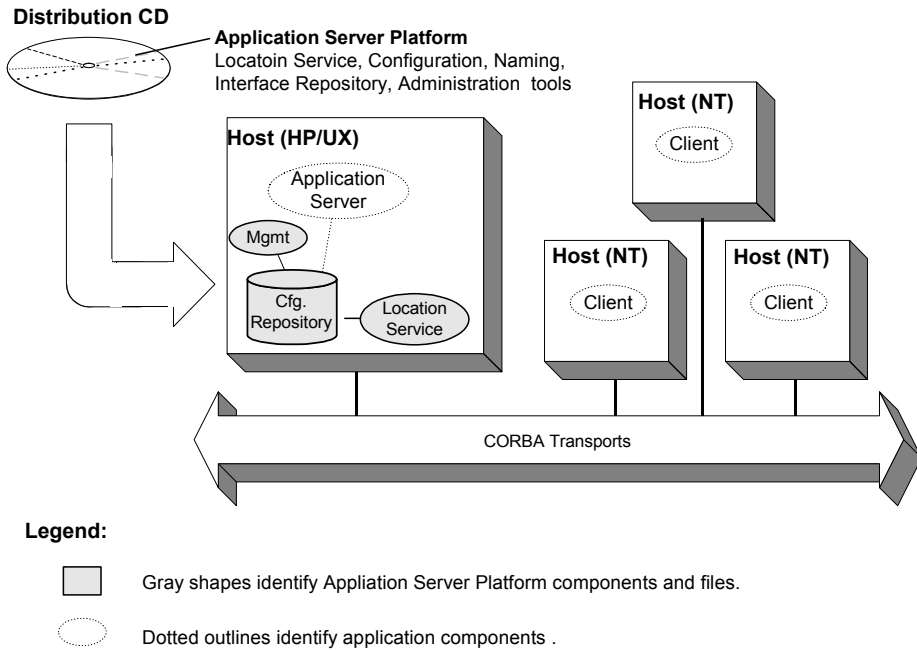


Figure 10: Application Server Platform Environment with Centralized Configuration

Getting the Most from Your Application Server Platform Environment

Overview

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

- [“Using Capabilities of Well-Designed Application Server Platform Applications” on page 30](#)
- [“Using the Right Data Storage Mechanism” on page 32](#)

Moving other Application Server Platform services (for example, a naming service), or moving application 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 Application Server Platform Applications

Application Server Platform optimizations

Like a major highway, Application Server Platform is designed to handle a lot of traffic. For example, when Application Server Platform 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 Application Server Platform 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 Application Server Platform. 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 the application 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 Application Server Platform 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

Application Server Platform provides standard storage mechanisms for storing persistent data used by Application Server Platform 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 Application Server Platform vendor for more information.

Getting the Most from Application Server Platform Configuration

Overview

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

- [Separate Application Server Platform environments](#)
- [Multiple configuration domains](#)

Separate Application Server Platform environments

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

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

Overview

There are several things you need to do in order to create an environment in which Application Server Platform applications can run:

- Install the licenses for the services you wish to run.
- Create a configuration domain and deploy services.
- Ensure that Java applications use the correct ORB classes.
- Specify configuration domains to clients and servers.

In this chapter

This chapter discusses the following topics:

Running the Orbix E2A Configure Tool	page 37
Licensing	page 42
Configuring an Application Server Platform Domain	page 44

Replicating a Services in a Domain	page 74
Starting and Stopping Application Server Platform Services	page 75
Setting Java ORB Classes	page 76
Running Sample Applications	page 77

Running the Orbix E2A Configure Tool

Overview

The Orbix E2A configure tool guides you through licensing and configuring Application Server Platform 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 E2A configure tool analyzes your installation and provides you with the options available for your system.

Prerequisites

Before you run the configure 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 configure 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 configure 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 E2A Application Server Platform installation on your system.

Syntax

To run the configure tool use the following command:

```
itconfigure [-ORBproduct_dir install_dir]
            [-ORBlicense_file license_file]
            [-ORBid Application Identifier]
            [-nogui]
            [-gui]
            [-load domain_descriptor]
            [-save filename]
            [-name domain_name]
            [-file]
            [-cfr]
            [-expert]
            [-j2ee]
            [-external]
            [-host hostname]
            [-range base_port]
            [-etc etc_dir]
            [-var var_dir]
            [-link cfr_host]
            [-tlsport tls_port]
            [-port iiop_port]
            [-replica]
            [-demos]
            [-help]
            [-libs]
```

The configure tool supports the following options:

<code>-ORBproduct_dir</code> <i>install_dir</i>	Used when Application Server Platform is installed in a non-default location and the <code>IT_PRODUCT_DIR</code> environment variable is not set.
<code>-ORBlicense_file</code> <i>license_file</i>	Used when the Application Server Platform license file is not stored in the default location and the <code>IT_LICENSE_FILE</code> environment variable is not set.
<code>-nogui</code>	Runs the configuration tool silently. This option must be used with either <code>-load</code> or <code>-j2ee</code> .
<code>-gui</code>	Runs the configuration tool's GUI(default).

<code>-load domain_descriptor</code>	Loads a preconfigured domain descriptor file. When used in conjunction with <code>-nogui</code> , this option silently deploys the local pieces of the configuration defined in the deployment descriptor.
<code>-save filename</code>	Used to save a deployment descriptor in the specified file. When used with <code>-nongui</code> , this option will not deploy the saved configuration.
<code>-name domain_name</code>	Used to specify the name of the domain. The specified name will override the name in a loaded domain descriptor.
<code>-file</code>	Used to create a file based configuration. This option will override the setting in a loaded domain descriptor.
<code>-cfr</code>	Used to create a repository based configuration. This option will override the setting in a loaded domain descriptor.
<code>-expert</code>	Causes the GUI to skip straight to Expert mode.
<code>-j2ee</code>	Creates a J2EE only configuration.
<code>-external</code>	Used to create a configuration that includes external services(default).
<code>-host hostname</code>	Used to specify the name of the domains host machine. This setting overrides the setting in a loaded domain descriptor.
<code>-range base_port</code>	Used to specify the base port number from which to begin allocating port numbers. This option is only used in conjunction with <code>-nogui</code> .
<code>-etc etc_dir</code>	Used to specify the directory where configuration information is stored.
<code>-var var_dir</code>	Used to specify the directory where database files are stored.
<code>-link cfr_host</code>	Used to specify the machine which hosts the domain's configuration repository.
<code>-tlsport tls_port</code>	Used to override the default CFR TLS port when used with <code>-link</code> .

<code>-port <i>iiop_port</i></code>	Used to override the default CFR IIOP port when used with <code>-link</code> .
<code>-replica</code>	Used to inform the tool that the machine will run replicated services.
<code>-demos</code>	Used to include the configuration needed to run the Orbix E2A demos in the domain.
<code>-help</code>	Displays an explanation of the command flags.
<code>-libs, -L</code>	Used by <code>itconfigure</code> to pass the library path supplied to the deployer, which prepends the path to the built-in path used when preparing and running Orbix services. The library path argument is a list of directories to be searched for shared libraries when a service is run. The syntax of the list is the same as the platform-specific path syntax, for example: <pre>itconfigure -load sample_dd.xml -libs /usr/my_libs:/home/me/lib -nogui</pre>

Main screen

Once the configure tool is running you should see the following screen:

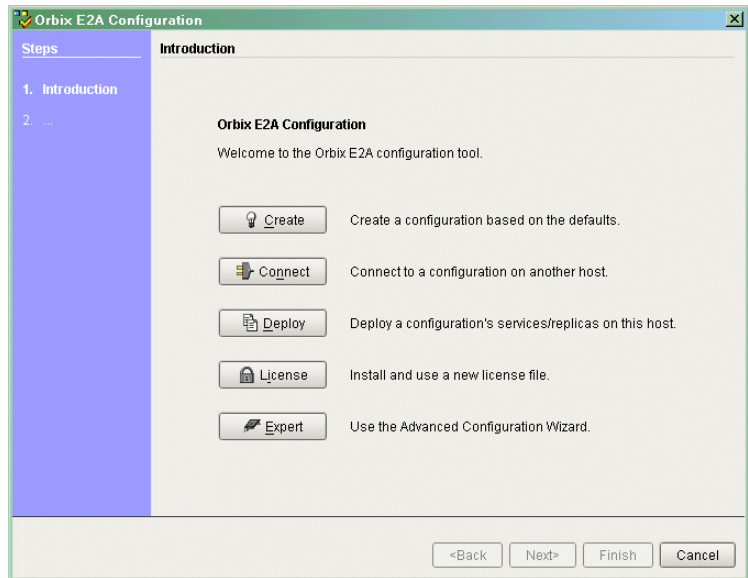


Figure 11: *Main Configuration Window*

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

Licensing

Overview

The Orbix E2A configure tool provides you with the tools to install or update a license file. The wizard allows you the option of specifying a location for your license file or installing additional licenses into the current license file.

Specifying a license file

To specify a license file:

1. From the Orbix E2A configure tool's main screen, click **License**.
2. A dialog similar to the one shown in [Figure 12 on page 42](#).

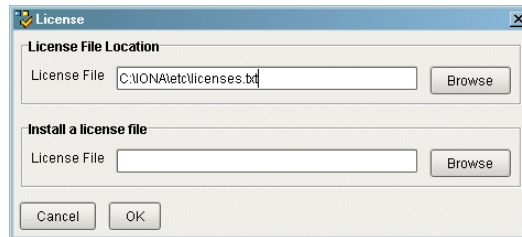


Figure 12: *Entering the License File*

3. Enter the name of the license file you wish to install in the **License File Location** line. You should have received this from your IONA representative and stored it in a secure location.
 4. Click **OK** to return to the main screen.
-

Updating an existing license file

If you have purchased additional features of the Application Server Platform or are adding functionality to a host in a deployed environment, you can update the licenses without reinstalling them.

To update an existing license file:

1. From the Orbix E2A configure tool's welcome screen, click **License**.
2. A dialog similar to the one shown in [Figure 12 on page 42](#).

3. Enter the name of the license file containing the updates in the **Install a license file** line. You should have received this from your IONA representative and stored it in a secure location.
4. Click **OK** to return the main screen.

Configuring an Application Server Platform Domain

Overview

A configuration domain contains all the configuration information used by Application Server Platform ORBs, services, and applications. The Orbix E2A configure tool configures and deploys Application Server Platform components into a configuration domain. It can also link a machine to an existing configuration domain.

Centralized domain design

The Orbix E2A 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 option is used to create a new configuration domain from scratch. It allows you 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 options allows you to deploy replicated services into a domain. It is also used to deploy services on the host machines in a domian. For more information, see [“Replicating a Services in a Domain” on page 74.](#)

- ◆ **Connect:** This option is used 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 will fail to create a domain if the configuration repository is not running or if the domain is file based.

- ◆ **Expert:** This option is used 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 E2A Administration.

In this section

This section discusses the following topics:

Create a New Domain	page 46
Connect a Client Machine to a Domain	page 54
Use Expert Mode	page 56
Localize a Preconfigured Domain	page 60
Deploy a Domain on the Remaining Hosts	page 63
Configure a Machine with no GUI	page 66
Configure a Multihomed Machine	page 67
Configure a Domain without Using the itconfigure GUI Interface	page 68
Configure ASP to Listen on a Fixed Port	page 72

Create a New Domain

Overview

The Orbix E2A 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 E2A configure tool.
2. Select **Create**.
3. You will see a screen similar to [Figure 13 on page 46](#).

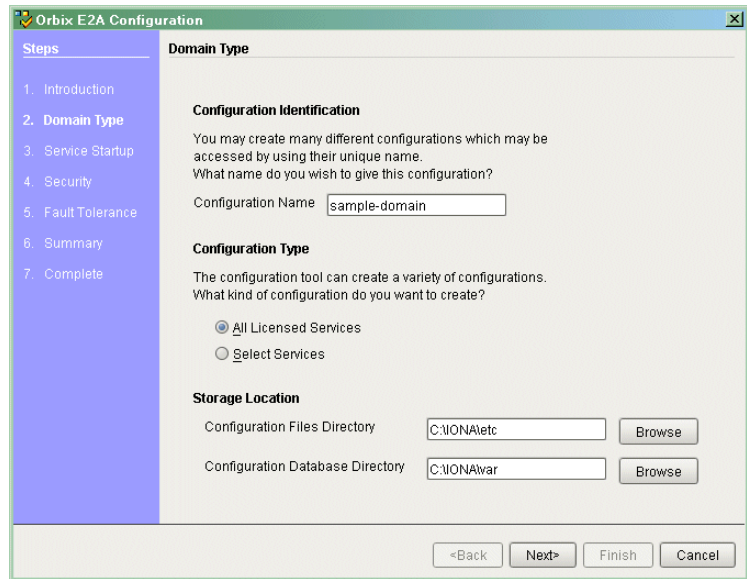


Figure 13: *Services to Deploy*

4. Specify the domain's 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 on page 47](#).

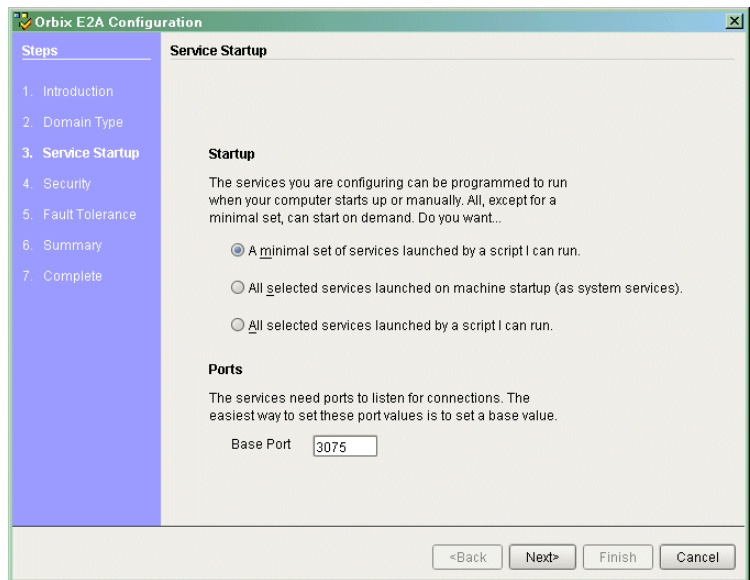


Figure 14: Startup Mode and Base Port

8. Choose one of the following options:

- ◆ **A minimal set of services launched by a script** 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** generates a script that will start all deployed services.
9. Enter a number for the **Base Port**. This is the number from which Orbix E2A 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 on page 49](#).

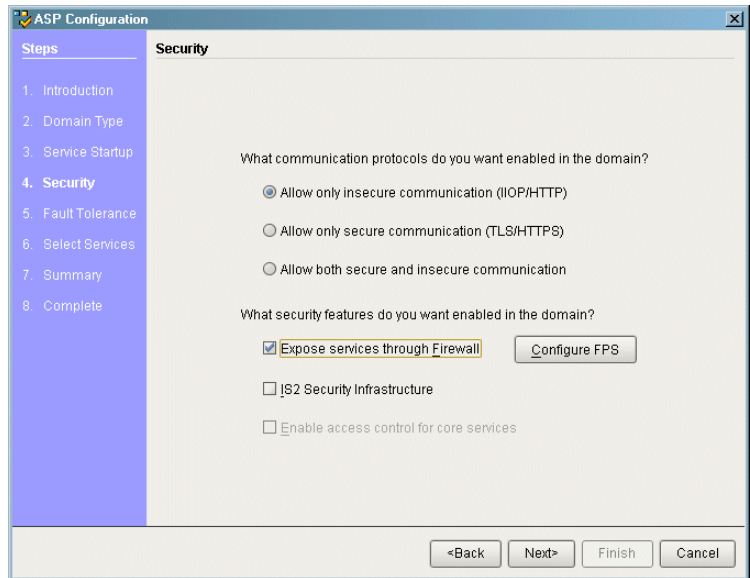


Figure 15: Setting security features

11. Select what security protocols you wish enabled.
- ◆ **Allow only 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.

- ◆ **Allow only 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.

- ◆ **Allow both secure and insecure communication** configures your system so that it can use IOP, TLS, HTTP, and HTTPS protocols.

Note: This option will automatically be selected if you chose to configure the IS2 Security Infrastructure. You will be able to select to only allow 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 *Application Server Platform 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 *Application Server Platform Security Guide*.

- Click **Next** to configure any replicas you wish to include in your domain. You should see a dialog similar to [Figure 16 on page 51](#).

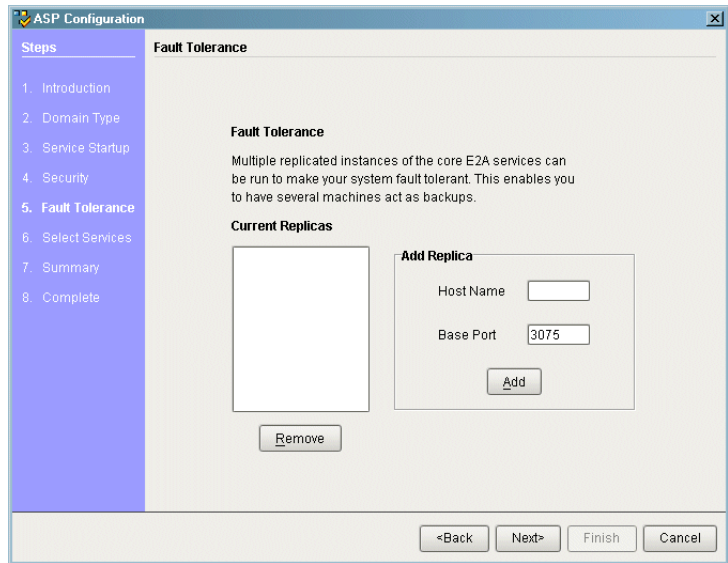


Figure 16: *Replica Configuration*

- To add a replica to the domain, enter the machine's host name and a listener port, then click on **Add**. To remove a replica from the list, highlight its hostname and click **Remove**.
- 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 on page 52](#).

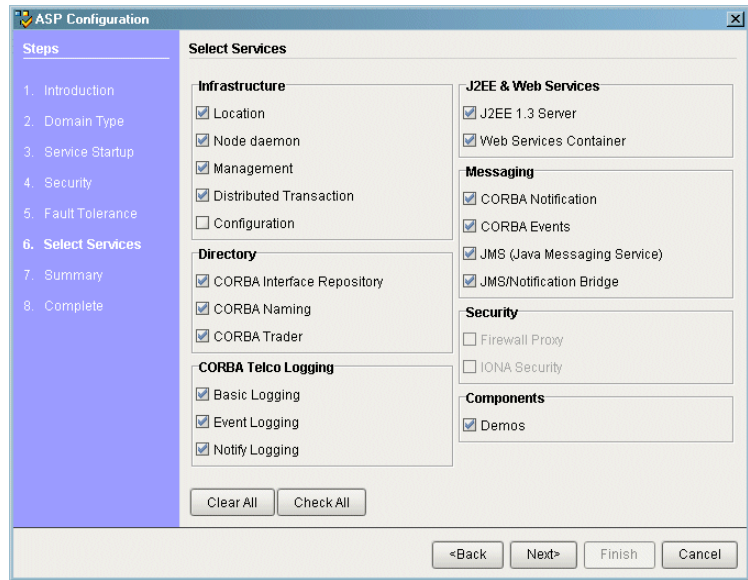


Figure 17: *Selecting services to deploy*

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

If you chose to deploy all licensed services, goto 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 53](#) should be displayed.

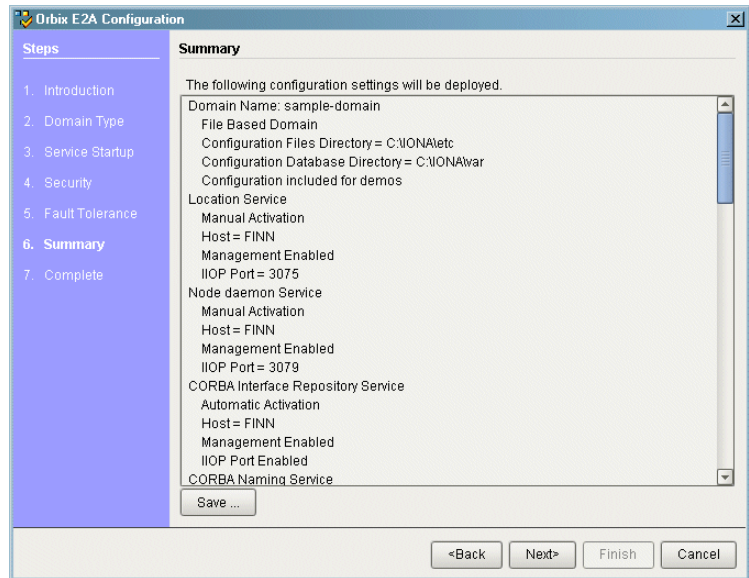


Figure 18: Configuration domain summary.

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. Once the domain is successfully created, the **Finish** button becomes available. Click it to close the tool.

Connect a Client Machine to a Domain

Overview

You will frequently need to configure machines into a domain that will only run client programs. The client programs do not need any of the CORBA services running, but they will need access to the domains configuration information.

The Orbix E2A configuration tool allows 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 host machine.

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

Procedure

To connect a new machine to an existing domain complete the following steps:

1. Select **Connect** from the configuration tool's main dialog, shown in [Figure 11 on page 41](#).
2. From the dialog shown [Figure 19 on page 54](#) in enter the hostname and port of the CFR to which you wish to connect the new machine. Also enter a location for the new machines configuration files.

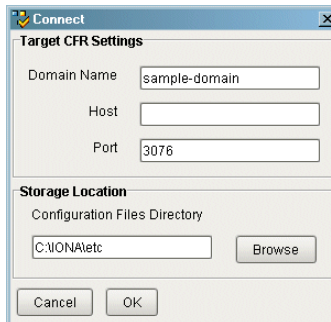


Figure 19: Select a CFR for linking

3. Click **OK** to see a summary of the configuration options.
4. Click **Next** to create the local files needed to connect the machine to the configuration domain and deploy the local services.
5. Once the machine is successfully connected to the domain the **Finish** button will be highlighted. Push it to close the dialog.

Use 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 E2A services and also allows 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 configuration tool's main dialog, shown in [Figure 11 on page 41](#).
2. In the dialog shown in [Figure 20 on page 56](#) enter a name for the domain and specify if the domain is to be file based or CFR based.

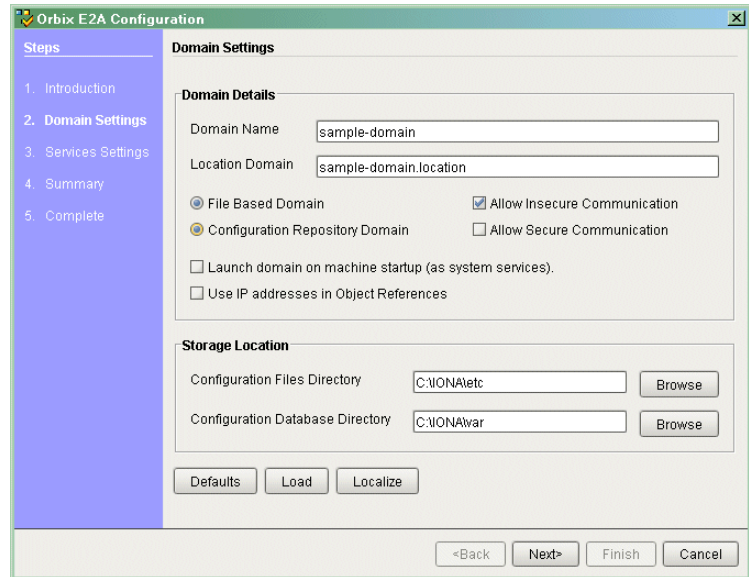


Figure 20: Advanced Domain Specification

3. 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**.
4. Select the level of security for your domain.
 - ◆ **Allow insecure communication** configures your domain to allow communication over insecure protocols such as HTTP.
 - ◆ **Allow secure communication** configures your system to allow secure communication using TLS or HTTPS.
5. To have the domain be started on system start-up place a check next to **Launch domain on machine startup (as system services)**.
6. To have services deployed in the domain using direct persistence, publishing their IP address as part of the service's object reference, place a check next to **Use IP address in Object References**.
7. To configure the default activation modes, replication settings, and optional services for the domain, click **Defaults**. A dialog similar to [Figure 21](#) is displayed.

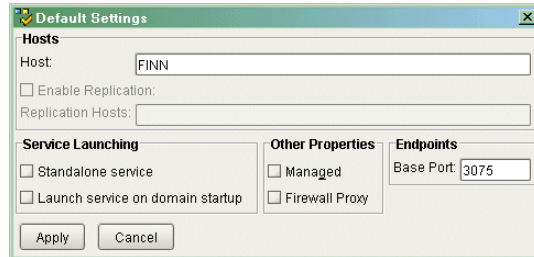


Figure 21: Configure the default domain settings

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.

- Click **Next** to select the services to deploy into the domain. A dialog similar to [Figure 22 on page 58](#) will be displayed.

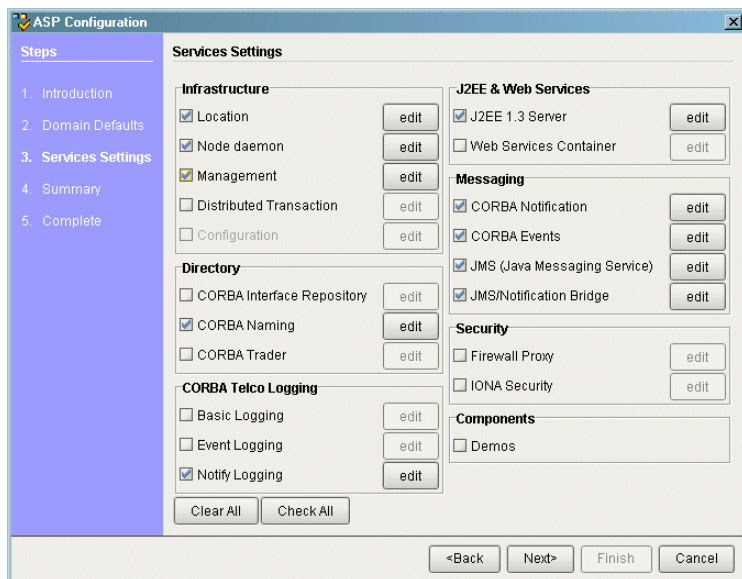


Figure 22: Advance Service Configuration

- From this dialog place a check mark next to the services you wish to deploy into the domain.

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

10. 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 23](#) on [page 59](#) will appear.

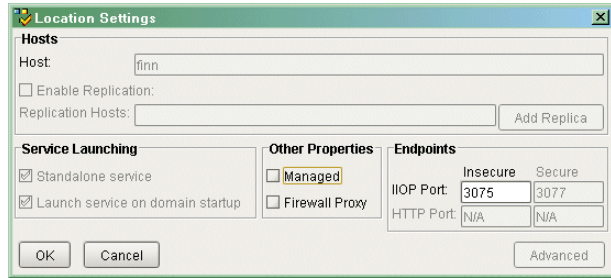


Figure 23: *Configure Advanced Service Settings*

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.

11. After selecting and configuring the desired services, click **Next** to view a summary of the configuration options you have chosen.
12. 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**.
13. If the configuration summary appears correct, click **Next** to create the domain and deploy any local services.
14. Once the domain is successfully created, the **Finish** button becomes available. Click it to close the tool.

Localize 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 E2A 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 E2A configuration tool's GUI interface to deploy a localized domain complete the following steps:

1. Select **Expert** from the configuration tool's main dialog, shown in [Figure 11 on page 41](#).
2. In the dialog shown in [Figure 20 on page 56](#) click on the **Localize** button, located under **Storage Location**.
3. Select the preconfigured domain descriptor from 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 allow 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. To have services deployed in the domain using direct persistence, publishing their IP address as part of the service's object reference, place a check next to **Use IP address in Object References**.
 9. To configure the default activation modes, replication settings, and optional services for the domain, click **Defaults**. A dialog similar to [Figure 21 on page 57](#) 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 22 on page 58](#) will be displayed.
 11. From this dialog place a check mark next to the services you wish to deploy into the domain.

Note: If you do not 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 23 on page 59](#) 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. Once the domain is successfully created, 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 localized domain from the command line by running

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

Running this command will replace the name of the host used to define the original configuration with the name of the local hostname. It will then deploy 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.

Deploy a Domain on the Remaining Hosts

Overview

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

The Orbix E2A configuration tool provides three options for deploying your domain on the remaining hosts:

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

Using the Deploy option

The most straight forward manner to deploy the local portion of a domain is to use the configure tool's **Deploy** option. To do this complete the following steps:

1. Select **Deploy** from the configuration tool's main dialog, shown in [Figure 11 on page 41](#).
2. Select the deployment descriptor from the file selection dialog.

3. A dialog similar to [Figure 24 on page 64](#) should appear. Enter the location for the configuration databases to be stored and verify the domain name. Click **OK**.

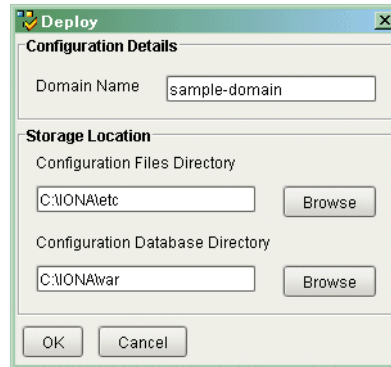


Figure 24: *Deployment details*

4. Verify that the configuration summary is accurate. If so, click **Next** to deploy the local services.
5. Once the domain is successfully deployed, the **Finish** button becomes available. Click it to close the tool.

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** form the configuration tool's main dialog, shown in [Figure 11 on page 41](#).
2. In the dialog shown in [Figure 20 on page 56](#) click on the **Load** button, located under **Storage Location**.
3. Select the domain descriptor from the file selection dialog.
4. A dialog similar to [Figure 22 on page 58](#) 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.

Configure a Machine with no GUI

Overview

You may need to occasionally configure and deploy an Application Server Platform 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 GU and choose expert mode.
2. Click on **Defaults**. This will bring up a dialog box similar to the one shown in [Figure 21 on page 57](#). In the host box, enter the name of the remote host, then click **Apply**.
3. To design the domain, follow the steps outlined in [“Use Expert Mode” on page 56](#).
4. When you reach the summary screen, click on **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.

Configure a Multihomed Machine

Overview

You may need to occasionally configure and deploy an Application Server Platform domain on a multihomed machine, that is a machine that has more than one IP addresses 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 `bill`, and an alternate hostname `ben`, the `itconfigure` command line would be:

```
itconfigure -multihome ben
```
2. Choose Expert mode and click on **Defaults**. This displays a dialog box similar to the one [Figure 21 on page 57](#). In the host box, enter the name of the alternate hostname, which should match the hostname specified by the `multihome` parameter on the command line. Click **Apply**.

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

Configure a Domain without Using the itconfigure GUI Interface

Overview

It is possible to configure a domain without using a GUI interface. To do this, you need a configuration file for itconfigure to get the information which would normally be obtained via the GUI interface. This file is an XML document. The easiest way to create this configuration file initially is by using the itconfigure GUI interface. Configure the domain as usual, and on the **Summary** screen save the configuration file. You can then cancel the option and exit.

If it is not possible to use the itconfigure GUI, use one of the following files as a starting point:

- `file.xml` (deployment descriptor for file-based domain).
- `cfr.xml` (deployment descriptor for cfr-based domain).

Required fields

Open the configuration file in your favorite text editor and check that the values in the following fields are correct for the domain you need to configure:

`dd:domain`—this is the name of the domain.

`dd:source`—this is a file for a file based domain, or `cfr` for a configuration repository based domain.

`dd:nodes`

—`name`, the hostname to be configured.

—`ip`, the IP address of the host to be configured.

`dd:profile`—for profile use the same value as for `name`.

—for `id` use the same value as you did for profile in `dd:nodes`.

Note: If you have used one of the sample files, you need to check through the rest of the file to ensure that only the services you need to deploy are included, and that the port numbers and other settings are correct. If you created the file using the itconfigure GUI, the rest of the values in the file should already be correct.

Available switches

You can now use this file to configure a domain. To do this, you need to use the following switches:

<code>-nogui</code>	Informs itconfigure not to use the GUI interface.
<code>-load <i>filename</i></code>	Informs itconfigure of the file to be loaded to read configuration information from.
<code>-etc <i>path</i></code>	Informs itconfigure of the path to store the configuration files. If its not specified, the default <code>\$IT_PRODUCT_DIR/etc</code> is used (optional).
<code>-var <i>path</i></code>	Informs itconfigure of the path to store the database and log files. It defaults to <code>\$IT_PRODUCT_DIR/var</code> (optional).
<code>-demos</code>	Informs itconfigure to include configuration for demos in this domain (optional).

For example:

```
itconfigure -nogui-load configure.xml -etc /my/config/file/path  
-var /my/database/file/path -demos
```

For a complete list of flags available to itconfigure, use:

```
itconfigure -help
```

A typical output for a successfully completed Windows based configuration is as follows:

```
ERROR System service is defined for some but not for all services
that are not started on demand.
ERROR Defaulting to: no services will be installed as system
services.
COPY RESOURCES C:\ASP-6.0\asp\6.0\templates\etc\admin TO
C:\ASP-6.0\etc\domains\sample-domain
STARTING TO PREPARE: iona_services.management
COMPLETED PREPARE: iona_services.management
STARTING TO PREPARE: iona_services.locator.csfloater
COMPLETED PREPARE: iona_services.locator.csfloater
STARTING TO RUN: iona_services.locator.csfloater
COMPLETED RUN: iona_services.locator.csfloater
STARTING TO PREPARE: iona_services.node_daemon.csfloater
COMPLETED PREPARE: iona_services.node_daemon.csfloater
STARTING TO RUN: iona_services.node_daemon.csfloater
COMPLETED RUN: iona_services.node_daemon.csfloater
START-UP MODE: on_demand
STARTING TO PREPARE: iona_services.ifr.csfloater
COMPLETED PREPARE: iona_services.ifr.csfloater
START-UP MODE: on_demand
STARTING TO PREPARE: iona_services.naming.csfloater
```

```
COMPLETED PREPARE: iona_services.naming.csfloater
START-UP MODE: on_demand
STARTING TO PREPARE: iona_services.jms
COMPLETED PREPARE: iona_services.jms
WARNING : C:\ASP-6.0\var\sample-domain\webservices does not
        exist
MKDIR C:\ASP-6.0\var\sample-domain\webservices
COPY RESOURCES C:\ASP-6.0\asp\6.0\templates\var\webservices TO
        C:\ASP-6.0\var\sample-domain\webservices
WARNING : C:\ASP-6.0\etc\domains\sample-domain\certs does not
        exist
MKDIR C:\ASP-6.0\etc\domains\sample-domain\certs
COPY RESOURCES C:\ASP-6.0\asp\6.0\templates\etc\webservices TO
        C:\ASP-6.0\etc\domains\sample-domain\certs
COPY RESOURCES C:\ASP-6.0\asp\6.0\templates\etc\j2ee TO
        C:\ASP-6.0\etc\domains\sample-domain
COPY RESOURCES C:\ASP-6.0\asp\6.0\templates\var\j2ee TO
        C:\ASP-6.0\var\sample-domain
STARTING TO SHUTDOWN: iona_services.node_daemon.csfloater
COMPLETED SHUTDOWN: iona_services.node_daemon.csfloater
STARTING TO SHUTDOWN: iona_services.locator.csfloater
COMPLETED SHUTDOWN: iona_services.locator.csfloater
Configuration completed successfully
```

You can view the log file in the `<install-dir>\etc\log` directory.

Configure ASP to Listen on a Fixed Port

Overview

The `simple_persistent` server from the ASP demos, available in `$IT_PRODUCT_DIR/asp/6.0/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.

Application Server Platform provides IONA proprietary POA policies that allow you to instruct a POA to listen on a fixed port. These policies are called `WellKnownAddressing` policy and `DirectPersistent` policy. You can set these policies 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 *Application Server Platform 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;
```


Where:

`<FQPN>` is the Fully Qualified POA Name. The ASP 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 ASP 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 ASP 6.0 Java ORB, then you must set the direct persistent and well known addressing policies programmatically.

Replicating a Services in a Domain

Overview

You can configure machines to host 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: In order to configure a machine to host replicas, you must specify it when you create the configuration domain (see [page 51](#)).

Procedure

To deploy a replica, follow these steps:

1. Copy the generated deployment descriptor from the host machine to the replica you wish to configure. The deployment descriptor will have the name `domain_name_dd.xml`. For example, the domain descriptor for domain Cuculain will be `Cuculain_dd.xml`.
2. Run the configure tool. From the main dialog, click **Deploy**.
3. Select the domain descriptor from the file selection dialog. 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 Application Server Platform Services

Overview

The configure 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 Application Server Platform Services Manually” on page 203](#)).

Starting Application Server Platform services

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

```
config-dir/bin/start_domain-name_services
```

Stopping Application Server Platform services

You can stop Application Server Platform services manually with the following command:

```
config-dir/bin/stop_domain-name_services
```

Setting an Application Server Platform environment

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

```
config-dir/bin/domain-name_env
```

Setting Java ORB Classes

Overview

In order to run Java applications, Application Server Platform must use its own ORB classes instead of Sun ORB classes. To ensure that Application Server Platform 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.ionacorba.art.artimpl.ORBImpl
org.omg.CORBA.ORBSingletonClass=com.ionacorba.art.artimpl.ORBSingleton
```

Using Java system properties

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

```
java -Dorg.omg.CORBA.ORBCLASS=com.ionacorba.art.artimpl.ORBImpl
-Dorg.omg.CORBA.ORBSingletonClass=com.ionacorba.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 configure 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 Application Server Platform Environment

In this part

This part contains the following chapters:

Configuring Application Server Platform Applications	page 81
Managing Persistent CORBA Servers	page 101
Configuring Scalable Applications	page 127
Managing the Naming Service	page 147
Managing an Interface Repository	page 161
Orbix E2A Firewall Proxy Service	page 169
Managing CORBA Service Databases	page 175
Setting Up Application Server Platform Logging	page 185

Configuring Application Server Platform Applications

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

The Application Server Platform 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

page 83

Locating the Configuration Domain	page 85
Obtaining an ORB's Configuration	page 87
Managing Configuration Domains	page 98

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 26](#) summarizes how an initializing ORB obtains its configuration information in a repository-based system, where services are distributed among various hosts.

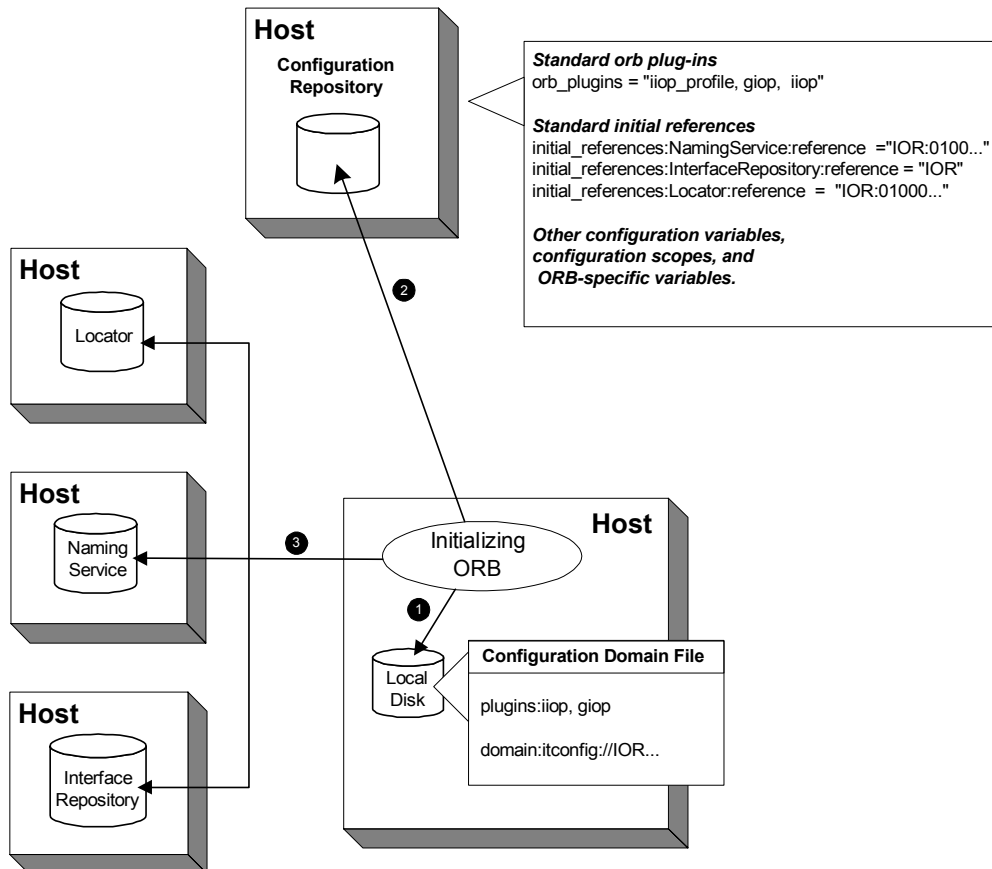


Figure 25: How an Application Server Platform Application Obtains its Configurations

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/iora
```

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 387](#) 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. Application Server Platform services such as the naming service have their own configuration scopes. Application Server Platform 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 26](#) shows how a configuration domain might be organized into several scopes:

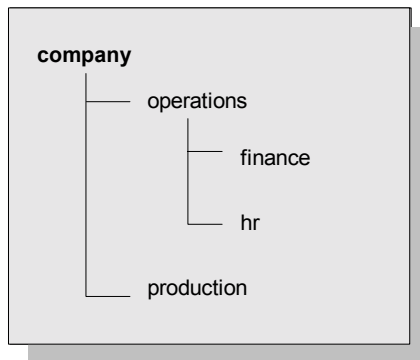


Figure 26: *Hierarchy of Configuration Scopes*

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 Application Server Platform configuration information defines the hierarchy of scopes shown in [Figure 26 on page 87](#):

```
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 26 on page 87](#):

```

1 company
  {
    # company-wide settings

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

    # Standard initial references.
    initial_references:RootPOA:plugin = "poa";
    initial_references:ConfigRepository:reference
      = "IOR:010000002000...00900";
    initial_references:InterfaceRepository:reference
      = "IOR:010000002000...00900";

    # Standard IIOP configuration
    policies:iiop:buffer_sizes_policy:max_buffer_size = -1

2 operations
  {
    # Settings common to both finance and hr

    # limit binding attempts
    max_binding_iterations = "3";

3 finance
  {
    # finance-specific settings

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

  }

```

Example 1: *A File-Based Configuration*

```

4     hr
      {
        # hr-specific settings

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

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

    }

  } # close company scope

```

Example 1: A File-Based Configuration

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 name. 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#166;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 Value

```
policies:{iiop#166;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.

Configuration Value

```
policies:{iiop#166;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 IOP 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. The IONA ASP CORBA ORBs support fragmentation and are fully compliant with the 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. Application Server Platform 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

Application Server Platform 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: <ul style="list-style-type: none"> The <code>itconfigure-generated start_domain-name_services</code> script starts the configuration repository and other domain services. <code>itconfig_rep run</code> starts the configuration repository only.
Stop the configuration repository	<code>itadmin config stop</code>
View configuration repository contents	<code>itadmin config dump</code>
List all replicas of the configuration repository.	<code>itadmin config list_servers</code>
Create scope	<code>itadmin scope create</code>
List scopes	<code>itadmin scope list</code>
View scope contents	<code>itadmin scope show</code>
Create namespace	<code>itadmin namespace create</code>
List namespaces	<code>itadmin namespace list</code>
View namespace contents	<code>itadmin namespace show</code>
Remove namespace	<code>itadmin namespace remove</code>
Create variable	<code>itadmin variable create</code>
View variable	<code>itadmin variable show</code>
Modify variable	<code>itadmin variable modify</code>
Remove variable	<code>itadmin variable remove</code>

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):

<code>-ORBdomain_name</code>	Specifies the configuration for <code>itadmin</code> . This is typically a temporary file-based configuration created for this purpose only.
<code>-ORBadm_in_domain_name</code>	Specifies the configuration domain repository to modify.
<code>-ORBadm_in_config_domains_dir</code>	Specifies the directory in which to find 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
        -ORBadm_in_domain_name acme-products
        variable create -type list
        -value iiop_profile,giop,iiop orb_plugins
```


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, application 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	page 103
Server Environment Settings	page 106
Managing a Location Domain	page 110
Using Direct Persistence	page 121

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 -orbnam my_app.server_orb \  
    banking_service \  
itadmin poa create \  
    banking_service/account -transient \  
itadmin poa create -orbnam 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 119](#).

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

Application Server Platform 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 139](#).

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	page 107
UNIX Environment Settings	page 108

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` (`rxwxr-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.

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 Application Server Platform has been installed and configured to run within your network environment. For more on configuring Application Server Platform, see [Chapter 3 on page 35](#).

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 Application Server Platform location domain, the node daemon is responsible for activating and managing server processes. Every host running an application 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 application servers starting on this machine. When an application 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`.

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
<code>process list</code>	Lists the names of all target processes registered in the location domain.
<code>process show</code>	Lists the attributes of server processes registered with the locator daemon.
<code>orbname list</code>	Lists all ORB names in the location domain.
<code>orbname show</code>	Lists the attributes of ORB names registered with the locator daemon.
<code>poa list</code>	Lists the names of all POAs in the location domain.
<code>poa show</code>	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 15 on page 233](#).

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
<code>process modify</code>	Modifies the specified process entry.
<code>orbname modify</code>	Associates an ORB name with the specified process name.
<code>poa modify</code>	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
<code>process remove</code>	Removes a process entry.
<code>orbname remove</code>	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.
<code>poa remove</code>	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 Application Server Platform 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 allows Application Server Platform to bypass the locator daemon when resolving persistent object references or contacting Application Server Platform services.

In this section

This section discusses the following topics:

CORBA Application Servers	page 122
Application Server Platform Services	page 125

CORBA Application Servers

In general, a CORBA application relies 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.

Requirements

Two requirements apply:

- The configuration must contain a well-known address configuration variable, with the following syntax:

```
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 136](#)).

- 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*).

Example

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

```
MyConfigApp {
  ...
  wka: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.ionacorba.*;
import com.iona.ITCORBA.*;
import com.iona.ITPortableServer.*;

// 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,
        persistent_mode_policy_value),
    global_orb.create_policy(
        WELL_KNOWN_ADDRESSING_POLICY_ID.value,
        well_known_addressing_policy_value),
};
...
```

Application Server Platform Services

In general, Application Server Platform 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 Application Server Platform 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, then `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.

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 errant backhoe 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 are liable to 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 run afoul of service bottlenecks and outages.

In this chapter

This chapter contains the following sections:

Active Connection Management	page 129
Fault Tolerance and Replicated Servers	page 131
Building a Replicated Server	page 139
Replicating Domain Services	page 145

Active Connection Management

Overview

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

You can control active connection management in Application Server Platform 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 Application Server Platform 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:iop:outgoing_connections:hard_limit specifies the maximum number of outgoing (client-side) connections permitted to IOP. IOP does not allow new outgoing connections above this limit. This variable defaults to -1 (disabled).

plugins:iop:outgoing_connections:soft_limit specifies the number of connections at which IOP 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:iop:outgoing_connections:hard_limit=1024;
```

The following `itadmin` command sets this variable:

```
itadmin variable create -type long -value 1024  
    plugins:iop: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 Application Server Platform, 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 Application Server Platform routes the requests to one of the member replicas. The actual routing to a replica is transparent to the client.

Benefits

Application Server Platform 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, Application Server Platform 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 Application Server Platform provides out-of-the-box round robin and random load-balancing strategies. The Application Server Platform load-balancing framework is pluggable, so you can easily implement your own strategies.

About Replicated Servers

Overview

The Application Server Platform 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 11](#)), Application Server Platform 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 Application Server Platform 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 Application Server Platform 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 Application Server Platform 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 140](#).
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 `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 Application Server Platform 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_server/server \
  -node_daemon daemon_name \
  -startupmode on_demand \
  -args "--ORBname demos.clustering.server_1 server_1"
  \
  demos.clustering.server_process_1
%
% process create \
  # same arguments as before \
  ... \
  -args "--ORBname demos.clustering.server_2 server_2"
  \
  demos.clustering.server_process_2
%
% process create \
  ... same arguments as before \
  -args "--ORBname demos.clustering.server_3 server_3"
  \
  demos.clustering.server_process_3
%
```

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

- ◆ The path name for the server executable.

- ◆ The name of the node daemon to start the server.

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

- ◆ A list of command line arguments passed to the server via the `-args` argument. These arguments include a unique ORB name that is associated with each server replica.
3. Call `orbnam 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:

```
% orbnam create \
    -process demos.clustering.server_process_1 \
    demos.clustering.server_1
% orbnam create
    -process demos.clustering.server_process_2 \
    demos.clustering.server_2
% orbnam 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.iior &
$ ./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

Application Server Platform 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:

```

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

Application Server Platform 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, including the locator daemon, naming service, trader, 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`, etc.) 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.

For details on replicating services in a domain, see [“Replicating a Services in a Domain” on page 74](#).

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

page 149

Controlling the Naming Service	page 152
Building a Naming Graph	page 153
Maintaining a Naming Graph	page 158
Managing Object Groups	page 159

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 263](#). 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 27](#).

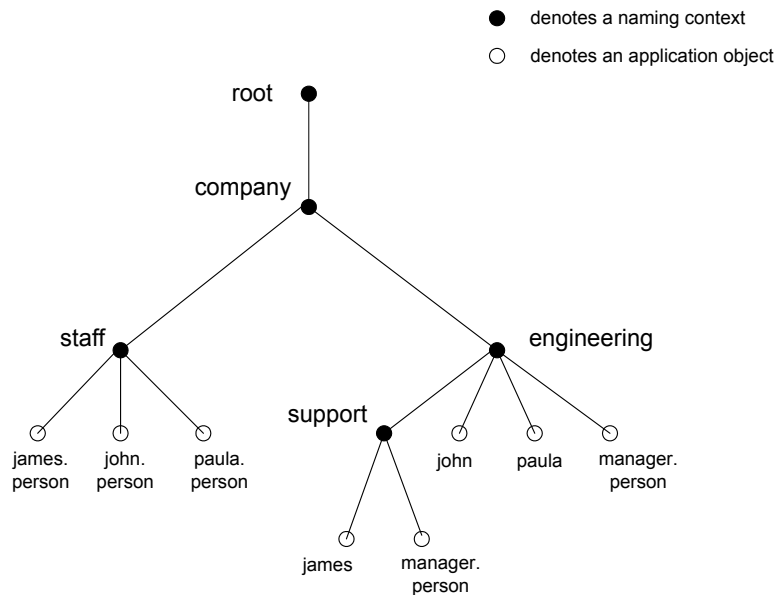


Figure 27: Naming Context Graph

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 27](#), 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:0000000000002356702b4944c3a6f6d672e6f7267...."
```

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 27 on page 153](#), 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 27 on page 153](#).

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
<code>ns list</code>	List all bindings in a naming context
<code>ns resolve</code>	Print the object reference for the application object or naming context to which a name is bound.
<code>ns unbind</code>	Unbind the binding for an object reference.
<code>ns remove</code>	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 159](#)).
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 268 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.

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.

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*.

In this chapter

This chapter contains the following sections:

Controlling the Interface Repository Daemon	page 163
Managing IDL Definitions	page 164

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 in order 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:

Browsing Interface Repository Contents	page 165
Adding IDL Definitions	page 167
Removing IDL Definitions	page 168

For a complete reference of the commands used to manage the interface repository, see [“Repository Management” on page 277](#).

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 want 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`.

Example

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
```


Orbix E2A 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	page 170
Using the Firewall Proxy Service in a Deployed Environment	page 171
Known Restrictions	page 174

Firewall Proxy Service Functionality

Overview

The main goal of the FPS 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

Once 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:

<code>ENABLE</code>	This is the default behavior of the firewall proxy service plug-in. A POA with its <code>INTERDICTION</code> policy set to <code>ENABLE</code> will be proxified.
<code>DISABLE</code>	This setting tells the firewall proxy service plug-in to not proxify the POA. POAs with their <code>INTERDICTION</code> policy set to <code>DISABLE</code> 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.ionacorba.IT_FPS.*;

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

Policy[] policies = new Policy[1];
policies[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 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_Colloc` from the `client_binding_list` configuration parameter.
- TLS is not supported by the firewall proxy service. This means that the firewall proxy service does not work with Iona's IS2 security infrastructure or any other systems that use TLS.
- The Orbix E2A Application Server does not support the firewall proxy service. The J2EE portion of your systems cannot be hidden behind a proxy.
- The runtime components of the firewall proxy service are not included in the Orbix E2A Application Server Platform J2EE Edition.

Managing CORBA Service Databases

This chapter explains how to manage databases that store persistent information about Application Server Platform services. It explains how to use the Berkeley DB database management system embedded in Application Server Platform.

A number of Application Server Platform services maintain persistent information (for example, the locator daemon, and naming service). By default, these Application Server Platform 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 back-ups, recoveries and log files.

In this chapter

This chapter contains the following sections:

Berkeley DB Environment	page 177
Performing Checkpoints	page 178
Managing Log Files	page 179
Performing Online Back-Ups	page 181

Berkeley DB Environment

Overview

A Berkeley DB environment consists of a set of database files and log files. In Application Server Platform, 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 Application Server Platform 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 Application Server Platform 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 accidentally 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 Application Server Platform 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 *Application Server Platform 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 289](#).

Managing Log Files

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 needs to be to the power of 2. For more information, see the section on the `plugins:shmiop` namespace in the *Application Server Platform Configuration Reference Guide*.

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 in one of the following ways:

Using configuration settings

By default, each Application Server Platform 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.iior  
itadmin pss_db delete_old_logs env-home/env.iior
```

For more information, see [“Persistent State Service” on page 289](#).

WARNING: Deleting old transaction log files can make recovery from a catastrophic failure impossible. See [“Performing a Recovery” on page 182](#).

Performing Online Back-Ups

Overview

A Berkeley DB environment can be backed up while it is in use. However, transaction log files should never be archived or deleted while performing a back-up.

If you are using periodic deletion of old log files (the default behavior) or periodic archival of old log files, use the `itadmin pss_db pre_backup` command before the back-up, and the `itadmin pss_db post_backup` command when the back-up is complete. `pre_backup` disables periodic log archival/deletion, and `post_backup` re-enables periodic log archival/deletion. For more information, see [“Persistent State Service” on page 289](#).

Back-up types

You can perform two forms of back-ups:

- full back-up - Perform a back-up of the data files, then the log files. The order is very important. After a successful full back-up, you can discard older full and incremental back-ups.
- incremental back-up - Perform a back-up of the log files that have changed or have been created since the last full or incremental back-up.

Note: If you wish to use incremental back-ups, do not automatically delete old log files.

Performing a Recovery

Overview

Each time you start an Application Server Platform service that uses Berkeley DB, the service performs a *normal recovery*. If the service was killed 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: you must perform a *catastrophic recovery*.

Case 1: Data files are missing or corrupted

If the data files are missing or corrupted, perform the following steps:

1. Backup your log files.
2. Delete any corrupted data files.
3. Replace your environment with the latest full back-up of this environment.
4. In order (from the oldest to the newest), copy the log files from each incremental back-up performed after this full back-up to the `logs` directory.

If you have never performed a back-up of the data files, copy all log files since the creation of the environment.

5. Copy the log files backed up in step 1 to the `logs` directory.
6. Set the following configuration variable to `true`:

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

7. Start the Application Server Platform services.
8. Set the following configuration variable to `false`:

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

Case 2: Log files are missing or corrupted

If the log files are missing or corrupted, perform the following steps:

1. Delete the corrupted log files.

2. Replace you environment with the latest full back-up of this environment.
3. In order (from the oldest to the newest), copy the log files from each incremental back-up performed after this full back-up to the `logs` directory.

If you have never backed up the data files, copy all log files since the creation of the environment.

4. Set the following configuration variable to `true`:

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

5. Start the Application Server Platform services.
6. Set the following configuration variable to `false`:

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

Note: The environment is then in the state it was when the last archived log file was written.

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

Setting Up Application Server Platform Logging

Application Server Platform 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 Application Server Platform Windows NT Services, refer to [“Logging Application Server Platform Windows Services”](#) on page 359.

In this chapter

This chapter covers the following topics:

Setting Logging Filters	page 187
Setting Logging for J2EE	page 189
Registering Log4J with JMX	page 191
Logging Subsystem	page 193

Logging Severity Levels	page 195
Redirecting Log Output	page 197

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 Application Server Platform subsystem that reports the messages (see [Table 6 on page 193](#)). The `severity` field indicates the severity levels that are logged by that subsystem (see [Table 7 on page 196](#)).

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
```

Setting Logging for J2EE

Overview

By default, the Orbix E2A Application Server logging is switched on—logging all fatal errors, other errors and warnings.

To change the level of logging the syntax of the `event_log:filters` configuration item is:

```
event_log:filters = 'filter_spec1','filter_spec2',...
filter_spec = {' (subsystem_id | `*') `=' priority_list `}'
priority_list = priority1`,`priority2,... | `*' )
priority = "INFO_ALL" | "INFO_LO[W]" | "INFO_HI[GH]"
          | "INFO_MED[IUM]" | "WARN[ING]" | "ERR[OR]" | "FATAL[_ERROR]"
          | "ALL_EVENTS" | "NO_EVENTS"
```

The following example:

```
"{IT_POA=ERROR,FATAL},{IT_CORE=*},{*=WARN,ERR,FATAL}"
```

sets the POA filter to `LOG_ERROR` and `LOG_FATAL_ERROR`, sets the ORB core filter to `LOG_ALL_EVENTS` and sets the default filter to `LOG_WARNING`, `LOG_ERROR` and `LOG_FATAL_ERROR`.

Turning logging off

To turn logging off, you can either change the level of logging to `NO_EVENTS` or you can remove the entry `local_log_stream` from the `orb_plugins` reference in the main Orbix E2A Application Server configuration scope and any other scope where it is defined.

Management

By default, the Orbix E2A Application Server management is switched on. This allows Orbix E2A Administrator to see all Orbix E2A Application Server instances. To turn this feature off, change the following entry in the `<domain-name>.cfg` file:

```
ipas:management:enabled="true";
```

to

```
ipas:management:enabled="false";
```

and remove all references to `it_mgmt` from `orb_plugins` references.

Servlet engine logging

To turn on verbose servlet engine logging, you must specify the `IT_SERVLET_ADAPTER` sub-system. For example:

```
" {IT_SERVLET_ADAPTER=*} "
```

Changing the IONA Administrator's web console port

In order to change the port number of the IONA Administrator web console, change the following line in the `ipa.cfg` file:

```
web_server:port_number = "3085";
```

Enabling JMX Reference Implementation's HTTP adaptor

The JMX RI HTTP adaptor is used for debugging any MBeans that you want to develop. To enable the JMX RI HTTP adaptor for loading into the Orbix E2A Application Server, set the following lines in the `<domain-name>.cfg` file:

```
ipas:jmx:httpd:enabled="true";  
ipas:jmx:httpd:port="8083";
```

Registering Log4J with JMX

Overview

If you are writing J2EE applications and using Log4J version 1.2.6 with the Application Server Platform, you can modify the Log4J configuration at runtime using the IONA Administration Console. For example, you can change the logging level at any time. Log4J comes with beta JMX classes which are registered using the following code sample.

Registering Log4J

Register Log4J as follows:

Example 2: *Registering Log4J with JMX*

```
public void enableManagement()
{
    Log.debug(this, "Entered enableManagement");

    if (loggerMBean == null)
    {
        try
        {
            loggerMBean = new HierarchyDynamicMBean();
            IT_IIOPAdaptorServer adaptorServer =
                (IT_IIOPAdaptorServer)com.iona.management.jmx_iiop.IT_Dynamic
                Loading.getDefaultIIOPAdaptorServer();
            loggerMBeanName = new ObjectName("iPAS:name=Log4j");
            MBeanServer mBeanServer = adaptorServer.getMBeanServer();
```

Example 2: *Registering Log4J with JMX*

```
        if (mBeanServer.isRegistered(loggerMBeanName))
        {
            try
            {
                adaptorServer.removeFromRootMBean(loggerMBeanName);
                mBeanServer.unregisterMBean(loggerMBeanName);
                Log.info(this, "UnRegistered" +
loggerMBean.toString());
            }
            catch (Exception ex)
            {
                Log.error(this, "Exception unregistering Log4J
mBeans", ex);
            }
        }

        mBeanServer.registerMBean(loggerMBean, loggerMBeanName);
        adaptorServer.addToRootMBean(
            loggerMBeanName, // ObjectName
            "Log MBeans"); // Attribute name
        Log.info(this, "Registered"+loggerMBean.toString());
    }
    catch (Exception ex)
    {
        System.out.println("Exception caught while initializing
MBean for " + "iBankAccount: " + ex);
    }
}

Log.debug(this, "Exited enableManagement");
}
```


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, Application Server Platform logs warnings, errors, and fatal errors for all subsystems.

Table 6: *Application Server Platform 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).

Table 6: *Application Server Platform Logging Subsystems*

Subsystem	Description
IT_TS	Threading/synchronization package.

Logging Severity Levels

Overview

Application Server Platform 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 Application Server Platform 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 Application Server Platform encounters an error. The Application Server Platform 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 Application Server Platform 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 Application Server Platform logging severity levels.

Table 7: *Application Server Platform Logging Severity Levels*

Severity Level	Description
INFO_LO[W]	Low verbosity informational messages.
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 Application Server Platform 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",];
```

Part III

Reference

In this part

This part contains the following chapters:

Starting Application Server Platform Services	page 201
Managing Application Server Platform Services With itadmin	page 213

Starting Application Server Platform Services

This chapter describes commands that start Application Server Platform services. For information on starting Application Server Platform services as Windows NT services, see [Appendix A on page 349](#).

In this chapter

This chapter contains the following sections:

Starting and Stopping Configured Services	page 202
Starting Application Server Platform Services Manually	page 203
Stopping Services Manually	page 212

Starting and Stopping Configured Services

Start and stop scripts

The Orbix E2A Application Server Platform configure tool generates two scripts that start and stop all configured Application Server Platform 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 Application Server Platform services you configured using the configure tool. For example, given a domain name of `AcmeServices`, the following command starts all services on Windows:

```
start_AcmeServices_services.bat
```

Start-up order

Application Server Platform 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 Application Server Platform Services Manually

Application Server Platform 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:

Command	Starts
<code>itconfig_rep run</code>	Configuration repository
<code>itlocator run</code>	Locator daemon
<code>itnode_daemon run</code>	A node daemon
<code>itnaming run</code>	Naming service database
<code>itifr run</code>	Interface repository
<code>itevent run</code>	Event service
<code>itnotify run</code>	Notification service

Table 8: *Commands to Manually Start Application Server Platform Services.*

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 Application Server Platform 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

<p><code>-ORBdomain_name</code> <i>cfr-domain-name</i></p>	<p>The configuration repository's domain file name, which is generated when you create the domain. The generated configuration domain file has the name <i>cfr-domain-name.cfg</i>.</p> <p>For example, given configuration domain <code>acmeproducts</code>, the configuration repository initializes itself from <code>cfr-acmeproducts.cfg</code>.</p>
<p><code>-ORBname</code> <i>ORB-name</i></p>	<p>Directs the initializing configuration repository to retrieve its configuration from the specified configuration scope.</p> <p>By default, this is the <code>config_rep</code> scope. Use the <code>-ORBname</code> argument to specify a different configuration scope. For example:</p> <pre>itconfig_rep -ORBname config_rep.config2 run</pre>

`-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 Application Server Platform 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

<code>-ORBname</code> <i>ORB-name</i>	Directs the initializing node daemon to retrieve its configuration from the specified configuration scope.
	By default, this is the <code>iona_services.node_daemon</code> scope. Use the <code>-ORBname</code> argument to specify a different configuration scope. For example:
	<pre>itnode_daemon -ORBname iona_services.node_daemon.nd2 run</pre>
<code>-background</code>	Runs the node daemon in the background. Control returns to the command line only after the service successfully launches. If you omit the <code>-background</code> argument, the node daemon runs in the foreground. You can abbreviate this argument to <code>-bg</code> . For example:
	<pre>itnode_daemon run -bg</pre>
	The <code>-background</code> argument is especially useful in scripts that start multiple services. It guarantees that services always launch in the same sequence as the script specifies.
<code>-ORBsecure_directories</code>	Specifies a list of secure directories in which the node daemon launches processes. This overrides the path specified for the registered process. For example:
	<pre>itnode_daemon -ORBsecure_directories [c:\Acme\bin,c:\my_app]</pre>
	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 Application Server Platform 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 Application Server Platform 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

<code>-ORBname</code> <i>ORB-name</i>	<p>Directs the initializing interface repository to retrieve its configuration from the specified configuration scope.</p> <p>By default, this is the <code>ifr</code> scope. Use the <code>-ORBname</code> argument to specify a different configuration scope. For example:</p> <pre>itifr -ORBname ifr.ifr2 run</pre>
<code>-background</code>	<p>Runs the interface repository in the background. Control returns to the command line only after the service successfully launches. If you omit the <code>-background</code> argument, the interface repository runs in the foreground. You can abbreviate this argument to <code>-bg</code>. For example:</p> <pre>itifr run -bg</pre> <p>The <code>-background</code> argument is especially useful in scripts that start multiple services. It guarantees that services always launch in the same sequence as the script specifies.</p>

itevent run**Synopsis**

```
itevent [-ORBname ORB-name] run [-background]
```

Description

Starts the event service. The event service must already be configured in your Application Server Platform 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

<code>-ORBname</code> <i>ORB-name</i>	<p>Directs the initializing event service to retrieve its configuration from the specified configuration scope.</p> <p>By default, this is the <code>event</code> scope. Use the <code>-ORBname</code> argument to specify a different configuration scope. For example:</p> <pre>itevent -ORBname event.event2 run</pre>
<code>-background</code>	<p>Runs the event service in the background. Control returns to the command line only after the service successfully launches. If you omit the <code>-background</code> argument, the event service runs in the foreground. You can abbreviate this argument to <code>-bg</code>. For example:</p> <pre>itevent run -bg</pre> <p>The <code>-background</code> argument is especially useful in scripts that start multiple services. It guarantees that services always launch in the same sequence as the script specifies.</p>

itnotify run**Synopsis**

```
itnotify [-ORBname ORB-name] run [-background]
```

Description

Starts the notification service. The notification service must already be configured in your Application Server Platform 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

<code>-ORBname</code> <i>ORB-name</i>	<p>Directs the initializing notification service to retrieve its configuration from the specified configuration scopes.</p> <p>By default, this is the <code>notify</code> scope. Use the <code>-ORBname</code> argument to specify a different configuration scope. For example:</p> <pre>itnotify -ORBname notify.notify2 run</pre>
<code>-background</code>	<p>Runs the notification service in the background. Control returns to the command line only after the service successfully launches. If you omit the <code>-background</code> argument, the notification service runs in the foreground. You can abbreviate this argument to <code>-bg</code>. For example:</p> <pre>itnotify run -bg</pre> <p>The <code>-background</code> argument is especially useful in scripts that start multiple services. It guarantees that services always launch in the same sequence as the script specifies.</p>

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 Application Server Platform services:

Service	Shut-down command
Configuration repository	<code>itadmin config stop</code>
Locator	<code>itadmin locator stop</code>
Node daemon	<code>itadmin node_daemon stop</code>
Naming service	<code>itadmin ns stop</code>
Interface repository	<code>itadmin ifr stop</code>
Event service	<code>itadmin event stop</code>

Table 9: *itadmin* Commands for Stopping Application Server Platform Services

Managing Application Server Platform Services With itadmin

This chapter provides an overview of using the command-line tool `itadmin` to manage Application Server Platform services. Typical management tasks in the Application Server Platform include creating, viewing, and removing data stored in service repositories.

In this chapter

This chapter contains the following sections:

Using itadmin	page 214
Command Syntax	page 217
Services and Commands	page 220

Using itadmin

Overview

`itadmin` lets you manage information used by Application Server Platform 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 220](#).

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 Application Server Platform 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 301](#).

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 Application Server Platform 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	page 217
Specifying negative values	page 218
Abbreviating command parameters	page 218
Obtaining help	page 219

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 Application Server Platform services:

Configuration Domain	page 221
Location Domain	page 233
Naming Service	page 263
Interface Repository	page 275
Event Service	page 281
Notification Service	page 293
Persistent State Service	page 289
Object Transaction Service	page 301
Object Transaction Service Encina	page 305
Security Service	page 313
Trading Service	page 323

Configuration Domain

Overview

A subset of `itadmin` commands let you manage a configuration domain, both file- and configuration repository-based. These commands manage the following components of a configuration domain:

Configuration Repository	page 222
Namespaces	page 224
Scopes	page 227
Variables	page 229

Note: To use `itadmin` in a repository-based configuration domain, the configuration repository must be running (see [“Starting Application Server Platform Services”](#) on page 201).

Configuration Repository

Overview

The following commands let you manage the configuration repository:

<code>config dump</code>	Displays the entire contents of the configuration domain.
<code>config list_servers</code>	Shows all deployed replicas of the configuration repository.
<code>config stop</code>	Stops the configuration repository.

config dump

Synopsis

```
config dump
```

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...723a312e3000000000100000
  00001a00"

initial_references:POACurrent:plugin = "poa"

initial_references:NameService:reference =
  "IOR:010000002f00000049444c3a696f6e61...2e6362f49545f4e616d69
  6e60600000010000003500"

initial_references:DynAnyFactory:plugin = "it_dynany"

initial_references:ConfigRepository:reference =
  "IOR:010000002000000049444c3a495000002000...00006000000010000
  000900"
...
```

config list_servers

Synopsis

```
config list_servers [-active]
```

Description

Shows all active deployed replicas of the configuration repository.

Arguments

`-active` Displays the total number of active deployed replicas.

config show_server

Synopsis

```
config show_server cfr replica name
```

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.

Namespaces

Overview

The following commands let you manage configuration namespaces:

<code>namespace create</code>	Creates namespaces in the specified scope.
<code>namespace list</code>	Lists the namespaces in the given namespace or configuration scope.
<code>namespace remove</code>	Removes a namespace and all its contained namespaces and variables from the configuration domain.
<code>namespace show</code>	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

<code>-scope</code>	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

<code>-scope</code>	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:

<code>scope create</code>	Creates a configuration scope.
<code>scope list</code>	Displays all sub-scopes defined within a scope.
<code>scope remove</code>	Removes a configuration scope and all its contained namespaces, variables, and scopes.
<code>scope show</code>	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]
```

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.

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:

<code>variable create</code>	Creates a variable or namespace within the configuration domain.
<code>variable modify</code>	Changes one or more variable values.
<code>variable remove</code>	Removes a variable from the configuration domain.
<code>variable show</code>	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 96](#).

`-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 217](#) 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:004332434235234235933..."
initial_references:InterfaceRepository: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:

<code>-scope <i>scoped-name</i></code>	The configuration scope in which to modify the variable or namespace. The default is the root configuration scope.
<code>-type <i>type</i></code>	The type of the variable. Supply one of the following types: <ul style="list-style-type: none"> • long • bool • list (a comma-separated list of strings) • string
<code>-value <i>value</i></code>	The variable's value. The value must match the type specified by the <code>-type</code> switch. <p>The following values are valid for the specified type:</p> <p>long: any signed long value</p> <p>bool: true OR false</p> <p>list: list items must be separated by commas. Empty elements or list items containing spaces must be quoted—for example:</p> <pre>foo,"bar none",baz</pre> <p>See "Specifying lists" on page 217 for more details.</p> <p>string: Enclose values in double quotes.</p>

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

<code>-scope <i>scoped-name</i></code>	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

<code>-scope</code>	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	page 234
Named Key	page 237
Node Daemon	page 240
ORB Name	page 244
POA	page 247
Server Process	page 253

Locator Daemon

Overview

The following commands manage locator daemons:

<code>locator heartbeat_daemons</code>	Pings all the of the node daemons known to the specified locator, removing those that are no longer active.
<code>locator list</code>	Displays all locators in the location domain.
<code>locator show</code>	Displays all attributes of the specified locator daemon.
<code>locator stop</code>	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

<code>-alldomain</code>	Stops the locator, all registered node daemons, and monitored processes running in a location domain.
<code>-ior</code>	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 be recreated.

To pass the IOR of a server to a client using a named key, the user will need to supply an address in the following format:

```
corbaloc:iiop:ver@host:port/named_key
```

<code>ver</code>	The IIOP version the server uses to communicate
<code>host</code>	The hostname for the machine running the locator daemon
<code>port</code>	The port used by the locator
<code>named_key</code>	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:

<code>named_key create</code>	Creates an association between a specified well-known object key and a specified object reference.
<code>named_key list</code>	Lists all well known object keys that are registered with the locator daemon.

<code>named_key remove</code>	Removes the specified <i>object-key</i> from the location domain.
<code>named_key show</code>	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 the Application Server Platform 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 Application Server Platform 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:

<code>node_daemon list</code>	Displays all node daemon names implicitly registered with the locator daemon.
<code>node_daemon remove</code>	Removes a node daemon from the location domain that is created implicitly when the specified node daemon starts.
<code>node_daemon show</code>	Displays all attributes of the specified node daemon.
<code>node_daemon stop</code>	Stops the node daemon.
<code>add_node_daemon.tcl</code>	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.iior`.

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:

<code>orbname create</code>	Creates an ORB name in the location domain.
<code>orbname list</code>	Displays all ORB names in the location domain.
<code>orbname modify</code>	Modifies the specified ORB name entry either by associating it with another process entry, or by disassociating it from any process.
<code>orbname remove</code>	Removes an ORB name from the location domain.
<code>orbname show</code>	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 253](#)).

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:

<code>poa create</code>	Creates a POA name in the location domain.
<code>poa list</code>	Displays POA names in the location domain.
<code>poa modify</code>	Modifies the indicated POA name as specified.
<code>poa remove</code>	Removes a POA name from the location domain.
<code>poa show</code>	Displays all data that is entered for <i>POA-name</i> .

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`

<code>-replicas</code> <i>replica-list</i>	<p>Associates the specified POA with multiple ORBs specified in <i>replica-list</i>, where <i>replica-list</i> is a comma-delimited list of ORBs:</p> <p><i>orb[,orb]...</i></p> <p><code>-replicas</code> cannot be combined with <code>-persistent</code>, <code>-orbname</code>, or <code>-transient</code>.</p>
<code>-persistent</code>	<p>Marks the POA as persistent without associating it with an ORB.</p> <p><code>-persistent</code> cannot be combined with <code>-replicas</code>, <code>-orbname</code>, or <code>-transient</code>.</p>
<code>-transient</code>	<p>Marks the POA as transient.</p> <p><code>-transient</code> cannot be combined with <code>-replicas</code>, <code>-orbname</code>, or <code>-persistent</code>.</p>
<code>-alldynamic</code>	<p>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).</p>
<code>-alldynreplicas</code>	<p>Must be set to <i>yes</i> or <i>no</i>:</p> <ul style="list-style-type: none"> • <i>yes</i>: (default) Any ORB creating the POA is automatically added to the POA's replica list. • <i>no</i>: Only those ORBs that are configured in the cluster through replicas are allowed to create the POA.
<code>-load_balancer</code> <i>lb-name</i>	<p>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.</p> <p>The Orbix distribution provides support for the following algorithms:</p> <ul style="list-style-type: none"> • <i>round_robin</i>: 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. • <i>random</i>: 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 -orbnname 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

<code>-active</code>	Lists only entries for POAs that are currently active. <code>-active</code> and <code>-transient</code> parameters are mutually exclusive.
<code>-children <i>FQPN</i></code>	Lists only entries for child POAs of the specified parent POA.
<code>-count</code>	Lists the total number of POA names in the location domain.
<code>-persistent</code>	Lists only POA names for persistent POAs.
<code>-transient</code>	Lists only POA names for transient POAs. <code>-transient</code> and <code>-active</code> 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

<code>-allowdynamic</code>	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).
<code>-allowdynreplicas</code>	Must be set to <i>yes</i> or <i>no</i> : <ul style="list-style-type: none"> <i>yes</i>: (default) Any ORB creating the POA is automatically added to the POA's replica list. <i>no</i>: Only those ORBs that are explicitly configured in the cluster through replicas are allowed to create the POA.
<code>-orbname <i>ORB-name</i></code>	Associates the specified ORB name with the specified POA. This argument requires an <i>ORB-name</i> argument with the following syntax: <pre>ORBNameSegment . ORBNameSegment . ORBNameSegment</pre>

<code>-replicas</code> <code> <i>replica-list</i></code>	Associates the specified POA with multiple ORBs specified in <i>replica-list</i> , where <i>replica-list</i> is a comma-delimited list of ORBs: <code>orb[,orb]...</code> <code>-replicas</code> cannot be combined with <code>-orbname</code> .
<code>-clear_replicas</code>	Disassociates the POA from any ORBs.
<code>-load_balancer</code>	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: <ul style="list-style-type: none"> • <code>round_robin</code>: 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. • <code>random</code>: 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

<code>-active</code>	Removes currently active entries for the specified POA only. <code>-active</code> and <code>-allactive</code> arguments are mutually exclusive.
<code>-allactive</code>	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:

<code>process create</code>	Creates a server process name in the location domain.
<code>process disable</code>	Disables the specified server process for process activation, using the node daemon.
<code>process enable</code>	Enables a target server process for on-demand activation by the node daemon.
<code>process kill</code>	Kills the specified process that was started by its associated node daemon.
<code>process list</code>	Lists names of server processes in the location domain.
<code>process modify</code>	Modifies the process as specified.
<code>process remove</code>	Removes a server process name from the location domain.
<code>process show</code>	Displays a complete server process entry.
<code>process start</code>	Starts a registered server process.
<code>process stop</code>	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.svr1' 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 106](#).
- `-group` Group name that starts the target process. The default is nobody. For more information, see [page 108](#).

<code>-user</code>	User name that starts the target process. The default is nobody. For more information, see page 108 .
<code>-umask</code>	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 <code>rxwxr-xr-x</code>).

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.

Arguments

<code>-signal</code>	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

<code>-count</code>	Displays the total number of process names in the location domain.
<code>-node_daemon</code>	Lists all monitored processes for a given node daemon. This is useful if you want to perform the node_daemon stop command.
<code>-active</code>	

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

<code>-args</code>	<p>Arguments supplied to the process when it starts. At a minimum, supply the <code>-ORBname</code> argument with the name of the ORB associated with this server process.</p> <p>Enclose all arguments with quotation marks, and separate multiple arguments with spaces. For example:</p> <pre>itadmin process create -args "-ORBname company.production.sver1" my_app</pre> <p>If you are registering a Java server, the argument list generally includes the class path.</p>
<code>-description</code>	A brief description of the target process.

<code>-startupmode</code>	<p>Specifies when to start the target process using one of these arguments:</p> <ul style="list-style-type: none"> • <code>on_demand</code> (default) starts the process when requested by a client. • <code>disable</code> disables the process from starting.
<code>-node_daemon</code>	The name of the node daemon that will start or modify this process.
<code>-pathname</code>	<p>The complete pathname of the executable that will be started when the process is activated.</p> <p>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.</p>
<code>-directory</code>	<p>Specifies the working directory where the target process writes output files, error logs, and so on.</p> <p>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.</p> <p>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.</p> <p>On UNIX, if the current working directory path does not exist, it is created automatically with permissions <code>drwx-----</code>.</p> <p>Use this argument in order to:</p> <ul style="list-style-type: none"> • 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.

<code>-env</code>	<p>Explicitly sets the process environment. This argument takes a list of space-delimited <i>variable=value</i> pairs, enclosed in quotation marks:</p> <pre>env "DISPLAY=circus:0.0 CLOWN=Bozo HOME=/tent"</pre> <p>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.</p> <p>For more information about environment settings, see “Server Environment Settings” on page 106.</p>
<code>-group</code>	<p>Group name that starts the target process. The default is nobody. For more information, see page 108.</p>
<code>-user</code>	<p>User name that starts the target process. The default is nobody. For more information, see “File access permissions” on page 108.</p>
<code>-umask</code>	<p>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 <code>rwxx-rx-x</code>).</p>

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:

<code>-active</code>	Removes only the active process entry from the locator's active process table.
<code>-deep</code>	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 process' configured node daemon resides.

process stop

Synopsis

```
process stop [-signal number] process-name
```

Description

Stops the specified process that was started by its associated node daemon. The `-signal` argument specifies the UNIX signal number to stop the process.

`-signal` Specifies the UNIX signal number to stop a process. The default is 9.

WARNING: The signal number is ignored for a Windows NT process, it is terminated abruptly using the `TerminateProcess()` API.

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	page 264
Object Groups	page 268

Names

Overview

The following `ns` commands let you manage and browse the naming service:

<code>ns bind</code>	Creates an association between a context or object reference and the specified compound name.
<code>ns list</code>	Lists the contents of the specified path.
<code>ns list_servers</code>	Lists all active naming servers.
<code>ns newnc</code>	Creates a new naming context or object and binds it to the specified path.
<code>ns remove</code>	Removes the specified context or object.
<code>ns resolve</code>	Displays a resolved string name form of the IOR for a specified path.
<code>ns show_server</code>	Displays the naming server details for the server name specified.
<code>ns stop</code>	Stops the naming service.
<code>ns unbind</code>	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:

<code>nsog add_member</code>	Adds the specified member object to the specified object group.
<code>nsog bind</code>	Binds the specified object group to the specified path.
<code>nsog create</code>	Creates the specified object group, with the specified selection policy.
<code>nsog list</code>	Lists all object groups currently existing in the naming service.
<code>nsog list_members</code>	Lists the names of members belonging to the specified object group.
<code>nsog modify</code>	Modifies the selection policy for the specified object group.
<code>nsog remove</code>	Removes the specified object group from the naming service.
<code>nsog remove_member</code>	Removes the specified member object from the specified object group.
<code>nsog set_member_timeout</code>	Sets the load timeout period for a member of an active object group.
<code>nsog show_member</code>	Displays the object reference that corresponds to the specified member of an object group.
<code>nsog update_member_load</code>	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:

<i>-og_name</i> <i>group-name</i>	Specifies the object group to which the member is added.
<i>-member_name</i> <i>member-name</i>	Specifies a unique group member name.
<i>IOR</i>	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

<i>-og_name</i> <i>group-name</i>	Specifies the name of the object group to bind.
<i>path</i>	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 160](#).

Arguments

<code>-type</code>	Specifies the object group's selection algorithm with one of the following values:
<code><i>selection-policy</i></code>	<ul style="list-style-type: none"> <code>rr</code>: round-robin <code>rand</code>: random <code>active</code>: active load balancing
<code><i>group-name</i></code>	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` Specifies the target object group.
group-name

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 159](#)).

Arguments

`-type` Specifies the object group's selection algorithm with one of the following values:
selection-policy
`rr`: round-robin
`rand`: random
`active`: active load balancing (see [“Active load balancing” on page 160](#)).

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          The target object group.
  group-name
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 271](#)).

Arguments

<code>-og_name</code> <i>group_name</i>	Specifies the target object group.
<code>-member_name</code> <i>member</i>	Specifies the target object.
<code>timeout-value</code>	Specifies the timeout value in seconds. A value of <code>-1</code> 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 272](#)). 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	page 276
Repository Management	page 277

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:

<code>ifr cd</code>	Changes the current container (in shell mode).
<code>ifr destroy_contents</code>	Destroys the contents of the interface repository.
<code>ifr ifr2idl</code>	Outputs the contents of the interface repository to the specified file.
<code>ifr list</code>	Lists the contents of the current container.
<code>ifr pwd</code>	Prints the name of the current container (in shell mode).
<code>ifr remove</code>	Removes an IDL definition from the interface repository.
<code>ifr show</code>	Prints specified IDL definitions contained in the interface repository.
<code>ifr stop</code>	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.
<i>scoped-name</i>	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 `IObject::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	page 282
Event Channel	page 284

Event Service Management

Overview

The following commands let you manage an event service instance:

<code>event show</code>	Displays the attributes of the specified event service.
<code>event stop</code>	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 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 Application Server Platform Services” on page 201](#).

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:

<code>ec create</code>	Creates an untyped event channel with the specified name.
<code>ec create_typed</code>	Creates a typed event channel with the specified name.
<code>ec list</code>	Displays all untyped event channels managed by the event service.
<code>ec remove</code>	Removes the specified untyped event channel.
<code>ec remove_typed</code>	Removes the specified typed event channel.
<code>ec show</code>	Displays all attributes of the specified untyped event channel.
<code>ec show_typed</code>	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 *Application Server Platform Configuration Reference Guide*.

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

<code>pss_db archive_old_logs</code>	Archives old log files for the IOR specified.
<code>pss_db delete_old_logs</code>	Deletes old log files for IOR specified.
<code>pss_db checkpoint</code>	Performs checkpoint operations on the database referenced in the specified file.
<code>pss_db name</code>	Returns the name of the object reference to the database.
<code>pss_db post_backup</code>	Performs post-backup operations on the database referenced in the specified file.
<code>pss_db pre_backup</code>	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 *Application Server Platform Configuration Reference Guide*.

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	page 294
Event Channel	page 298

Notification Service Management

The following commands let you manage an notification service instance.

<code>notify checkpoint</code>	Performs checkpoint operations on the notification service's Berkeley DB database.
<code>notify post_backup</code>	Performs post-backup operations on the notification service database.
<code>notify pre_backup</code>	Performs pre-backup operations on the notification service database.
<code>notify show</code>	Displays the attributes of the specified notification service.
<code>notify stop</code>	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 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 Application Server Platform Services” on page 201](#).

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:

<code>nc create</code>	Creates an untyped event channel with the specified name.
<code>nc list</code>	Displays all untyped event channels managed by the notification service.
<code>nc remove</code>	Removes the specified untyped event channel.
<code>nc show</code>	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 *Application Server Platform Configuration Reference Guide*.

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:

<code>tx begin</code>	Starts a transaction.
<code>tx commit</code>	Commits a transaction.
<code>tx resume</code>	Resumes a transaction.
<code>tx rollback</code>	Rolls back a transaction.
<code>tx suspend</code>	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:

<code>encinalog add</code>	Adds a file/raw disk to the list of files/raw disks backing up a physical volume of an Encina transaction log.
<code>encinalog add_mirror</code>	Creates a new physical volume and adds this to the list of volumes mirroring an Encina transaction log.
<code>encinalog create</code>	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.
<code>encinalog display</code>	Displays information about the physical volumes of an Encina transaction log.
<code>encinalog expand</code>	Expands an Encina transaction log.
<code>encinalog init</code>	Initializes an Encina transaction log, thereby creating restart and backup restart files.

<code>encinalog remove_mirror</code>	Removes a physical volume from an Encina transaction log.
<code>otstm stop</code>	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_physicalVoln`, where *n* is the lowest number for which there is no physical volume mirroring the transaction log.

Arguments

<code>-restart</code> <i>restart-file</i>	Identifies the target transaction log.
<code>-backup</code> <i>backup-file</i>	Optionally identifies the target transaction log. If no backup restart file is specified, the default path is derived from <i>restart-file.bak</i> .
<code>-silent</code>	Suppresses the display of the completion status.
<i>file-spec</i>	The path name of a file or raw disk created with <code>encinalog create</code> .

Examples

The following example adds a physical volume backed up by file `otsmirror.log` 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

<code>-size-type</code> <i>file-size</i>	Specifies a non-default size, where <code>-size-type</code> is one of the following literals: <ul style="list-style-type: none"> • <code>-msize</code> specifies the size in megabytes. • <code>-ksize</code> specifies the size in kilobytes. • <code>-size</code> specifies the size in bytes. <p>The minimum size is 1 megabyte; the maximum size is 16 megabytes.</p>
<code>-replace</code>	Overwrites an existing file.
<code>-silent</code>	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

<code>-restart</code> <i>restart-file</i>	Identifies the target transaction log.
<code>-backup</code> <i>backup-file</i>	Optionally identifies the target transaction log. If no backup restart file is specified, the default path is derived from <code>restart-file.bak</code> .

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

<code>-restart</code> <i>restart-file</i>	Identifies the transaction log to expand
<code>-backup</code> <i>backup-file</i>	Optionally identifies the transaction log to expand. If no backup restart file is specified, the default path is derived from <i>restart-file.bak</i> .
<code>-silent</code>	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

<code>-restart</code> <i>restart-file</i>	Specifies the restart file name.
<code>-backup</code> <i>backup-file</i>	Optionally identifies the transaction log to initialize. If no backup restart file is specified, the default path is derived from <code>restart-file.bak</code> .
<code>-replace</code>	Overwrites the existing restart files.
<code>-silent</code>	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

<code>-restart</code> <i>restart-file</i>	Identifies the target transaction log.
--	--

`-backup` Optionally identifies the target transaction log. If no backup restart file is specified, the default path is derived from `restart-file.bak`.

backup-file

`-silent` Suppresses the display of the completion status.

Examples

The following example removes the physical volume `logVol_physicalVoll` 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_physicalVoll
```

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 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	page 315
Managing Checksum Entries	page 316
Managing Pass Phrases	page 319

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_admin` or `checksum`).

Arguments

- | | |
|------------------------|--|
| <code>login</code> | This argument specifies the name of an X.509 certificate that identifies the administrator.

The <i>identity</i> parameter specifies the name of a PKCS#12 certificate file, <i>identity.p12</i> , located in the directory specified by the <code>itadmin_x509_cert_root</code> configuration variable. |
| <code>-password</code> | This argument lets you specify the pass phrase for the <i>identity.p12</i> 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:

<code>checksum confirm</code>	Confirms that the process IMR entry for the specified process has not been changed since the checksum was created.
<code>checksum create</code>	Creates a checksum for the specified process IMR entry and store the checksum in the KDM database.
<code>checksum list</code>	Lists process names that have security checksum information in the KDM database.
<code>checksum remove</code>	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:

<code>kdm_adm change_pw</code>	Changes the pass phrase for encrypting the KDM database.
<code>kdm_adm confirm</code>	Confirms that the pass phrase associated with the specified ORB name has the value you expect.
<code>kdm_adm create</code>	Creates an entry in the KDM database that associates a pass phrase with the specified ORB name.
<code>kdm_adm list</code>	Lists the ORB names that have pass phrase information in the KDM database.
<code>kdm_adm new_pw</code>	Creates a new pass phrase for encrypting the KDM database.
<code>kdm_adm remove</code>	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_admin create -orbname my_orb_name
Please enter password for orb my_orb_name :
%
```

kdm_admin list**Synopsis**

```
kdm_admin 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	page 324
Federation Links	page 329
Regular Offers	page 333
Proxy Offers	page 335
Type Repository	page 337

Trading Service Administrative Settings

Overview

The following commands let you manage trading service administrative settings:

<code>trd_admin get</code>	Displays administrative settings.
<code>trd_admin set</code>	Modifies administrative settings.
<code>trd_admin stop</code>	Stops the trading service.

`trd_admin get`

Synopsis

```
trd_admin get arg
```

Description

Displays administrative settings.

Arguments

Supply one of the following arguments:

<code>-request_id_stem</code>	Displays the request id stem assigned to this instance of the trading service.
<code>-def_search_card</code>	Displays the default search cardinality-the default upper bound of offers to be searched.
<code>-max_search_card</code>	Displays the maximum search cardinality-maximum upper bound of offers to be searched.
<code>-def_match_card</code>	Displays the default match cardinality-default upper bound of matched offers to be ordered.
<code>-max_match_card</code>	Displays the maximum match cardinality-maximum upper bound of matched offers to be ordered.
<code>-def_return_card</code>	Displays the default return cardinality-default upper bound of ordered offers to be returned.
<code>-max_return_card</code>	Displays the maximum return cardinality-maximum upper bound of ordered offers to be returned.

<code>-max_list</code>	Displays the upper bound on the size of any list returned by the trading service, namely the returned offers parameter in query, and the <code>next_n</code> operations in <code>OfferIterator</code> and <code>OfferIdIterator</code> .
<code>-modifiable_properties</code>	Displays whether the trading service supports properties modification.
<code>-dynamic_properties</code>	Displays whether the trading service supports dynamic properties.
<code>-proxy_offers</code>	Displays whether the trading service supports proxy offers.
<code>-def_hop_count</code>	Displays the default hop count-default upper bound of depth of links to be traversed in a federated query.
<code>-max_hop_count</code>	Displays the maximum hop count-maximum upper bound of depth of links to be traversed in a federated query.
<code>-def_follow_policy</code>	Displays the default federation link follow policy.
<code>-max_follow_policy</code>	Displays the limiting link follow policy for all links of the trader. This setting overrides both link and importer policies.
<code>-max_link_follow_policy</code>	Displays the most permissive follow policy allowed when creating new links.
<code>-type_repos</code>	Displays the stringified IOR of the service 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:

<code>-request_id_stem <i>id_stem</i></code>	Modifies the request id stem of this instance of the trading service.
<code>-def_search_card <i>value</i></code>	Modifies the default search cardinality—the default upper bound of offers to be searched. The value must be a positive integer.
<code>-max_search_card <i>value</i></code>	Modifies the maximum search cardinality—the maximum upper bound of offers to be searched. The value must be a positive integer.
<code>-def_match_card <i>value</i></code>	Modifies the default match cardinality—the default upper bound of matched offers to be ordered. The value must be a positive integer.
<code>-max_match_card <i>value</i></code>	Modifies the maximum match cardinality—the maximum upper bound of matched offers to be ordered. The value must be a positive integer.
<code>-def_return_card <i>value</i></code>	Modifies the default return cardinality—the default upper bound of ordered offers to be returned. The value must be a positive integer.
<code>-max_return_card <i>value</i></code>	Modifies the maximum return cardinality—the maximum upper bound of ordered offers to be returned. The value must be a positive integer.
<code>-max_list <i>value</i></code>	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 <code>OfferIterator</code> and <code>OfferIdIterator</code> . The value must be a positive integer.

<code>-modifiable_properties</code> <i>boolean-value</i>	Specifies whether to enable support of modifiable properties.
<code>-dynamic_properties</code> <i>boolean-value</i>	Specifies whether to enable support of dynamic properties.
<code>-proxy_offers</code> <i>boolean-value</i>	Specifies whether to enable support of proxy offers.
<code>-def_hop_count</code> <i>value</i>	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.
<code>-max_hop_count</code>	Sets the maximum hop count-the maximum upper bound of depth of links to be traversed in a federated query.
<code>-def_follow_policy</code> <i>policy</i>	Sets the default federation link follow policy with one of the following values: <ul style="list-style-type: none"> • <code>local_only</code> • <code>if_no_local</code> • <code>always</code>
<code>-max_follow_policy</code> <i>policy</i>	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: <ul style="list-style-type: none"> • <code>local_only</code> • <code>if_no_local</code> • <code>always</code>
<code>-max_link_follow_policy</code> <i>policy</i>	Specifies the most permissive follow policy allowed when creating new links with one of the following values: <ul style="list-style-type: none"> • <code>local_only</code> • <code>if_no_local</code> • <code>always</code>
<code>-type_repos</code> <i>IOR</i>	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 manage federation links:

<code>trd_link create</code>	Creates a federation link.
<code>trd_link list</code>	Lists all federation links.
<code>trd_link modify</code>	Modifies a federation link.
<code>trd_link remove</code>	Removes a federation link.
<code>trd_link show</code>	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

<code>-target <i>IOR</i></code>	Defines the trading service instance the link points to. An IOR to a <code>CosTrading::Lookup</code> interface is expected.
<code>-def_pass_on_follow_rule <i>rule</i></code>	Defines default link-follow behavior to pass on for a particular link, if an importer does not specify its <code>link_follow_rule</code> ; it must not exceed <code>limiting_follow_rule</code> . Supply one of the following values for <code>rule</code> : <ul style="list-style-type: none"> • <code>local_only</code> • <code>if_no_local</code> • <code>always</code>

`-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

<code>-def_pass_on_follow_rule</code> <i>rule</i>	Defines the default link-follow behavior to be passed on for a particular link if an importer does not specify its <code>link_follow_rule</code> ; it must not exceed <code>limiting_follow_rule</code> . Supply one of the following values for <i>rule</i> : <ul style="list-style-type: none"> • <code>local_only</code> • <code>if_no_local</code> • <code>always</code>
<code>-limiting_follow_rule</code> <i>rule</i>	Defines limiting link follow behavior for a particular link. Supply one of the following values for <i>rule</i> : <ul style="list-style-type: none"> • <code>local_only</code> • <code>if_no_local</code> • <code>always</code>
<i>link-name</i>	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 manage regular offers:

<code>trd_offer list</code>	Lists all regular offers.
<code>trd_offer remove</code>	Removes a regular offer.
<code>trd_offer show</code>	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:00000000000000224...
service type:
    Printer
properties:
    boolean color TRUE
    long dpi 3200
    short ppm 30
```

Proxy Offers

Overview

The following commands let you manage proxy offers:

<code>trd_proxy list</code>	Lists all proxy offers.
<code>trd_proxy remove</code>	Removes a proxy offer.
<code>trd_proxy show</code>	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

The following commands effect the server type repository:

<code>trd_type list</code>	Lists all service types in the service type repository.
<code>trd_type mask</code>	Masks a service type.
<code>trd_type remove</code>	Removes a service type from the service type repository.
<code>trd_type show</code>	Displays details on a given service type.
<code>trd_type unmask</code>	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:

<code>bridge create</code>	Creates a bridge.
<code>bridge destroy</code>	Destroys a bridge.
<code>bridge list</code>	Lists all of the instantiated bridges in a deployment.
<code>bridge show</code>	Displays the status of a bridge.
<code>bridge start</code>	Starts the flow of messages through a bridge.
<code>bridge stop</code>	Stops the flow of messages through a bridge.
<code>bridge suspend</code>	Suspends the flow of messages through a bridge.
<code>endpoint_admin show</code>	Displays a bridge's endpoint admin's name and the type of endpoints it supports.
<code>endpoint destroy</code>	Destroys an endpoint.
<code>endpoint list</code>	Lists the endpoints associated with an endpoint admin.
<code>endpoint show</code>	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

<code>-source_admin</code>	The IOR or initial reference of the administrative object used to connect to the message source. To use the default notification endpoint admin use <code>"IT_NotificationEndpointAdmin"</code> ; to use the default JMS endpoint admin use <code>"IT_JMSEndpointAdmin"</code> .
<code>-source_type</code>	The type of object that passes messages into the bridge. It can take one of three values: <code>topic</code> if the messages originate from a JMS topic, <code>queue</code> if the messages originate from a JMS queue and <code>channel</code> if the messages originate from a notification channel.
<code>-source_name</code>	The name of the object that passes messages to the bridge.
<code>-sink_admin</code>	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 <code>Destination</code> . To use the default notification admin use <code>"IT_NotificationEndpointAdmin"</code> ; to use the default JMS admin use <code>"IT_JMSEndpointAdmin"</code> .
<code>-sink_type</code>	The type of object that receives messages from the bridge. It can take one of three values: <code>topic</code> if the messages are being forwarded to a JMS topic, <code>queue</code> if the messages are being forwarded to a JMS queue and <code>channel</code> if the messages are being forward to a notification channel.
<code>-sink_name</code>	The name of the object that receives messages from the bridge.
<code>bridge name</code>	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:

<code>jms start</code>	Starts the JMS broker.
<code>jms stop</code>	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:

Application Server Platform Windows Services	page 349
Run Control Scripts for Unix Platforms	page 363
ORB Initialization Settings	page 387
Internationalization Configuration Variables	page 393
Development Environment Variables	page 399
Debugging IOR Data	page 401

Application Server Platform Windows Services

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

This appendix describes how you can manage Application Server Platform 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 Application Server Platform Services on Windows	page 351
Application Server Platform Windows Service Commands	page 352
Application Server Platform Windows Service Accounts	page 355
Running Application Server Platform Windows Services	page 357
Logging Application Server Platform Windows Services	page 359
Uninstalling Application Server Platform Windows Services	page 360
Troubleshooting Application Server Platform/Windows Services	page 361

Managing Application Server Platform Services on Windows

Overview

If you choose to install Application Server Platform 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 Application Server Platform commands (see "[Application Server Platform Windows Service Commands](#)").

Note: In order to install and uninstall Application Server Platform services as Windows services, you must execute the [install](#) and [uninstall](#) commands.

Identifying Application Server Platform services as Windows services

Each installed Application Server Platform 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.

Application Server Platform Windows Service Commands

Overview

You can manage Application Server Platform 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 387](#). 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 Application Server Platform 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 Application Server Platform service as a Windows service. Because the Application Server Platform configure 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 Application Server Platform 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 Application Server Platform Windows services by running the Application Server Platform configure 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 Application Server Platform 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 Application Server Platform 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 Application Server Platform 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 Application Server Platform service as a Windows service. See [“Uninstalling Application Server Platform Windows Services” on page 360](#) for more details.

Application Server Platform Windows Service Accounts

Overview

By default, Application Server Platform 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, select **Control Panel | Services | Startup**. The Service dialog displays. Use this dialog also to enable interaction with the desktop for a `LocalSystem` account only.



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, select **Control Panel | Services | Startup** to change the `user/group` and `passwd` for the service.

Application Server Platform node daemons run under the `LocalSystem` account and activate other processes as the `LocalSystem` account. If this is not desired, select **Control Panel | Services | Startup** to change the `user/group` and `passwd` for this service.

Running Application Server Platform Windows Services

Overview

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

When Application Server Platform 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 Application Server Platform 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 Application Server Platform services as Windows services in a file-based domain, run Application Server Platform 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 Application Server Platform 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 185](#) for more information on Application Server Platform logging.

Uninstalling Application Server Platform Windows Services

Overview

In order to cleanly remove any version of Application Server Platform from your system, you should first uninstall all Application Server Platform 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 Application Server Platform 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 Application Server Platform/Windows Services

The following sections describe several common problems related to Application Server Platform/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 E2A Application Server Platform 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 E2A Application Server Platform 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 E2A Application Server Platform services for automated startup and shutdown or to use run control scripts generated by Orbix E2A Application Server Platform as a starting point for customization.

Note: For reliable startup and shutdown of Orbix E2A Application Server Platform services, it is recommended that you install the Java runtime, the Orbix E2A Application Server Platform components, the license file, the domain configuration files, the service databases and the log files on locally mounted filesystems.

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

Operating Systems

Follow the links below for details on your operating system:

Solaris	page 365
AIX	page 369
HP-UX	page 373
IRIX	page 379
Red Hat Linux	page 383

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 E2A services for domain
    ${DOMAIN}"
    ${DOMAIN_START_SCRIPT}
else
echo "ERROR: Failed to start IONA Orbix E2A 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 E2A services for domain
    ${DOMAIN}"
    ${DOMAIN_STOP_SCRIPT}
else
echo "ERROR: Failed to stop IONA Orbix E2A services for domain
    ${DOMAIN} - \
    domain stop script ${DOMAIN_STOP_SCRIPT} does not exist
    or is not executable"
rval=1
fi
;;
*)
```

```

echo "IONA Orbix E2A 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 E2A Application Server Platform 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 E2A 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 E2A services for domain ${DOMAIN}"
${DOMAIN_START_SCRIPT}
else
echo " ERROR: Failed to start IONA Orbix E2A 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 E2A services for domain <domain>"
${DOMAIN_STOP_SCRIPT}
else
echo "Can not stop IONA Orbix E2A servies for domain <domain> - \
      domain stop script ${DOMAIN_STOP_SCRIPT} does not exist
      or is not executable"
rval=1
fi
;;
*)
echo "IONA Orbix E2A 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 E2A <domain> >
if [ -x /etc/rc.itsvs_<domain> ]; then
/etc/rc.itsvs_<domain> stop
else
echo "ERROR: Failed to stop IONA Orbix E2A services for domain
<domain> - \
      /etc/rc.itsvs_<domain> does not exist or is not
      executable"
fi
#</IONA Orbix E2A <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 E2A <domain> >` and `</IONA Orbix E2A <domain> >` tags in `/etc/rc.shutdown`.

Enable automatic services

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

1. Uncomment the code between `<IONA Orbix E2A <domain> >` and `</IONA Orbix E2A <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 E2A domain registered for automatic startup and shutdown, remove file */etc/rc.shutdown*. Otherwise, remove the code between *<IONA Orbix E2A <domain> >* and *</IONA Orbix E2A <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 E2A services for domain ${DOMAIN}"
;;

'stop_msg')
echo "Stopping IONA Orbix E2A services for domain ${DOMAIN}"
;;
```

```
'start')
if [ "ITSVS_${DOMAIN}" -eq 1 ]; then
  if [ -x ${DOMAIN_START_SCRIPT} ]; then
    echo "Starting IONA Orbix E2A services for domain ${DOMAIN}"
    ${DOMAIN_START_SCRIPT}
    rval=4
  else
    echo "ERROR: Failed to start IONA Orbix E2A 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 E2A services for the ${DOMAIN} domain"
    ${DOMAIN_STOP_SCRIPT}
    rval=4
  else
    echo "ERROR: Failed to start IONA Orbix E2A 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 E2A 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 E2A services, domain <domain> configuration
#      ITSVS_<DOMAIN>: set to 1 to enable Orbix E2A 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 E2A services for domain ${DOMAIN}"
    ${DOMAIN_START_SCRIPT}
else
    echo "ERROR: Failed to start IONA Orbix E2A 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 E2A services for domain ${DOMAIN}"
    ${DOMAIN_STOP_SCRIPT}
else
    echo "ERROR: Failed to stop IONA Orbix E2A servies for domain
    ${DOMAIN} - "
    echo "domain stop script ${DOMAIN_STOP_SCRIPT} does not exist
    or is not executable"
    rval=1
fi
;;

```

```

*)
echo "IONA Orbix E2A 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 E2A Application Server Platform determines the default run level.

Run control scripts

Run control scripts generated by Orbix E2A Configure are compatible with `chkconfig(8)` and `linuxconf`.

For a domain named `<domain>`, the following files are generated by Orbix E2A Configure:

```
/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 E2A 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 E2A services for domain <domain>"
    ${DOMAIN_START_SCRIPT}
    touch ${DOMAIN_LOCK_FILE}
else
    echo "ERROR: Failed to start IONA Orbix E2A 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 E2A services for domain <domain>"
    ${DOMAIN_STOP_SCRIPT}
else
    echo "ERROR: Failed to stop IONA Orbix E2A 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 E2A 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 *Application Server Platform Configuration Reference Guide*.

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 its 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.

Internationalization Configuration Variables

Internationalization is the process of enabling software so that one binary can address issues arising in different languages and countries. With internationalization, it becomes possible, for example, to avoid having a French branch and Japanese branch of application source code. An internationalized product such as Application Server Platform installs correctly irrespective of locale and handles user data correctly irrespective of the encoding that data uses. The Application Server Platform provides a mechanism for users to write applications that work easily in their own particular language market.

In this appendix

This appendix contains the following sections:

Description of Configuration Variables	page 394
Default Values	page 396

Description of Configuration Variables

CORBA

Table 10 on page 394 describes the variables used to control the internationalization behavior of the Application Server Platform's CORBA services and CORBA applications deployed into an Application Server Platform domain.

Note: CORBA conversion codesets are specified using OSF Codeset Registry identifiers. For a full list of identifiers see ftp://ftp.opengroup.org/pub/code_set_registry.

Table 10: CORBA Configuration Variables

Configuration Variables	Description
<code>plugins:codeset:char:ncs="OSF-codeset-id";</code>	Specifies the native narrow character codeset.
<code>plugins:codeset:char:ccs=["OSF-codeset-registry-id", "OSF-codeset-id2", . . .];</code>	Specifies the list of conversion narrow character codesets supported.
<code>plugins:codeset:wchar:ncs="OSF-codeset-id";</code>	Specifies the native wide character codeset.
<code>plugins:codeset:wchar:ccs=["OSF-codeset-registry-id1", "OSF-codeset-id2", . . .];</code>	Specifies the list of conversion wide character codesets supported.
<code>plugins:codeset:always_use_defaults=<i>boolean</i></code>	Specifies that hardcoded default values are used and the above variables are ignored if set in the same configuration scope or higher.

J2EE

Table 11 on page 395 describes the configuration variables used to control the internationalization behavior of the Application Server Platform's J2EE application server.

Table 11: *J2EE Configuration Variables*

Configuration Variables	Description
<pre>plugins:118n:locale:locale-ianac charset-map=["locale1=IANA-char set1", "locale2=IANA-charset2", . . .];</pre>	<p>Maps from a locale (for example: "ko_KR") to a codeset that makes sense for that locale (for example: "EUC-KR"). If a JSP or Servlet makes a <code>HttpServletResponse::setLocale(locale)</code> call, then the encoding associated with that locale is be used to encode any string parameters in the <code>HttpServletResponse</code>.</p>
<pre>plugins:118n:characterencoding:ian acharset-javaconverter-map=["IANA-charset1=java-converte r1", "IANA-charset2=java-converter 2", . . .];</pre>	<p>Maps from an IANA character set name to a Java converter name. Needed in a small but significant number of cases.</p>
<pre>plugins:118n:characterencoding:u rl-inputcharset-map=["url1/*= IANA-charset1", url2/*=IANA-charset2", . . .];</pre>	<p>Maps from a JSP/Servlet URL to a fallback encoding to use when handling <code>HttpServletRequest</code> parameters to the JSP/Servlet. Encodings specified by the JSP/Servlet using <code>HttpServletRequest::setCharacterEncoding()</code> Or <code>HttpServletRequest::setContentType()</code> take precedence.</p>

Default Values

C++ CORBA

The default settings for C++ CORBA vary based on the type of platform the Application Server Platform is deployed on. For non-MVS platforms consult [Table 12 on page 396](#). For MVS platforms consult [Table 13 on page 396](#).

Table 12: C++ CORBA (non-MVS platforms)

Narrow NCS	Narrow CCS	Wide NCS	Wide CCS
0x00010001 (ISO 8859-1)		0x00010100 (UCS-2, Level 1)	0x00010109 (UTF-16)
		0x00010104 (UCS-4, Level 1)	

Note: For C++ CORBA, the default narrow code set is fixed to ISO 8859-1 or EBCDIC, which only covers select Western European languages. Users must set the configuration variables explicitly in order to deploy for non ISO-8859-1 languages using narrow character interfaces.

Note: One of the two Wide NCS values is chosen depending on the OS platform.

Table 13: C++ CORBA MVS Platforms

Narrow NCS	Narrow CCS	Wide NCS	Wide CCS
0x10020025 (IBM-037 (EBCDIC))	0x00010001 (ISO 8859-1)	0x00010100 (UCS-2, Level 1)	0x00010109 (UTF-16)
		0x00010104 (UCS-4, Level 1)	

Java CORBA

Java CORBA uses the `file.encoding` property to determine default values for different locales.

Table 14: *Java CORBA (ISO-8859-1/Cp-1252/US-ASCII locale)*

Narrow NCS	Narrow CCS	Wide NCS	Wide CCS
0x00010001 (ISO 8859-1)	0x05010001 (X/Open UTF-8)	0x00010109 (UTF-16)	0x00010100 (UCS-2, Level 1)

Table 15: *Java CORBA (Shift_JIS locale)*

Narrow NCS	Narrow CCS	Wide NCS	Wide CCS
0x05010001 (X/Open UTF-8)	0x05000011 (OSF Japanese SJIS-1)	0x00010109 (UTF-16)	0x00010100 (UCS-2, Level 1)
	0x00030010 (JIS eucJP)		0x05000011 (OSF Japanese SJIS-1)
	0x00010001 (ISO 8859-1)		0x00030010 (JIS eucJP)

Table 16: *Java CORBA (EUC-JP locale)*

Narrow NCS	Narrow CCS	Wide NCS	Wide CCS
0x05010001 (X/Open UTF-8)	0x00030010 (JIS eucJP)	0x00010109 (UTF-16)	0x00010100 (UCS-2, Level 1)
	0x05000011 (OSF Japanese SJIS-1)		0x05000011 (OSF Japanese SJIS-1)
	0x00010001 (ISO 8859-1)		0x00030010 (JIS eucJP)

Table 17: *Java CORBA (other locales)*

Narrow NCS	Narrow CCS	Wide NCS	Wide CCS
0x05010001 (X/Open UTF-8)	"file encoding"	0x00010109 (UTF-16)	0x00010100 (UCS-2, Level 1)
	0x00010001 (ISO 8859-1)		"file encoding"

J2EE

`locale-ianacharset-map`: empty

`ianacharset-javaconvertor-map`: empty

`url-inputcharset-map`: empty (ISO-8859-1 is used as the default charset).

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`.

Debugging IOR Data

The Application Server Platform 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	page 402
Using iordump	page 405
iordump Output	page 407
Data, Warning, Error and Information Text	page 412

IOR Data Formats

Overview

CORBA inter-operable object reference (IOR) data can be presented in one of two forms:

- Stringified form which is coded by converting each binary byte of coded data into an ASCII pair of characters representing the hex equivalent in readable form.
- CDR encoded (and aligned) binary data, which encodes each CORBA defined data type on its natural boundary. Short values are encoded on a 2-byte boundary, long values on a 4-byte boundary and, so on. Data contains padding between data types in order to ensure aligned data.

Stringified IOR data

Stringified IOR data is in the format `IOR:` followed by a series of hex value pairs. For example:

```
IOR:010000001c00000049444c3a53696d706c652f53696d706c654f626a6
```

It is best known as the CORBA IOR: URL passed to the IDL operation `CORBA::ORB::string_to_object()`. The stringified IOR data format of an encoded IOR can be obtained by using the IDL operation `CORBA::ORB::object_to_string()`.

IDL definition

Raw IOR data is encoded as the CDR representation of the IOR structure, defined in the CORBA GIOP specification, declared by the IDL shown in [Example 3](#):

Example 3: *IOR data IDL definition*

```

// IDL
typedef unsigned long ProfileId;

const ProfileId TAG_INTERNET_IOP = 0;
const ProfileId TAG_MULTIPLE_COMPONENTS = 1;

// A TaggedProfile contains opaque profile and component
// data and a tag to indicate the type and format of the data.
struct TaggedProfile
{
    ProfileId tag;
    sequence <octet> profile_data;
};

// IOR is a sequence of object specific protocol profiles
// (TaggedProfiles) plus a type id.
struct IOR
{
    string type_id;
    sequence <TaggedProfile> profiles;
};

// A MultipleComponentProfile is contained in a TaggedProfile
// with the tag TAG_MULTIPLE_COMPONENTS.
typedef unsigned long ComponentId;

struct TaggedComponent
{
    ComponentId tag;
    sequence <octet> component_data;
};

typedef sequence <TaggedComponent> MultipleComponentProfile;

```

Example 3: *IOR data IDL definition*

```
// This declares IIOP ProfileBody data contained in a
// TaggedProfile with the tag TAG_INTERNET_IOP.
// IIOP 1.0/1.1/1.2 revisions are given.
struct Version
{
    octet major;
    octet minor;
};

struct ProfileBody_1_0
{
    Version iiop_version;
    string host;
    unsigned short port;
    sequence <octet> object_key;
};

struct ProfileBody_1_1
{
    Version iiop_version;
    string host;
    unsigned short port;
    sequence <octet> object_key;
    sequence <IOP::TaggedComponent> components; // Added in 1.1
};

typedef ProfileBody_1_1 ProfileBody_1_2; // Same as 1.1
```

Using iordump

Overview

`iordump` is a utility that 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 specified either in the standard CORBA defined stringified form or raw binary CDR encoded data. The IOR content is displayed in both stringified and ASCII-hex formats. The tools emphasis is on reporting all possible erroneous values or suspect data, while also displaying the meaning and value of each data item.

Parameters

`iordump` takes the following parameters:

<code>-no_host_check</code>	The default behavior is to attempt a host lookup on each host specified in the IOR. This option prevents this host lookup check.
<code>file</code>	Specifies the name of the file from which to read the IOR data.
<code>-</code>	Specifies that the IOR data is to be read from <code>stdin</code> .
<code>IOR:...</code>	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 format. 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 412](#).

Example

[Example 4](#) shows a sample output from `iordump`.

Example 4: Sample `iordump` output

```
C:\>iordump simple1.ior

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

>>
  IOR:010000001c00000049444c3a53696d706c652f53696d706c654f626a6
  563743a312e3000010000000000000006a000000010102000e00000036332e
  36352e3133332e32353000a70f1b0000003a3e0231310c00000000ec09000
  08d2000000800000000000000000000020000000100000001800000001000000
  0100010000000000000101000100000009010100060000000600000001000
  0001100
-----
>> +0 [01]
      Byte order of IOR: (1) Little Endian
>> +1 [00][00][00]
      (padding)
>> +4 [1c][00][00][00]
      TypeId length: 28 bytes (including null)
>> +8
  [49][44][4c][3a][53][69][6d][70][6c][65][2f][53][69][6d][70][
  6c][65][4f][62][6a][65][63][74][3a][31][2e][30][00]
      TypeId value: 'IDL:Simple/SimpleObject:1.0.'
>> +36 [01][00][00][00]
      Number of tagged profiles: 1
```

Example 4: *Sample iordump output*

```

Profile 1:
>> +40 [00][00][00][00]
      Tag: (0) TAG_INTERNET_IOP
>> +44 [6a][00][00][00]
      Profile length: 106 bytes
>> +48 [01]
      Byte Order: (1) Little Endian
>> +49 [01][02]
      Version: 1.2
>> +52 [0e][00][00][00]
      Host length: 14 bytes (including null)
>> +56 [36][33][2e][36][35][2e][31][33][33][2e][32][35][30][00]
      Host string: '63.65.133.250.'
      * host IP address lookup succeeded, but failed to
      find a hostname (warning)
>> +70 [a7][0f]
      Port: 4007
>> +72 [1b][00][00][00]
      Object Key length: 27 bytes (including any
      trailing null)
>> +76
      [3a][3e][02][31][31][0c][00][00][00][00][ec][09][00][00][8d][
      20][00][00][
      08][00][00][00][00][00][00][00][00]
      Object key data: '>.11.....'
      (looks like an Orbix ART Transient key)
>> +103 [00]
      (padding)
>> +104 [02][00][00][00]
      Number of tagged components: 2

```


Example 4: *Sample iordump output*

```

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	page 410
ASCII-Hex Data Output	page 411

Stringified Data Output

All output begins with the stringified IOR such as:

```
Stringified IOR is: ([string/coded data] length: 312 / 154 bytes)
>>
IOR:010000001c00000049444c3a53696d706c652f53696d706c654f626a6
563743a312e300001000000000000006a000000010102000e00000036332e
36352e3133332e32353000a70f1b0000003a3e0231310c00000000ec09000
08d200000080000000000000000000002000000010000001800000001000000
01000100000000000000101000100000009010100060000000600000001000
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	page 413
Warnings	page 417

Errors

* 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 IOP version (attempting to read as 1.0 data)

The `ProfileBody` is not one of the supported IOP versions recognized by `iordump`. An attempt is made to interpret the initial part of the data as 1.0 IOP 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, `ior_dump` 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 since 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, since 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

* 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 `ior_dump`), 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 IIO 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

A

- active connection management 129
 - client-side configuration 130
 - server-side configuration 129
- active load balancing 160
- admin_logon 315

B

- Berkeley DB environment 177
 - back up 181
 - checkpoints 178
 - data files 177
 - file types 177
 - recovery 182
 - store environment files 177
 - transaction log files 177
 - archive 179
 - delete 179
 - size 179

C

- checkpoints
 - Berkeley DB 178
- checksum 313
- confirm 316
- create 317
 - list 317
 - list all processes 317
 - manage 316
 - remove 318
- command-line parameters
 - ORBadm_in_config_domains_dir 99
 - ORBadm_in_domain_name 99
 - ORBconfig_domain 85
 - ORBdomain_name 99
- config dump 222
- config list 223
- config stop 223
- configuration
 - file-based 26
 - itadmin commands 221
 - namespace management 224

- repository-based 27
- scope management 227
- variable management 229
- configuration directory
 - default 85
- configuration domain
 - create 44
 - obtain for ORB 83
 - C++ applications 85
 - Java applications 85
 - replicate 74
 - troubleshoot 99
- configuration program. See Orbix E2A Configure
- configuration repository 27
 - dump contents 222
 - list replicas 223
 - manage 222
 - start 204
 - stop 223
- configuration scope 87
 - define 89
 - file-based configuration 89
 - itadmin commands 90
 - map to ORB name 88
 - name 88
 - share 93
- configuration variables
 - components 96
 - data type 96
 - constructed 96
 - namespace 96
 - precedence of settings 90
 - set value 97
- CREATE_DEFAULT_ERROR_MODE 107
- CREATE_NEW_PROCESS_GROUP 107

D

- data files
 - Berkeley DB 177
- default-domain.cfg 85
- demo programs 77
- DETACHED_PROCESS 107
- direct persistence

failover 136

E

ec

- create 284
- list 285
- remove 285
- show 286

encinalog

- add 306
- add_mirror 307
- create 307
- display 308
- expand 309
- init 310
- remove_mirror 310

Encina transactions

- add backup files 306
- add mirror volume 307
- create log backup 307
- display mirror volume data 308
- expand transaction log 309
- initialize transaction log 310
- remove mirror 310
- stop service 311

environment variables

- development 400
- ORB initialization 387

event

- show 282
- stop 283

event channel

- create 284, 298
- list all 285, 298
- manage 284, 298
- remove 285, 299
- show attributes 286, 299

event service

- itadmin commands 281
- manage 282
- show attributes 282
- start 209
- stop 283

F

failover 131, 135

- direct persistence 136

federation links,manage 329

file-based configuration 26

filename 197

filters 187

FQPN 9

H

hard_limit

- IIOP 129, 130

I

IDL 16

- compile 16

IDL definitions, manage 164, 276

ifr

- cd 277
- destroy_contents 278
- ifr2idl 278
- list 278
- pwd 278
- remove 279
- show 279
- stop 163, 279

IIOP plug-in configuration

- hard connection limit
 - client 130
 - server-side 129
- soft connection limit
 - client 130
 - server 129

implementation repository 11

Interface Definition language. See IDL

interface repository

- add IDL definitions 167, 276
- browse contents 165
- destroy contents 278
- display containment hierarchy 165
- itadmin commands 275
- list container contents 165, 278
- list current container 278
- maintain 16
- manage 275
- navigate to other containment levels 165, 277
- remove definitions 168, 279
- show scoped name 279
- start 163
- start daemon 208
- stop daemon 163, 279
- usage 16

- write contents to file 278
- interfaces
 - add to interface repository 167, 276
 - define 16
 - obtain from interface repository 16
 - remove definitions from interface repository 168
- interoperable object reference. See IOR
- iona.properties file 76
- IOR 11
- itadmin commands 214
 - abbreviations 218
 - command-line usage 214
 - configuration domain 221
 - event service 281
 - help 219
 - interface repository 275
 - lists 217
 - location domain 233
 - naming service 264
 - negative values 218
 - nested 214
 - notification service 293
 - object group 268
 - OTS 301
 - OTS Encina 305
 - PSS 289
 - shell usage 214
 - SSL/TLS 313
 - syntax 217
 - Tcl scripts 215
 - trading service 323, 341
 - undo 216
- IT_CONFIG_DIR 389
- IT_CONFIG_DOMAIN 85
- IT_CONFIG_DOMAINS_DIR 388
- itconfig_rep run 204
- itconfigure 44, 98
 - JAVA_HOME setting 37
 - syntax 38
 - UNIX access permissions 37
- IT_DOMAIN_NAME 388
- itevent run 209
- IT_IDL_CONFIG_FILE 399
- IT_IDLGEN_CONFIG_FILE 400
- itifr run 163, 208
- itlocator run 112, 205
- itnaming run 152, 207
- itnode_daemon run 114, 206
- itnotify run 210

IT_PRODUCT_DIR 390

J

- JAVA_HOME 37
- Java interpreter 76

K

- KDM 313
 - database 313
 - log on 315
- kdm_adm change_pw 319
- kdm_adm confirm 320
- kdm_adm create 320
- kdm_adm list 321
- kdm_adm remove 322

L

- load balancing
 - active selection 160
 - replicated servers 131
 - selection strategies 159, 270, 271
- location domain
 - daemon. See locator daemon
 - implementation repository 11
 - itadmin commands 233
 - list registered entries 117
 - modify entries 118
 - register ORB 104
 - register POA 105
 - register server process 103
 - remove entries 118
- locator
 - list 234
 - show 235
 - stop 112, 235
- locator daemon 11
 - list all 234
 - manage 234
 - restart 113
 - show attributes 235
 - start 112, 205
 - stop 112, 235
 - usage 12
- locator daemon configuration
 - find persistent objects 11
- logging configuration
 - message severity levels 195
 - set filters for subsystems 187

- subsystems 193
- logstream configuration 197
 - output to local file 197
 - output to rolling file 197
 - output to system log 198

N

name

- bind to object 264
- rebind 158

named_key

- create 238
- list 238
- remove 239
- show 239

named keys

- create 238
- list all 238
- manage 237
- remove 239
- show object reference 239

namespace

- create 224
- list 224
- remove 225
- show 226

namespaces

- create 224
- list 224
- manage 224
- remove from configuration 225
- show contents 226

naming context

- create 155
- unbound 155

naming graph 150

- build 153

naming service 6

- administer 149
- bind name 264
- bind name to object 156
- build naming graph 153
- itadmin commands 264
- list contents 265
- manage 264
- naming context
 - create 155
 - unbound 155
- naming graph 150

new context 265

object groups 159, 268

rebind name 158

resolve name 266

start 152, 208

stop 152, 267

unbind 266, 267

nc

- create 298

- list 298

- remove 299

- show 299

node daemon 114

- list 240

- list active processes 115

- manage 240

- remove 241

- run several on host 115

- show attributes 241

- start 114, 206

- stop 115, 241

- usage 12

node_daemon

- list 240

- remove 241

- show 241

- stop 115, 241

NORMAL_PRIORITY_CLASS 107

notification service

- checkpoint operations 294

- itadmin commands 293

- manage 294

- post-backup operations 295

- pre-backup operations 295

- show attributes 295

- start 210

- stop 296

notify

- checkpoint 294

- post_backup 295

- pre_backup 295

- show 295

- stop 296

ns

- bind 156, 264

- list 265

- newnc 155, 265

- remove 266

- resolve 158, 266

- stop 152, 267
- unbind 158, 267
- nsog
 - add_member 269
 - bind 269
 - create 270
 - list 270
 - list_members 270
 - modify 271
 - remove 271
 - remove_member 272
 - set_member_timeout 272
 - show_member 273
 - update_member_load 273
- O**
- object group 159
 - active load balancing 160
 - add member 269
 - bind 269
 - create 159, 270
 - identifier 159
 - itadmin commands 268
 - list all 270
 - list members 270
 - manage 268
 - member identifiers 159
 - member IOR 273
 - member load value updates 273
 - member timeout 272
 - modify selection strategy 271
 - remove 271
 - remove member 272
 - selection strategies 159, 270, 271
- OBJECT_NOT_EXIST exception 11
- object references 6
 - client invocations on 6
 - map to servants 8
- object request broker. See ORB
- objects
 - persistent 11
 - transient 11
- on-demand activation 103
 - replicated server 140
- ORB
 - configuration 87
 - initialization 84, 387
 - map name to configuration scope 88
 - register in location domain 104

- register root POA name 119
- server 4
 - share configuration scope 93
- ORBadmin_config_domains_dir 99
- ORBadmin_domain_name 99
- ORBconfig_dir 389
- ORBconfig_dir Java property 389
- ORBconfig_domain 85
- ORBconfig_domain Java property 85
- ORBconfig_domains_dir 388
- ORBconfig_domains_dir Java property 388
- ORBDefaultInitRef 390
- ORBdomain_name 99, 388
- ORBdomain_name Java property 388
- ORB initialization 387
 - configuration directory 389
 - default initial reference 390
 - domain name 388
 - domains directory 388
 - initial reference 389
 - Java properties 387
 - ORB name 389
 - precedence of settings 387
 - product directory 390
- ORBInitRef 389
- Orbix services
 - order of startup 202
 - start and stop scripts 75, 202
 - start commands 203
 - stop commands 212
- ORBname 389
- ORB name 389
 - create 244
 - list all 244
 - manage 244
 - modify 245
 - remove 245
 - show attributes 246
- orbname
 - create 104, 244
 - register replicated server 141
 - list 244
 - modify 245
 - remove 245
 - show 246
- ORBproduct_dir 390
- ORBproduct_dir Java property 390
- OTS
 - itadmin commands 301

INDEX

- manage 301
- OTS Encina
 - itadmin commands 305
 - manage 305
- otstm stop 311

P

- pass phrases 313
 - change 319
 - confirm 320
 - create 320
 - list 321
 - manage 319
 - remove 322
- persistent objects 11
 - direct persistence
 - and failover 136
 - invoke on 11
 - locate 103
 - replicated 133
- POA 8
 - FQPN 9
 - list 249
 - manage 247
 - modify 250
 - name root POA 119
 - names 9
 - persistent 103
 - register in location domain 105, 247
 - remove 251
 - replicas 105, 132
 - show attributes 252
 - transient 105
- poa create 105, 247
 - replicated POA 141
- poa list 249
- poa modify 250
- poa remove 251
- poa show 252
- portable object adapter. See POA
- process
 - create 103, 253
 - disable 256
 - enable 256
 - list 115, 256
 - modify 257
 - remove 259
 - show 260
 - start 111, 261

- stop 111, 261
- proxy offers, manage 335
- PSS
 - checkpoint 289
 - itadmin commands 289
 - manage 289
 - obtain IOR to 290
 - post-backup operations 290
 - pre-backup operations 291
- pss_db
 - checkpoint 289
 - name 290
 - post_backup 290
 - pre_backup 291

R

- recovery
 - Berkeley DB 182
- regular offers, manage 333
- replicated servers 131
 - add server replicas 143
 - build 139
 - deploy 132
 - failover 135
 - IONA services 74, 145
 - load balancing 135
 - change strategy 144
 - specifying strategy 141
 - on-demand activation 140
 - register ORB names 141
 - register POA 141
 - register processes 140
 - startup 133
- replicated services
 - configuration repository 74
 - location daemon 74
 - set up 74, 145
- repository-based configuration 27
- rolling_file 197
- root_name 119
- root POA
 - register name 119

S

- sample applications 77
- scope
 - create 227
 - list 227

- list sub-scopes 227
- manage 227
- remove 228
- show 228
- show contents 228
- scope See configuration scope
- secure_directories 111
- server process
 - disable on-demand activation 256
 - enable on-demand activation 256
 - list 256
 - manage 253
 - modify 257
 - register 253
 - register for on-demand activation 103
 - on replicated server 140
 - remove 259
 - secure directories 111
 - show attributes 260
 - start 261
 - start and stop 111
 - stop 261
- servers, reactivate with node daemon 12
- soft_limit
 - IIOP 129, 130
- SSL/TLS
 - itadmin commands 313
 - KDM 313
 - manage 313

T

- Tcl scripts, itadmin commands 215
- trading service
 - create federation link 329
 - federation links 329
 - itadmin commands 323, 341
 - list federation links 330
 - list offer IDs 333
 - list proxy offer IDs 335
 - list service types 337
 - manage 323, 341
 - mask service type 337
 - modify administrative settings 326
 - modify federation link 330
 - obtain administrative settings 324
 - proxy offers 335
 - regular offers 333
 - remove federation link 331
 - remove offer 333

- remove proxy offer 335
- remove service type 338
- show federation link attributes 332
- show offer attributes 333
- show proxy offer attributes 335
- show service type attributes 338
- stop 328
- type repositories 337
- unmask service type 338
- transaction
 - begin 301
 - commit 302
 - resume 302
 - roll back 302
 - suspend 303
- transaction log files 177
- transient objects 11
- trd_admin
 - get 324
 - set 326
 - stop 328
- trd_link
 - create 329
 - list 330
 - modify 330
 - remove 331
 - show 332
- trd_offer
 - list 333
 - remove 333
 - show 333
- trd_proxy
 - list 335
 - remove 335
 - show 335
- trd_type
 - list 337
 - mask 337
 - remove 338
 - show 338
 - unmask 338
- tx
 - begin 301
 - commit 302
 - resume 302
 - rollback 302
 - suspend 303
- type repository ,manage 337

V

variable

- create 229
- manage in configuration 229
- modify 231
- remove 232
- show 232
- show setting 232

W

WELL_KNOWN_ADDRESSING_POLICY 122

Windows NT services 349

- accounts 355
- commands 352
- identify Orbix services 351
- install Orbix service 353
- logging 359
- manage 351
- obtain data 354
- obtain help on service 353
- pause background service 353
- prepare Orbix service 353
- run 354, 357
 - in file-based configuration 357
 - in repository-based configuration 357
- security 355
- stop Orbix service 354
- troubleshoot 361
- uninstall service 354, 360