Orbix Mainframe 6.3.1

Management User's Guide

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

2021-03-18

Contents

Part 2 Programmer's Guide

List of Figures

Preface

Orbix Mainframe provides full integration with the Orbix management infrastructure, which provides support for enterprise-level management across different platform and programming language environments. The Orbix management tools, integrated with the Orbix Adaptive Runtime Technology, enable seamless management of distributed enterprise applications.

Audience

Part 1 is aimed at z/OS systems programmers managing distributed enterprise applications.

Part 2 is aimed at z/OS application programmers writing distributed enterprise applications in C++ who wish to enable their applications for management by Orbix management tools. It assumes a prior knowledge of C++.

Organization of this guide

This guide is divided as follows:

Part 1, "Administrator's Guide"

This part is aimed at z/OS systems programmers. First it introduces Orbix enterprise management in general, and the tools used to manage distributed applications. Then it describes how to manage Orbix Mainframe services and monitor events.

Part 2, "Programmer's Guide"

This part is aimed at z/OS application programmers writing distributed enterprise applications in C++ who wish to enable their applications for management by Orbix management tools. It explains how to enable CORBA C++ applications for management, and display them in the Orbix management tools.

Related documentation

The Orbix Mainframe library includes the following related documentation:

- *CORBA Administrator's Guide*
- *CORBA Configuration Reference*
- *CORBA Programmer's Guide, C++ Edition*

The *Management User's Guide* in the Orbix library can also be referred to for more details.

The latest updates to the Orbix Mainframe documentation can be found at https://www.microfocus.com/documentation/orbix/.

Additional resources

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

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

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

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

Typographical conventions

This guide uses the following typographical conventions:

`Constant width`	Constant width (courier font) in normal text represents portions of code and literal names of items such as classes, functions, variables, and data structures. For example, text might refer to the `CORBA::Object` class. Constant width paragraphs represent code examples or information a system displays on the screen. For example: `#include <stdio.h>`

Italic	Italic words in normal text represent *emphasis* and *new terms*. Italic words or characters in code and commands represent variable values you must supply, such as arguments to commands or path names for your particular system. For example: `% cd /users/`*your_name* **Note:** Some command examples may use angle brackets to represent variable values you must supply. This is an older convention that is replaced with *italic* words or characters.

Keying conventions

This guide may use the following keying conventions:

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, Windows NT, Windows 95, or Windows 98 command prompt.
`...` . . .	Horizontal or vertical ellipses in format and syntax descriptions indicate that material has been eliminated to simplify a discussion.
[ ]	Brackets enclose optional items in format and syntax descriptions.
{ }	Braces enclose a list from which you must choose an item in format and syntax descriptions.
|	A vertical bar separates items in a list of choices enclosed in { } (braces) in format and syntax descriptions.

Part 1

Administrator's Guide

In this part

This part contains the following chapter:

CHAPTER 1

Introduction to Orbix Management

The Orbix management tools are a set of tools that enable you to manage component-based distributed enterprise applications. This chapter introduces the Orbix management tools and outlines typical administration tasks.

In this chapter

This chapter contains the following sections:

Orbix Management Tools

Overview

The Orbix management tools enable you to manage and configure component-based distributed enterprise applications. They are integrated with the Orbix *Adaptive Runtime Technology* (ART). This enables them to provide seamless management of Orbix and any applications developed using it.

Orbix management tools are not aimed solely at any specific technology (for example, CORBA or Web services), but provide a generic management paradigm that enable the application to be managed without the administrator requiring knowledge of the technology used to create that application.

Scope of Orbix management tools

Orbix management tools enable you to manage and configure distributed applications that have been developed using Orbix and Orbix Mainframe. For detailed information about the Orbix product range, see the web site:

https://www.microfocus.com/en-us/products/orbix/overview

Assumptions

Orbix management tools do not assume that you are familiar Orbix or Orbix Mainframe. What is required is a basic understanding of distributed applications, regardless of whether they are based on CORBA or Web services. In fact, you can use Orbix management tools to manage any C++ system that has been enabled for management.

Orbix Management Tools

Overview

Orbix management tools include the following main components:

- “Administrator Web Console”.
- “Orbix Management Service”
- “Orbix Configuration Explorer”.
- “Orbix Configuration Authority”.

Note: The Orbix Configuration Explorer is introduced here for the sake of completeness, but it is not supported with Orbix Mainframe.

Administrator Web Console

The *Administrator Web Console* provides a web browser interface to the Orbix management tools. It enables you to manage applications and application events from anywhere, without the need for download or installation. It communicates with the management service using HTTP (Hypertext Transfer Protocol), as illustrated in Figure 1.

Orbix Management Service

The *Orbix Management Service* is the central point of contact for accessing management information in a *domain*. A domain is an abstract group of managed server processes within a physical location. The management service is accessed by both the Administrator Web Console and the *Orbix Configuration Explorer*.

Note: Managed applications can be written in C++. The same management service process (`iona_services.management`) can be used by CORBA C++ applications.

Orbix Configuration Explorer

The *Orbix Configuration Explorer* is a Java graphical user interface (GUI) that enables you to manage your configuration settings. It communicates with your Configuration Repository (CFR) or configuration file using IIOP (Internet Inter-ORB Protocol).

Note: The Orbix Configuration Explorer is not supported with Orbix Mainframe. You must manually browse your Orbix Mainframe configuration file.

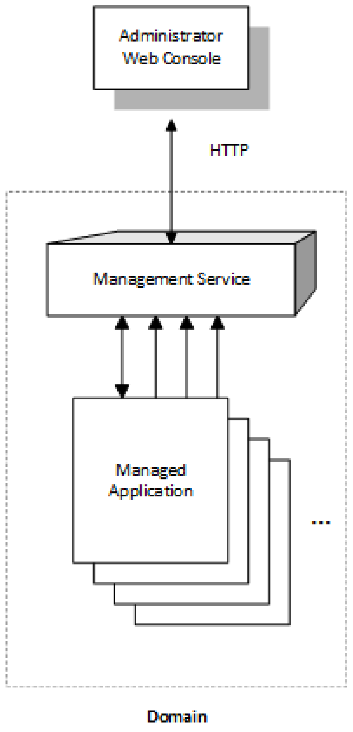


Figure 1: *Management Overview*

Orbix Configuration Authority

The *Orbix Configuration Authority* provides a web browser interface to descriptive information about all Orbix configuration settings. You can browse and search for information about Orbix configuration variables in your CFR or configuration file.

Additional features

Application programmers can add instructions to their code to monitor specific components in their system. This is known as adding management *instrumentation*.

Adding management instrumentation

Orbix products provide default instrumentation that publishes core information to the management service for any application built using these products.

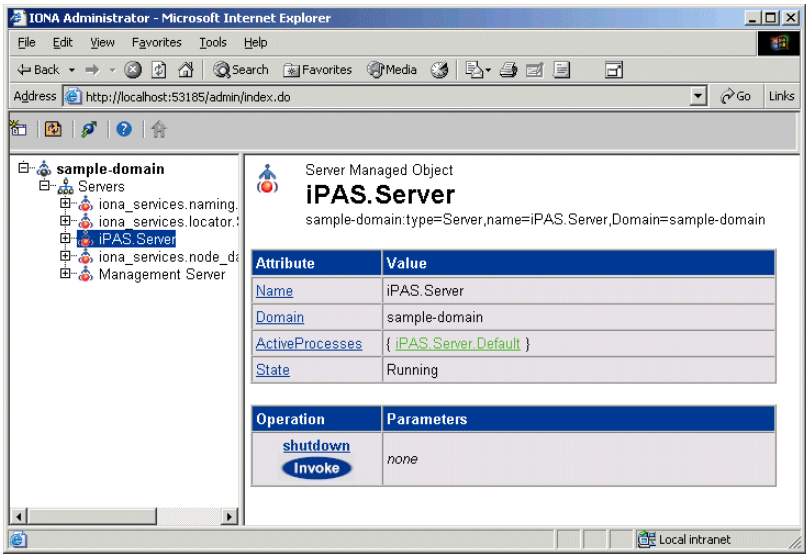
However, programmers might also wish to add custom instrumentation to an application to suit their needs. Orbix, therefore, enables full instrumentation of server code. For information on how to write instrumentation code, see "Programmer's Guide" on page 49.

Administrator Web Console

Overview

The Administrator Web Console provides a standard web browser interface to explore and manage distributed applications. The Administrator Web Console uses HTML and JavaScript to create a standard explorer view to represent the data.

Figure 2 shows an example Administrator Web Console interface.


IONA Administrator - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:53185/admin/index.do
Go Links
sample-domain
Servers
iona_services.naming.
iona_services.locator.
iPAS.Server
iona_services.node_d
Management Server
Local intranet

Server Managed Object

iPAS.Server

sample-domain:type=Server,name=iPAS.Server,Domain=sample-domain

Attribute	Value
Name	iPAS.Server
Domain	sample-domain
ActiveProcesses	{ iPAS.Server.Default }
State	Running

Operation	Parameters
shutdown Invoke	none

Figure 2: *Administrator Web Console*

Multiple applications and domains

You can use one instance of the Administrator Web Console to manage multiple applications in a single domain. You also can use multiple instances of the web console to manage multiple domains from a single machine. This is shown in Figure 3.

Interaction with the management service

Each Orbix management service makes management data available using a special URL. The management service is the central point of contact for management information in each domain. It publishes information about all managed servers within its domain.

Management architecture

Figure 3 gives an overview of the management architecture. Each Administrator Web Console interacts with one management service only. This means that each console can administer the managed servers in one of the two domains only.

Multiple instances of the web console can interact with the same management service through the same HTTP port.

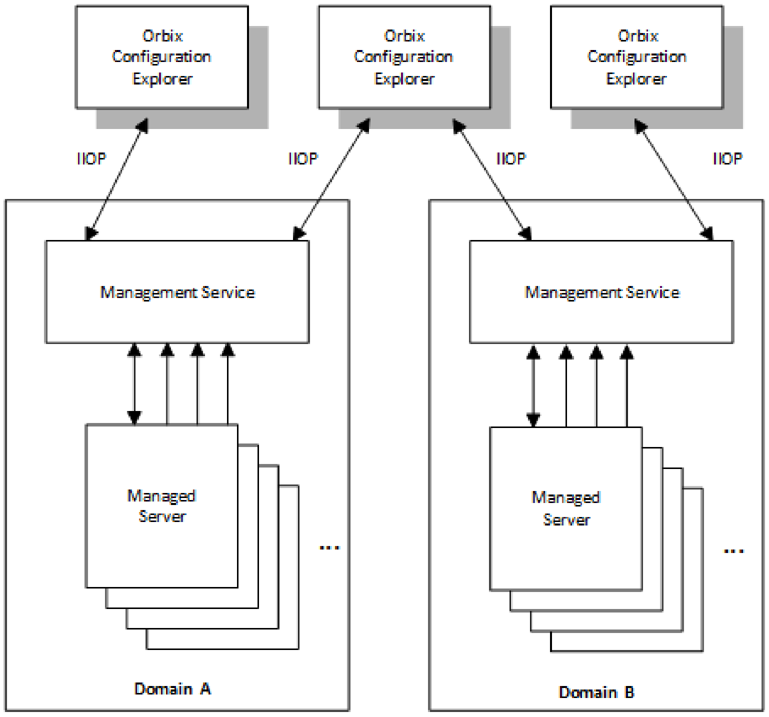


Figure 3: *Administrator Architecture*

Orbix Management Service

Overview

The Orbix management service is the central point of contact for accessing management information in a domain. The management service acts as a buffer between managed applications and management tools.

Management information

The management service maintains key state information, reducing the need to constantly access the managed applications, and thereby improving performance.

The management service stores and publishes information about all managed servers within its domain. It exposes attributes, operations, and events for all managed servers in a domain. The management service also stores information about user roles and passwords for each user in a domain.

Note: Each domain can have only one management service.

Key features

Key features provided by the management service are:

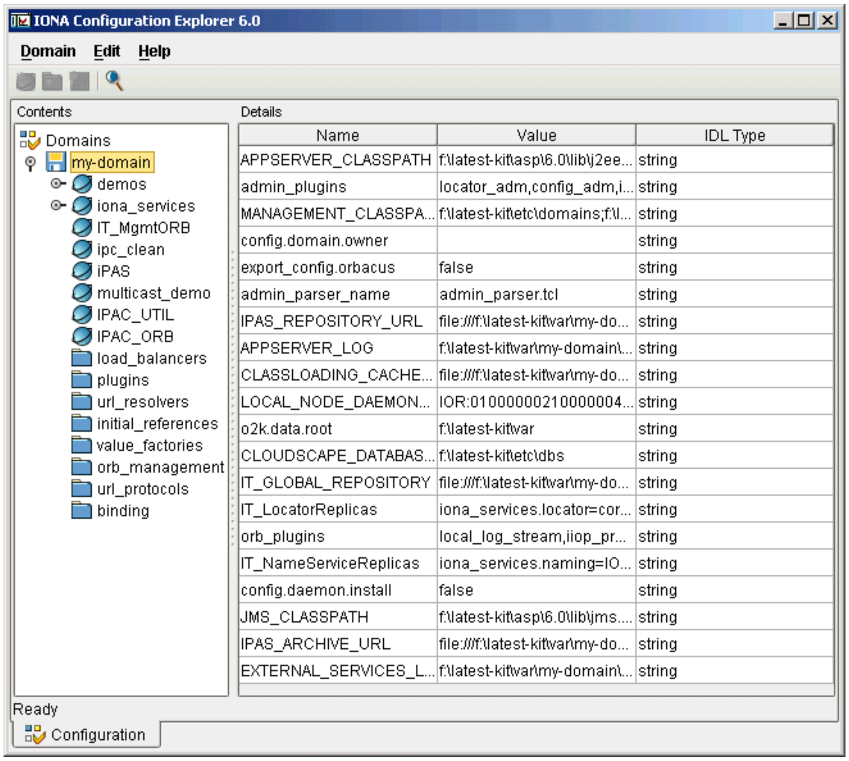
- Centralized repository for all management information.
- Centralized collection of event logging information.
- Persistent storage of event log and agent information.
- Load management gateway plugins (for example, an SNMP plugin).
- Capability to terminate server processes.

Orbix Configuration Explorer

Overview

The Orbix Configuration Explorer is an intuitive Java GUI that enables you to view, modify, and search for configuration settings.

In Figure 4, the **Contents** pane on the left shows the configuration scopes and namespaces displayed for a domain named `my-domain`. The **Details** pane on the right displays the configuration variables and their values. Clicking on a icon on the left displays its associated variables on the right.



Name	Value	IDL Type
APPSERVER_CLASSPATH	f:\latest-kit\asp\6.0\lib\j2ee...	string
admin_plugins	locator_adm,config_adm,i...	string
MANAGEMENT_CLASSPA...	f:\latest-kit\etc\domains;f:\l...	string
config.domain.owner		string
export_config.orbacus	false	string
admin_parser_name	admin_parser.tcl	string
IPAS_REPOSITORY_URL	file:///f:\latest-kit\var\my-do...	string
APPSERVER_LOG	f:\latest-kit\var\my-domain\...	string
CLASSLOADING_CACHE...	file:///f:\latest-kit\var\my-do...	string
LOCAL_NODE_DAEMON...	IOR:010000002100000004...	string
o2k.data.root	f:\latest-kit\var	string
CLOUDSCAPE_DATABAS...	f:\latest-kit\etc\dbs	string
IT_GLOBAL_REPOSITORY	file:///f:\latest-kit\var\my-do...	string
IT_LocatorReplicas	iona_services.locator=cor...	string
orb_plugins	local_log_stream,iiop_pr...	string
IT_NameServiceReplicas	iona_services.naming=IO...	string
config.daemon.install	false	string
JMS_CLASSPATH	f:\latest-kit\asp\6.0\lib\jms....	string
IPAS_ARCHIVE_URL	file:///f:\latest-kit\var\my-do...	string
EXTERNAL_SERVICES_L...	f:\latest-kit\var\my-domain\...	string

Figure 4: *Orbix Configuration Explorer*

Multiple Domains

You can use a single instance of the Orbix Configuration Explorer to manage configuration of multiple domains, both locally and on remote host machines. The Orbix Configuration Explorer communicates with CFRs in any domains that it can contact. It can also read file-based domains where they are locally visible.

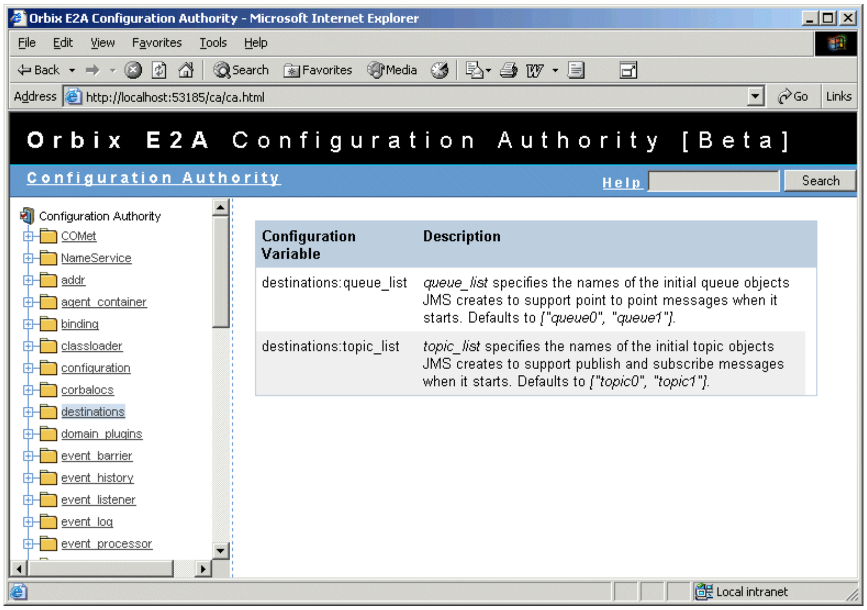
Note: Because the CFR is not supported with Orbix Mainframe, and the Configuration Explorer is run off-host, there is currently no way for the Configuration Explorer to interact with an Orbix Mainframe configuration domain. Therefore, you must manually browse the configuration file located in `HLQ.DOMAINS` in your Orbix Mainframe installation.

Orbix Configuration Authority

Overview

The Orbix Configuration Authority displays text descriptions of all Orbix configuration settings. Its web browser interface enables you to navigate to and search for configuration information, as shown in Figure 5.

The navigation tree, on the left of the screen displays a hierarchical list of configuration namespaces and variables. The details pane, on the right, displays information about the configuration variables associated with the selected node on the tree.



Configuration Variable	Description
destinations:queue_list	*queue_list* specifies the names of the initial queue objects JMS creates to support point to point messages when it starts. Defaults to *["queue0", "queue1"]*.
destinations:topic_list	*topic_list* specifies the names of the initial topic objects JMS creates to support publish and subscribe messages when it starts. Defaults to *["topic0", "topic1"]*.

Figure 5: *Orbix Configuration Authority*

The **Search** box located at the top left of the screen enables you to search for information about configuration variables containing a specified text string.

Orbix Management Tasks

Overview

Typical Orbix management tasks that you can perform include:

- "Managing domains".
- "Managing servers".
- "Monitoring events".
- "Managing configuration settings".
- "Getting started"

This section gives a quick overview of these tasks, and shows where you can find further information in this book.

Managing domains

Typical domain management tasks include:

- Viewing domains.
- Monitoring domain status (whether it is running or stopped).

For more details of how to manage domains, using the Administrator Web Console, see the *Management User's Guide* at https://supportline.microfocus.com/documentation/books/Orbix/639/management_user_guide_639.pdf.

Managing servers

Typical server management tasks include:

- Viewing servers.
- Monitoring server status (whether it is running or inactive).
- Controlling servers (shutting down, setting attributes, and invoking operations).

For more details of how to manage servers, using the Administrator Web Console, see the *Management User's Guide* at https://supportline.microfocus.com/documentation/books/Orbix/639/management_user_guide_639.pdf.

Monitoring events

Typical event management tasks include:

- Selecting a domain in which to manage events.
- Viewing full details of an event.
- Setting event viewing options. For example, you can set the number of events viewed, set the kind of events viewed.

For more details of how to manage events, using the Administrator Web Console, see the *Management User's Guide* at https://supportline.microfocus.com/documentation/books/Orbix/639/management_user_guide_639.pdf.

Managing configuration settings

Typical configuration management tasks include:

- Loading up a domain.
- Viewing configuration settings.
- Searching your configuration.
- Finding text descriptions of configuration variables.

For more details of how to find text descriptions of configuration variables using the Orbix Configuration Authority and manage configuration settings for the management service, see the *Management User's Guide* at https://supportline.microfocus.com/documentation/books/Orbix/639/management_user_guide_639.pdf.

Getting started

For details of how to get started with the Administrator Web Console, see the *Management User's Guide* at https://supportline.microfocus.com/documentation/books/Orbix/639/management_user_guide_639.pdf.

Managing Orbix Mainframe Services and Events

Orbix Mainframe provides full integration with the Orbix Management infrastructure. This allows Orbix servers running on the mainframe to be monitored from a centralized location, using Orbix Administrator. This chapter provides details on Orbix Mainframe instrumentation and the configuration items involved in managing Orbix Mainframe services. It also explains how to use the Administrator Web Console to monitor events.

In this chapter

This chapter discusses the following topics:

Introduction

Overview

This section provides an introductory overview of how Orbix management tools components are used in the management of Orbix services running on z/OS.

Graphical overview

Figure 6provides a graphical overview of how Orbix management tools components such as the Administrator Web Console and Orbix Management Service are used in the management of Orbix services on z/OS.

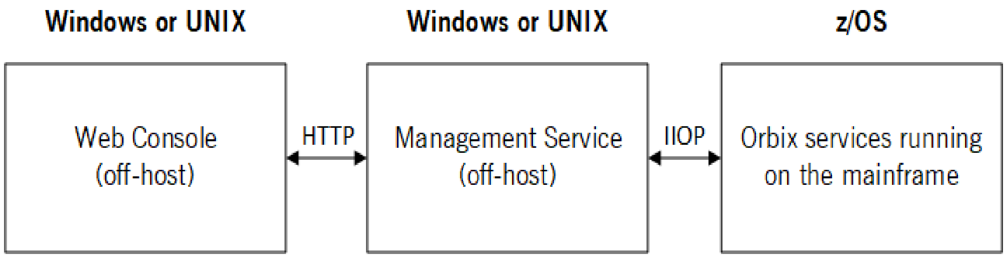

Windows or UNIX
Windows or UNIX
z/OS
Web Console (off-host)
HTTP
Management Service (off-host)
IIOP
Orbix services running on the mainframe

Figure 6: *Orbix Management Tools Integration with z/OS*

As shown in Figure 6, the Web Console and Management Service run off-host and communicate with each other over HTTP. The Management Service and the services running on z/OS communicate with each other over IIOP.

C++ and Java management

Orbix Mainframe fully supports the C++ Management runtime and C++ Management APIs for developing instrumentation capabilities within your Orbix applications. However, Orbix Mainframe does not include the Java Management Service component. Instead, the Java Management Service must be deployed in an off-host Orbix domain, and must be contactable by the Orbix Mainframe environment.

Orbix Mainframe Instrumentation

Overview

This section outlines the components involved in Orbix Mainframe instrumentation. It discusses the following topics:

- "Instrumentation components".
- "Instrumentation demonstration".

Instrumentation components

Orbix Mainframe instrumentation consists of:

- Default core instrumentation—all Orbix applications can be configured to expose ORB instrumentation statistics.
- Naming Service—the Orbix Naming Service supports instrumentation specific to management of, for example, naming contexts and load balancing.
- C++ custom development—the Orbix C++ Management API allows you to develop customized instrumentation for your own Orbix applications.

For more details on adding management instrumentation to an application, see "Programmer's Guide" on page 49.

Instrumentation demonstration

An instrumentation demonstration is provided in the UNIX System Services component of your Orbix Mainframe installation, as follows (where *install_dir* represents the full path to your Orbix Mainframe installation on UNIX System Services):

```
install_dir/asp/Version/demos/corba/pdk/instrumented_plugin
```

This instrumentation demonstration illustrates how to use the main Management APIs and how to write your own Generic Service application. You can use an ORB plug-in approach to build the Management code, to instrument existing services such as the CICS and IMS server adapters.

Management Configuration

Overview

This section provides details of the steps involved in configuring the management of Orbix services on z/OS. It also describes each of the associated configuration items that need to be set on the mainframe host. It discusses the following topics:

- "Domain interaction"
- "Configuration steps"

Domain interaction

This section assumes that an off-host Orbix domain is available and has been configured to enable management. It is also assumed that the Orbix Mainframe domain is compatible with this off-host Orbix domain, and that communication between them has already been verified. For example, if the off-host domain has been configured to be fully secure, the Orbix Mainframe domain must be deployed with a TLS domain. Before you attempt to run any managed services on z/OS, you should first confirm that the off-host locator and the other off-host services can be contacted successfully (for example, by using the `itadmin` or `ORXADMIN` tool from z/OS).

Configuration steps

The steps to enable the management of Orbix services on z/OS are:

1. Add the Management Service initial reference configuration settings to the Orbix Mainframe configuration file at the global scope, as follows:

```
initial_references:IT_MgmtService:reference = "IOR:000…";
initial_references:IT_MgmtServiceUser:reference =
   "IOR:000…";
initial_references:IT_MgmtServiceSec:reference = "IOR:000…";
```

   The IOR settings can be obtained from the off-host configuration domain.

2. Enable ORB instrumentation by adding the following configuration setting to the configuration scope for the relevant server:

```
plugins:orb:is_managed = "true";
```

3. Ensure that each service has a unique server ID across your entire management domain by adding the following configuration item to the configuration scope for the appropriate server:

```
plugins:it_mgmt:managed_server_id:name = "…"
```

Note: By default, the ORB name of the relevant server is used as the ID for a particular service. For example, to specify a unique server name for the locator service, you can choose to set the preceding variable to `"iona_services.locator.`*`mainframe_host`*`"`, where *`mainframe_host`* is the local TCP/IP hostname.

4. Enable instrumentation of the Naming Service by adding the following configuration settings to the `iona_services.naming` configuration scope:

```
plugins:orb:is_managed = "true";
plugins:naming:is_managed = "true";
plugins:it_mgmt:managed_server_id:name =
    "iona_services.naming.mainframe_bost";
```

5. If you are interested in viewing the event log from the management console, you must configure the managed service to log events to a file. For example:

```
plugins:local_log_stream:filename =
    "/opt/microfocus/var/logs/imsa.log";
```

Monitoring Orbix Services on z/OS

Overview

This section outlines the steps to monitor Orbix services on z/OS.

Steps

The steps to monitor Orbix services on z/OS are:

- Ensure that the Orbix off-host services are running. This includes the Management Service.
- Start the Orbix Mainframe managed services. On starting, these services attempt to register themselves with the off-host Management Service.

  Note: If a managed server is unable to contact the off-host Management Service, it starts and continues to run without issuing a warning message. If there is a communication problem, for example, the managed server does not appear in the Management console.

- Start the Web Console. After the various services have been successfully deployed, you can use the Web Console to contact the Management Service, to monitor the state of each of the various services.

Note: For more details on using the off-host Web Console and the off-host Management Service refer to the *Management User's Guide* at https://supportline.microfocus.com/documentation/books/Orbix/639/management_user_guide_639.pdf.

Managing Events in the Web Console

Overview

This chapter explains how to use the Administrator Web Console to monitor events. It explains how to start its Events Console, and view events for a domain.

The Administrator Web Console's Events Console enables you to view events generated by managed servers. The events console shows an up-to-date list of events in reverse chronological order. You can customize the severity of events and apply filters to selectively view events.

In this section

This section discusses the following topics:

Starting the Events Console

Overview

This subsection explains how to start the Administrator Web Console's Events Console.

Using the Events Button

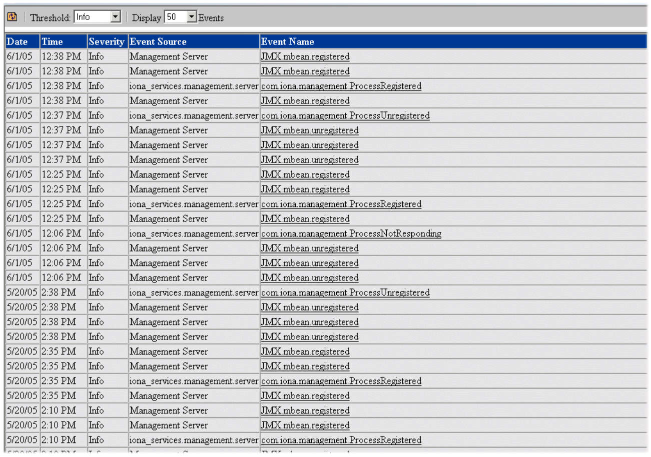
To start the Events Console, click the Events button in the Administrator Web Console toolbar, as shown in Figure 7.

Figure 7: *Events Button*

If an events console is already open, subsequent clicks on this button bring the web console to the foreground.

Example Events Console

An example Events Console started from the web console is shown in Figure 8. The events are shown in a list starting with the most recent event at the top.



Threshold: Info | Display 50 Events

Date	Time	Severity	Event Source	Event Name
6/1/05	12:38 PM	Info	Management Server	JMX.mbean.registered
6/1/05	12:38 PM	Info	Management Server	JMX.mbean.registered
6/1/05	12:38 PM	Info	iona_services.management.server	com.iona.management.ProcessRegistered
6/1/05	12:38 PM	Info	Management Server	JMX.mbean.registered
6/1/05	12:37 PM	Info	iona_services.management.server	com.iona.management.ProcessUnregistered
6/1/05	12:37 PM	Info	Management Server	JMX.mbean.unregistered
6/1/05	12:37 PM	Info	Management Server	JMX.mbean.unregistered
6/1/05	12:37 PM	Info	Management Server	JMX.mbean.unregistered
6/1/05	12:25 PM	Info	Management Server	JMX.mbean.registered
6/1/05	12:25 PM	Info	Management Server	JMX.mbean.registered
6/1/05	12:25 PM	Info	iona_services.management.server	com.iona.management.ProcessRegistered
6/1/05	12:25 PM	Info	Management Server	JMX.mbean.registered
6/1/05	12:06 PM	Info	iona_services.management.server	com.iona.management.ProcessNotResponding
6/1/05	12:06 PM	Info	Management Server	JMX.mbean.unregistered
6/1/05	12:06 PM	Info	Management Server	JMX.mbean.unregistered
6/1/05	12:06 PM	Info	Management Server	JMX.mbean.unregistered
5/20/05	2:38 PM	Info	iona_services.management.server	com.iona.management.ProcessUnregistered
5/20/05	2:38 PM	Info	Management Server	JMX.mbean.unregistered
5/20/05	2:38 PM	Info	Management Server	JMX.mbean.unregistered
5/20/05	2:38 PM	Info	Management Server	JMX.mbean.unregistered
5/20/05	2:35 PM	Info	Management Server	JMX.mbean.registered
5/20/05	2:35 PM	Info	Management Server	JMX.mbean.registered
5/20/05	2:35 PM	Info	iona_services.management.server	com.iona.management.ProcessRegistered
5/20/05	2:35 PM	Info	Management Server	JMX.mbean.registered
5/20/05	2:10 PM	Info	Management Server	JMX.mbean.registered
5/20/05	2:10 PM	Info	Management Server	JMX.mbean.registered
5/20/05	2:10 PM	Info	iona_services.management.server	com.iona.management.ProcessRegistered

Figure 8: *Events Console*

Viewing the Events Console

Overview

This section explains how to use the Administrator Web Console's Events Console. It includes the following:

- "Viewing Events in a Domain".
- "Refreshing the Event List".
- "Setting the Number of Events Displayed".
- "Setting the Event Threshold".
- "Information Displayed in the Event List".
- "Viewing Full Details of an Event".
- "Filtering Events".

Viewing Events in a Domain

Events are always shown on a per-domain basis. To view events from a different domain, start a web console connecting to the domain's management service and launch the events console from there. For more details of prerequisites to starting the web console, see the *Management User's Guide* at https://supportline.microfocus.com/documentation/books/Orbix/639/management_user_guide_639.pdf.

Refreshing the Event List

The event display shows an up-to-date list of events when first started. The display is not updated automatically. To refresh the display, click the **Refresh** button in the toolbar, as shown in Figure 24.

Figure 9: *Refresh Button*

Setting the Number of Events Displayed

To set the maximum number of events being retrieved from the management server, click the drop-down box at the **Display Events** field at the top of the console.

Setting the Event Threshold

The **Threshold** setting specifies the lowest severity of events that you want to include in the displayed list. There are four severities:

- `Critical`
- `Error`
- `Warning`
- `Info`

The highest event severity is `Critical` and the lowest is `Info`.

To set the events threshold, click the **Threshold** drop-down box at the top left of the console.

Information Displayed in the Event List

The event list shows the following summary information about each event:

- Date and time of the event.
- Severity of the event.
- Agent that created the event.
- Name of the event.

Viewing Full Details of an Event

To get comprehensive details of a particular event, simply click the event in the event list.

Filtering Events

You can also customize the severity of events and apply filters to selectively view events by modifying **shared** filters for a domain. For more information, see the section on Management Service Configuration in the *Management User's Guide* at https://supportline.microfocus.com/documentation/books/Orbix/639/management_user_guide_639.pdf.

Viewing the Event Log

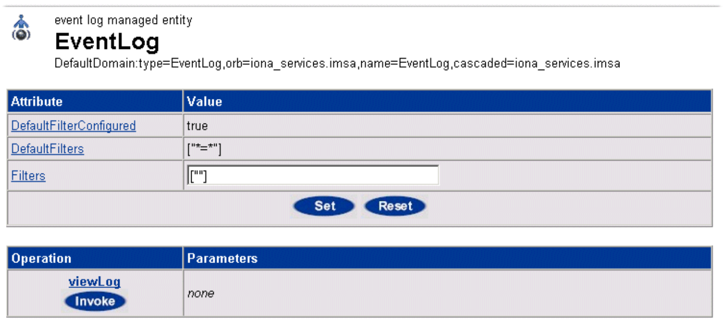
Overview

This section explains how to open and view the Administrator Web Console's event log for a particular server.

Note: To view the event log for the IMS or CICS server adapter, the `plugins:local_log_stream:filename` configuration item must be set in the adapter's configuration scope.

Opening the event log

To open the event log for a particular server, click the EventLog option under that server name on the left display panel of your management service browser. This opens an **EventLog** panel for the server similar to that shown in Figure 10.



event log managed entity

EventLog

DefaultDomain:type=EventLog,orb=iona_services.imsa,name=EventLog,cascaded=iona_services.imsa

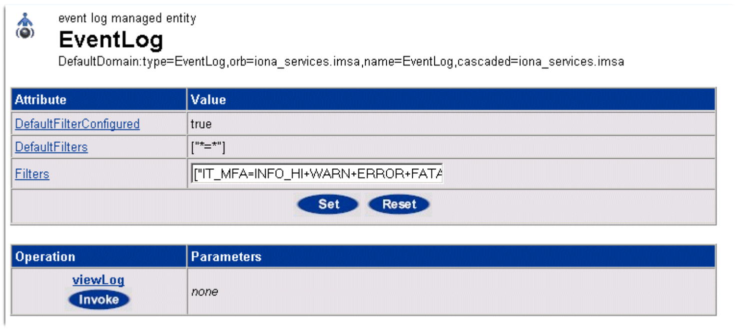
Attribute	Value
DefaultFilterConfigured	true
DefaultFilters	["*=*"]
Filters	[""]
	Set Reset

Operation	Parameters
viewLog Invoke	none

Figure 10: *Example of an EventLog Panel*

Setting the log filters

You can use the **Filters** field to determine the level of logging information that is to be generated for a particular plug-in. For example, Figure 11 shows a filters setting of `"IT_MFA=INFO_HI+WARN+ERROR+FATAL"`, to generate logging information for CICS or IMS server adapter events.



event log managed entity

EventLog

DefaultDomain:type=EventLog,orb=iona_services.imsa,name=EventLog,cascaded=iona_services.imsa

Attribute	Value
DefaultFilterConfigured	true
DefaultFilters	["*=*"]
Filters	["IT_MFA=INFO_HI+WARN+ERROR+FATA

Operation	Parameters
viewLog Invoke	*none*

Figure 11: *Setting the Log Filters for the IT_MFA Plug-in*

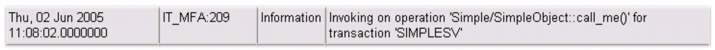
To save a setting in the **Filters** field, click **Set**. If you want to override any changes and return to the prior settings, click **Reset**.

Note: The **Reset** button can only override settings that have not already been saved via the **Set** button.

Opening the log viewer

Click the Invoke button on the **Event Log** to open the **Log Viewer** panel. This displays all the logged events for the plug-in(s) that you specified when setting the log filters.

Figure 12 shows an example of a logged event for an IMS server adapter contacted by the simple client demonstration supplied with your Orbix Mainframe installation.



Thu, 02 Jun 2005 11:08:02.0000000	IT_MFA:209	Information	Invoking on operation 'Simple/SimpleObject::call_me()' for transaction 'SIMPLESV'

Figure 12: *Example of Logged Event for IMS Adapter in the Log Viewer*

As shown in Figure 12, the following information is displayed for each logged event:

- The date and time of the event.
- The subsystem it relates to.
- The level of event (that is, Information, Warning, or Error).
- Details of the event.

Navigating the log viewer

It might not be possible for all event details to be displayed on one screen. To see details of more events, click the **Prev** and **Next** links on the **Log Viewer** as appropriate. If you click the **Back to Details** link, this reopens the **Event Log** panel.

Enterprise Performance Logging

Micro Focus's performance logging plugins enable Orbix to integrate effectively with Enterprise Management Systems (EMS).

In this chapter

This chapter contains the following sections:

Introduction

Overview

Performance logging plugins enable Orbix to integrate effectively with *Enterprise Management Systems* (EMS). The performance logging plugins can also be used in isolation or as part of a custom-made solution.

Enterprise Management Systems enable system administrators and production operators to monitor enterprise-critical applications from a single management console. This enables them to quickly recognize the root cause of problems that may occur, and take remedial action (for example, if a machine is running out of disk space).

Performance logging

When performance logging is configured, you can see how each Orbix server is responding to load. The performance logging plugins log this data to file or `syslog`. Your EMS can read the performance data from these logs, and use it to initiate appropriate actions, (for example, issue a restart to a server that has become unresponsive, or start a new replica for an overloaded cluster).

Configuring Performance Logging

Overview

This section explains how to manually configure performance logging. This section includes the following:

- "Performance logging plugins".
- "Monitoring Orbix requests".
- "Logging to a file or syslog".
- "Configuring a server ID".
- "Configuring a client ID".
- "Monitoring the Orbix work queue".
- "Configuring the CICS adapter to use performance logging".

Performance logging plugins

The performance logging component consists of three plugins:

Table 1: *Performance logging plugins*

Plugin	Description
Response time logger	Monitors response times of requests as they pass through the Orbix binding chains. This can be used to collect response times for CORBA, RMI-IIOP or HTTP calls.
Request counter	Performs the same function for Artix as the Response time logger does for Orbix.
Response time collector	Periodically harvests data from the response time logger and request counter plugins and logs the results.
MBean monitor	Periodically harvests statistics associated with MBean attributes (for example, monitoring the length of the ORB work queue).

Monitoring Orbix requests

You can use performance logging to monitor both Orbix server and client requests.

Monitoring server requests

To monitor Orbix server requests, perform the following configuration steps:

1. Add `it_response_time_logger` to the servlet binding list for the server you wish to instrument. For example:

```
binding:servlet_binding_list= [
   "it_response_time_logger + it_servlet_context + it_character_encoding
   + it_locale + it_naming_context + it_exception_mapping + it_http_sessions
   + it_web_security + it_servlet_filters  + it_web_redirector + it_web_app_activator "
];
```

2. Add `it_response_time_logger` to the server binding list for the server. For example:

```
binding:server_binding_list=[
   "it_response_time_logger+it_naming_context+CSI+j2eecsi+OTS+it_security_role_mapping",
   "it_response_time_logger+it_naming_context+OTS+it_security_role_mapping",
   "it_response_time_logger+it_naming_context + CSI+j2eecsi+it_security_role_mapping",
   "it_response_time_logger+it_naming_context+it_security_role_mapping",
   "it_response_time_logger+it_naming_context", "it_response_time_logger"
];
```

3. Add `it_response_time_logger` to the `orb_plugins` list for the server. For example:

```
orb_plugins=[
    "it_servlet_binding_manager", "it_servlet_context",
    "it_http_sessions", "it_servlet_filters", "http",
    "it_servlet_dispatch", "it_exception_mapping", "it_naming_context",
    "it_web_security", "it_web_app_activator",
    "it_default_servlet_binding", "it_security_service", "it_character_encoding",
    "it_locale", "it_classloader_servlet","it_classloader_mapping",
    "it_web_redirector", "it_deployer",
    "it_response_time_logger"
];
```

Monitoring client requests

To monitor Orbix client requests, add `it_response_time_logger` to the client binding list for the server. For example:

```
binding:client_binding_list = [
"it_response_time_logger+DemoOS+OTS+POA_Coloc","it_response_time_logger+DemoOS+POA_Coloc",
"it_response_time_logger+OTS+POA_Coloc", "it_response_time_logger+POA_Coloc",
   "it_response_time_logger+DemoOS+OTS+GIOP+IIOP", "it_response_time_logger+DemoOS+GIOP+IIOP",
   "it_response_time_logger+OTS+GIOP+IIOP", "it_response_time_logger+GIOP+IIOP",
   "it_response_time_logger"
];
```

Logging to a file or syslog

You can configure the collector plugin to log data either to a file or to `syslog`.

C++ configuration

The following example configuration for a C++ application results in performance data being logged to `/var/log/my_app/perf_logs/treasury_app.log` every 90 seconds:

```
plugins:it_response_time_collector:period = "90";
plugins:it_response_time_collector:filename =
"/var/log/my_app/perf_logs/treasury_app.log";
```

If you do not specify the response time period, it defaults to 60 seconds.

Note: You may only log data to a file in z/OS UNIX System Services.

Logging to a syslog daemon

You can configure the collector to log to a syslog daemon or Windows event log, as follows:

```
plugins:it_response_time_collector:system_logging_enabled =
    "true";
plugins:it_response_time_collector:syslog_appID = "treasury";
```

The `syslog_appid` enables you to specify your application name that is prepended to all syslog messages. If you do not specify this, it defaults to `iona`.

Configuring a server ID

You can configure a server ID that will be reported in your log messages. This server ID is particularly useful in the case where the server is a replica that forms part of a cluster.

In a cluster, the server ID enables management tools to recognize log messages from different replica instances. You can configure a server ID as follows:

```
plugins:it_response_time_collector:server-id = "Locator-1";
```

This setting is optional; and if omitted, the server ID defaults to the ORB name of the server. In a cluster, each replica must have this value set to a unique value to enable sensible analysis of the generated performance logs.

Configuring a client ID

You can also configure a client ID that will be reported in your log messages. Specify this using the `client-id` configuration variable, for example:

```
plugins:it_response_time_collector:client-id = "my_client_app";
```

This setting enables management tools to recognize log messages from client applications. This setting is optional; and if omitted, it is assumed that that a server is being monitored.

Monitoring the Orbix work queue

The `it_mbean_monitoring` plug-in enables you to periodically harvest statistics associated with MBean attributes. This plug-in can be used to monitor the work queue MBean associated with a particular ORB. Work queues are used to control the flow incoming requests.

To monitor an ORB work queue MBean, perform the following steps:

1. Add `it_mbean_monitoring` to the `orb_plugins` list of the ORB whose work queue you wish to monitor.

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

2. When `it_mbean_monitoring` is on your `orb_plugins` list, you can enable monitoring of the ORB work queue using the following variable:

```
plugins:it_mbean_monitoring:workqueue = "true";
```

3. The MBean attributes that are monitored by the plug-in are sampled periodically. The sampling interval is specified in milliseconds using the following variable:

```
plugins:it_mbean_monitoring:sampling_period = "100";
```

4. The response time collector plug-in is used to periodically log the MBean data. You must specify the following variables for the collector:

```
plugins:it_response_time_collector:period = "10";
```

C++ applications

```
plugins:it_response_time_collector:filename = "testing_mbeans.log";
```

For more information, see also "MBean log message formats" on page 39.

Configuring the CICS adapter to use performance logging

To enable the CICS server adapter to use performance logging, perform the following configuration steps:

1. Add `it_response_time_logger` to the ORB plugins list for the adapter. For example:

   ```
   orb_plugins = ["…", "it_response_time_logger"];
   ```

   Note: Ensure that you have a management license available.

2. Add `it_response_time_logger` to the server binding list for the adapter. For example:

   ```
   binding:server_binding_list = ["it_response_time_logger",
       ""];
   ```

   Note: In this case, the `""` at the end of the server binding list is required.

3. Add the following configuration items to the `iona_services.cicsa` scope:

```
# update the log every 30 seconds
plugins:it_response_time_collector:period = "30";
# the id of the server for the log output
plugins:it_response_time_collector:server-id = "ORXCICSA";
# location of the log
plugins:it_response_time_collector:filename =
    "/home/fred/mycicsperf.log";
```

The following is an example of output from the performance log where a nested sequences client, a simple client, an `mfa list` and an `mfa resolve` have been run against the CICS adapter:

```
2006-10-18 10:08:22 server=ORXCICSA status=starting_up
2006-10-18 10:08:22 server=ORXCICSA status=running
2006-10-18 10:08:52 server=ORXCICSA status=running
2006-10-18 10:09:22 server=ORXCICSA status=running
2006-10-18 10:09:22 server=ORXCICSA [ operation=test_bounded ] count=1 avg=110 max=110 min=110
    int=30001 oph=119
2006-10-18 10:09:22 server=ORXCICSA [ operation=test_unbounded ] count=1 avg=809 max=809 min=809
    int=30001 oph=119
2006-10-18 10:09:52 server=ORXCICSA status=running
2006-10-18 10:09:52 server=ORXCICSA [ operation=call_me ] count=1 avg=793 max=793 min=793
    int=29998 oph=120
2006-10-18 10:10:22 server=ORXCICSA status=running
2006-10-18 10:10:22 server=ORXCICSA [ operation=_get_currentMappings ] count=1 avg=0 max=0 min=0
    int=30000 oph=120
2006-10-18 10:10:52 server=ORXCICSA status=running
2006-10-18 10:11:22 server=ORXCICSA status=running
2006-10-18 10:11:52 server=ORXCICSA status=running
2006-10-18 10:12:22 server=ORXCICSA status=running
2006-10-18 10:12:22 server=ORXCICSA [ operation=resolve ] count=1 avg=0 max=0 min=0 int=29999
    oph=120
2006-10-18 10:12:52 server=ORXCICSA status=running
2006-10-18 10:12:57 server=ORXCICSA status=shutdown_started
2006-10-18 10:12:57 server=ORXCICSA status=shutdown_complete
```

Logging Message Formats

Overview

This section describes the logging message formats used by Orbix and related products. It includes the following:

- "Orbix log message format".
- "Artix log message format".
- "MBean log message formats".
- "Simple life cycle message formats".

Orbix log message format

Performance data is logged in a well-defined format. For Orbix applications, this format is as follows:

```
YYYY-MM-DD HH:MM:SS server=serverID [operation=name] count=n
    avg=n max=n min=n int=n oph=n
```

Table 2: *Orbix log message format arguments*

Argument	Description
`server`	The server ID of the process that is logging the message.
`operation`	The name of the operation for CORBA invocations or the URI for requests on servlets.
`count`	The number of operations of invoked (IIOP). or The number of times this operation or URI was logged during the last interval (HTTP).
`avg`	The average response time (milliseconds) for this operation or URI during the last interval.
`max`	The longest response time (milliseconds) for this operation or URI during the last interval.
`min`	The shortest response time (milliseconds) for this operation or URI during the last interval.

Table 2: *Orbix log message format arguments*

Argument	Description
`int`	The number of milliseconds taken to gather the statistics in this log file.
`oph`	Operations per hour.

Artix log message format

The format for Artix log messages is as follows:

```
YYYY-MM-DD HH:MM:SS server=serverID [namespace=nnn service=sss
   port=ppp operation=name] count=n avg=n max=n min=n int=n
   oph=n
```

Table 3: *Artix log message format arguments*

Argument	Description
`server`	The server ID of the process that is logging the message.
`namespace`	An Artix namespace.
`service`	An Artix service.
`port`	An Artix port.

The combination of namespace, service and port above denote a unique Artix endpoint. The description for the remainder of the fields are the same as for Orbix messages.

MBean log message formats

The format for the mbean monitoring log message is as follows:

```
12004-09-23 15:24:17,093 monitored_object=full-object-name-for-mbean
   object_alias=user-friendly-name count=n avg=n max=n min=n period=n
```

Table 4: *MBean log message format arguments*

`monitored_object`	The MBean being monitored (for example, `DefaultDomain:type=AutoWorkqueue,orb=_it_orb_id_1,name=Workqueue_1`).
`object_alias`	A user-friendly name for MBean being monitored (for example, `test.management.logging_mbeans.ORBWorkQueue`).
`count`	The number of times the MBean attribute has been sampled during this logging period.
`avg`	The average value for the attribute being monitored.
`max`	The maximum value for the attribute being monitored.
`min`	The minimum value for the attribute being monitored.
`period`	The sampling interval specified in milliseconds.

Simple life cycle message formats

The server also logs simple life cycle messages. All servers share the following common format.

```
YYYY-MM-DD HH:MM:SS server=serverID status=current_status
```

Table 5: *Simple life cycle message format arguments*

Argument	Description
`server`	The server ID of the process that is logging the message.
`status`	A text string describing the last known status of the server (for example, `starting_up`, `running`, `shutting_down`).

Remote Performance Logging

Overview

The performance logging plug-ins can be configured to log data to a local file or to a remote endpoint. Depending on your specific architecture, it might not always be desirable or feasible to deploy the required management tools on a particular platform (for example, on z/OS). In this case, it would not be appropriate to persist the performance logging data to a local file, because there would be no local application to consume it.

In some situations, NFS or a similar file sharing mechanism might be used to persist data across your distributed system. However, security and performance concerns often prevent the use of such protocols. In such cases, Orbix provides a remote logging facility for the purposes of sending logging data to a remote endpoint where the data can be persisted and subsequently consumed by an application that is native to that remote system.

Components of a remote logging framework

The components of a remote logging framework are as follows:

- The performance logging *collector* plug-in runs in a deployed application on the source host. This is the host that sends its logging data to a remote endpoint. The collector is configured to harvest the required performance logging data and to write this data to a remote CORBA endpoint (instead of, for example, to a local file on the source host).

  Note: Remote logging is only supported in the C++ version of the performance logging collector plug-in.

- The *remote logger daemon* is an Orbix application that is deployed on the remote target host. It loads the remote log receiver servant, which is accepts the performance logging data from the source applications and logs this data to a local file on the target host.
- The *EMS component* (for example, a Tivoli or BMC Patrol agent) runs on the remote target host. It consumes the data from the file and propagates the performance information to the centralized region manager.

Figure 13 shows how remote logging works.

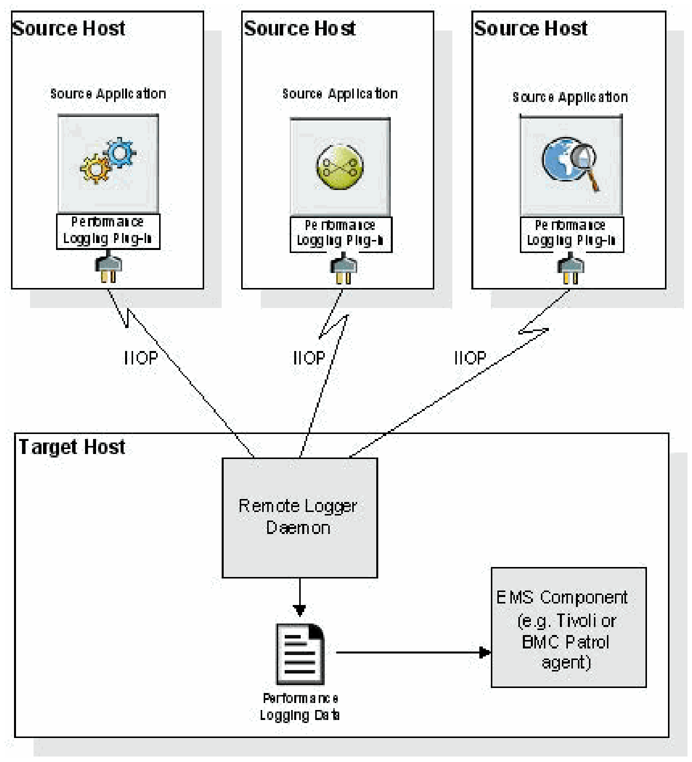


Figure 13: *Remote Logging Framework*

Deploying a remote logger daemon

As explained in "Components of a remote logging framework" on page 41, the remote logger daemon loads the remote log receiver servant, which accepts the performance logging data from the source application(s), and logs this data to a local file on the target host. You may deploy the remote logger plug-in in any Orbix application. The remote logger plug-in should be deployed in a standalone container whose sole purpose is to log data from one or more source applications. The local file on the remote host can then be consumed by the EMS agent running on that host, or used as part of some custom-made solution.

Points to note

The following points should be noted:

- IIOP is used for the data communication between the collector and the remote logger daemon. This adds very low overhead to the logging payload, because it uses a binary protocol on the wire (CDR).
- To secure the message transfer, IIOP/TLS can be used for data communication between the collector and the remote logger daemon.
- The timestamps embedded in the remote logging data are localized to the specific source system on which the monitored application is running. You must ensure that the system clocks on all participating systems are synchronized to an acceptable level, as governed by your EMS or your custom-made solution.

Configuring Remote Performance Logging

Overview

This section explains how to configure remote logging, which enables you to send logging data to a remote endpoint on another host rather than to a local file.

Configuring the remote logger daemon

To configure the remote logger daemon that runs on the remote target host, add the following configuration scope and settings to your Orbix configuration domain:

```
…
remote_logger_daemon
{
    orb_plugins = ["local_log_stream", "remote_log_receiver"];
    event_log:filters = ["IT_MGMT_LOGGING=*"];

    plugins:remote_log_receiver:log_filename =
       "/var/logs/remote_perflogs.txt";
    plugins:remote_log_receiver:ior_filename =
       "/var/publish/logger_ref.txt";
    plugins:remote_log_receiver:iiop:addr_list = ["host:port"];
    plugins:remote_log_receiver:prerequisite_plugins =
       ["iiop_profile", "giop", "iiop"];
};
…
```

Note: You can add this configuration scope directly to your configuration file, or create a separate configuration file that includes your existing configuration file.

Remote logging configuration settings

The settings for the `remote_log_receiver` plug-in are explained as follows:

`plugins:remote_log_receiver:log_filename`	This is the local file on the remote host to which all logs are directed.
`plugins:remote_log_receiver:ior_filename`	When the remote logger daemon is started, it writes a stringified Interoperable Object Reference (IOR) to the file specified by this configuration item. This IOR may be subsequently made available to the source applications that are acting as clients of the remote logger. However, this is not required if the source applications use a corbaloc URL rather than an IOR to contact the remote logger.
`plugins:remote_log_receiver:iiop:addr_list`	This specifies the hostname or IP address of the host on which the remote logger is running, and the port that it uses to listen for logging requests.
`plugins:remote_log_receiver:prerequisite_plugins`	This must specify the IIOP plug-ins that the remote logger needs for communication with the source host(s).

TLS security

If you are using TLS security:

- Ensure that you replace the `plugins:remote_log_receiver:iiop:addr_list` configuration item with `plugins:remote_log_receiver:iiop_tls:addr_list`.
- Ensure that the `plugins:remote_log_receiver:prerequisite_plugins` configuration item lists `iiop_tls` rather than `iiop`.

Configuring a deployed application on the source host

You must also configure your deployed application to use performance logging with the remote logger capability. For the purposes of illustration, it describes the steps that are required to configure an Orbix Mainframe application.

Configuration steps

To enable a deployed application (for example, on z/OS) to use performance logging with the remote logger capability:

1. Ensure that the remote logger daemon has been configured correctly and deployed on the target host, as described in "Configuring the remote logger daemon" on page 44.
2. Open the configuration domain for your deployed application. By default, this is `orbixhlq.DOMAINS(FILEDOMA)` for Orbix Mainframe applications.
3. Go to the appropriate configuration scope for your application.
4. Add `it_response_time_logger` to the end of the ORB plug-ins list setting. Also, ensure that IIOP is enabled for the application, for example:

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

Note: Ensure that you have a management license available.

5. Add `it_response_time_logger` to the server binding list for the application. For example:

```
binding:server_binding_list =
    ["SOAP+it_response_time_logger",
     "it_response_time_logger"];
```

6. Add the following collector plug-in configuration variables:

```
# update the log every 30 seconds
plugins:it_response_time_collector:period = "30";

# the id of the server for the log output
plugins:it_response_time_collector:server-id = "server-id";

# the remote endpoint details:
plugins:it_response_time_collector:remote_logging_enabled =
   "true";
initial_references:IT_PerfLoggingReceiver:reference =
   "corbaloc:iiop:1.2@remote_host:1234/IT_PerfLoggingReceiver ";
```

Note: Ensure that the *`server-id`* value is replaced with the actual server ID for the log output (for example, `cics-server-adapter-1`).

Example output

The following is example output from the performance log on the remote file system where a number of different operations have been run against the application:

```
2006-10-18 10:08:22 server=cics-server-adapter-1 status=starting_up
2006-10-18 10:08:22 server=cics-server-adapter-1 status=running
2006-10-18 10:08:52 server=cics-server-adapter-1 status=running
2006-10-18 10:09:22 server=cics-server-adapter-1 status=running
2006-10-18 10:09:22 server=cics-server-adapter-1 [ operation=test_bounded ] count=1 avg=110
   max=110 min=110
int=30001 oph=119
2006-10-18 10:09:22 server=cics-server-adapter-1 [ operation=test_unbounded ] count=1 avg=809
   max=809 min=809
int=30001 oph=119
2006-10-18 10:09:52 server=cics-server-adapter-1 status=running
2006-10-18 10:09:52 server=cics-server-adapter-1 [ operation=call_me ] count=1 avg=793 max=793
   min=793
int=29998 oph=120
2006-10-18 10:10:22 server=cics-server-adapter-1 status=running
2006-10-18 10:10:22 server=cics-server-adapter-1 [ operation=_get_currentMappings ] count=1 avg=0
   max=0 min=0
int=30000 oph=120
2006-10-18 10:10:52 server=cics-server-adapter-1 status=running
2006-10-18 10:11:22 server=cics-server-adapter-1 status=running
2006-10-18 10:11:52 server=cics-server-adapter-1 status=running
2006-10-18 10:12:22 server=cics-server-adapter-1 status=running
```

```
2006-10-18 10:12:22 server=cics-server-adapter-1 [ operation=resolve ] count=1 avg=0 max=0 min=0
   int=29999 oph=120
2006-10-18 10:12:52 server=cics-server-adapter-1 status=running
2006-10-18 10:12:57 server=cics-server-adapter-1 status=shutdown_started
2006-10-18 10:12:57 server=cics-server-adapter-1 status=shutdown_complete
```

Part 2

Programmer's Guide

In this part

This part contains the following chapters:

CHAPTER 4

Introduction to Application Management

This chapter gives an overview of Orbix enterprise application management. It introduces the Orbix management tools, Oracle's Java Management Extensions API, and Micro Focus's Orbix Management API. It also provides an overview of management programming tasks.

In this chapter

This chapter contains the following sections:

Introduction to the Orbix Management Tools

Overview

The Orbix management tools enable administrators to configure, monitor and control distributed applications at runtime. Orbix provides seamless management across the Orbix product , or any applications developed using those products, across different platform and programming language environments. The Orbix management tools include the following main components:

- "Administrator Web Console".
- "Orbix Management Service".
- "Orbix Configuration Explorer".
- "Orbix Configuration Authority".

Administrator Web Console

The *Administrator Web Console* provides a web browser interface to the Orbix management tools. It enables you to manage applications and application events from anywhere, without the need for download or installation. It communicates with the management service using HTTP (Hypertext Transfer Protocol), as illustrated in Figure 14.

Orbix Management Service

The *Orbix Management Service* is the central point of contact for accessing management information in a *domain*. A domain is an abstract group of managed server processes within a physical location. The management service is accessed by both the Administrator Web Console and the Orbix Configuration Explorer.

Note: Managed z/OS applications can be written in C++. CORBA C++ applications use the management service process, `iona_services.management`.

Orbix Configuration Explorer

The *Orbix Configuration Explorer* is a Java graphical user interface (GUI) that enables you to manage your configuration settings. It communicates with your Configuration Repository (CFR) or configuration file, using IIOP (Internet Inter-ORB Protocol).

Figure 14 shows how the Orbix management tools interact with managed applications to provide management capabilities.

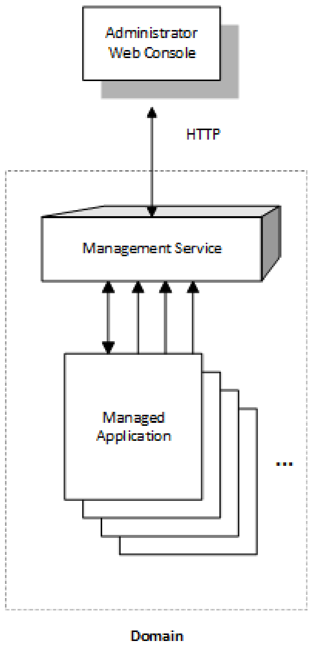


Figure 14: *Management Overview*

Orbix Configuration Authority

The *Orbix Configuration Authority* provides a web browser interface to descriptive information about all Orbix configuration settings. You can browse and search for information about Orbix configuration variables in your CFR or configuration file.

Further information

For detailed information about using the Orbix management tools, see the *Management User's Guide*.

Introduction to Java Management Extensions

Overview

Java Management Extensions (JMX) is a standards-based API from Oracle that provides a framework for adding enterprise management capabilities to user applications. This section explains the main JMX concepts and shows how JMX and Orbix interact to provide enterprise management for Java applications. This includes both J2EE and CORBA Java servers.

This section includes the following:

- "MBeans".
- "The MBean server".
- "Management instrumentation".
- "Standard and Dynamic MBeans".
- "Further information".

MBeans

The concept of an *MBean* (a managed bean) is central to JMX. An MBean is simply an object with associated attributes and operations. It acts as a handle to your application object, and enables the object to be managed.

For example, a `Car` MBean object, with an associated `speed` attribute, and `start()` and `stop()` operations, is used to represent a car application object, with corresponding attributes and operations. Application developers can express their application objects as a series of related MBeans. This enables administrators to manage these application objects using an administration console (for example, the Orbix management tools).

The MBean server

All the MBeans created by developers are managed and controlled by a MBean server, which is provided by JMX. All MBeans that are created must be registered with an MBean server so that they can be accessed by management applications, such as Orbix.

Figure 15 shows a Java example of the JMX components at work. It shows how these components interact with Orbix to provide management capability for your application.

For simplicity, this diagram only shows one MBean. An application might have multiple MBeans representing the application objects that you wish to manage. In addition, new instrumentation code is not solely confined to the MBean. You will need to add some new code to your sever implementation (for example, to enable your server to contact the management service).

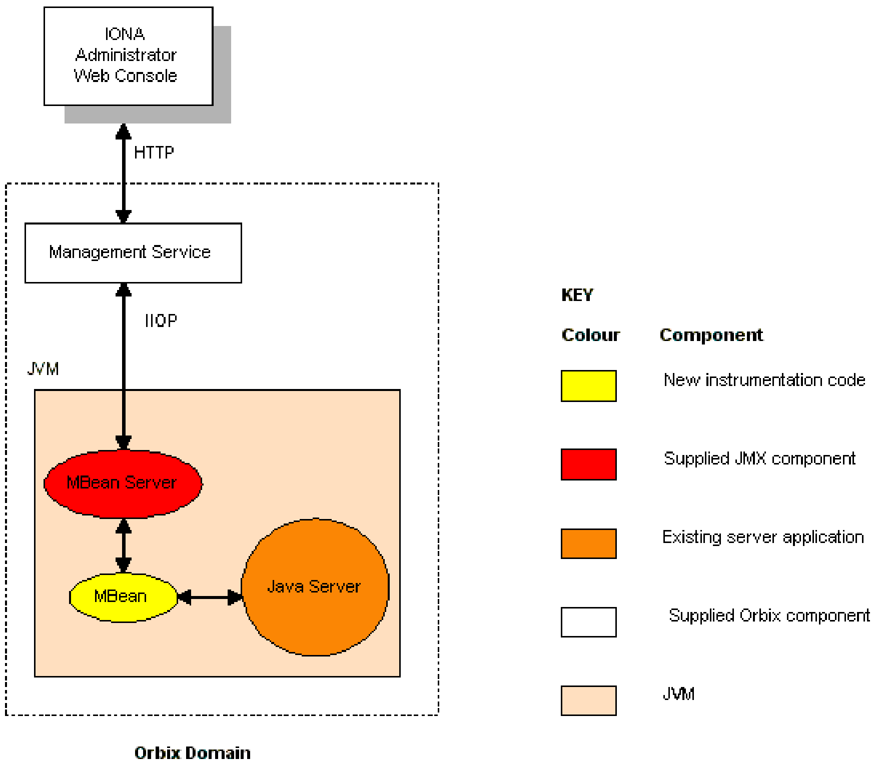


Figure 15: *JMX Management and Orbix*

Management instrumentation

Adding JMX management code to your application is also known as adding management *instrumentation* or *instrumenting* your existing application. These standard management terms are used throughout this book.

Figure 15 shows the new management instrumentation code as an MBean. MBeans must be added to your application to enable it for management.

Standard and Dynamic MBeans

The MBeans discussed so far in this chapter are referred to as *standard MBeans*. These are ideally suited to straightforward management scenarios where the structure of managed data is well defined and unlikely to change often. JMX specifies another category of MBeans called *dynamic MBeans*. These are designed for when the structure of the managed data is likely to change regularly during the lifetime of the application.

Implementing dynamic MBeans is more complex than for standard MBeans. If your management solution needs to provide integration with existing and future management protocols and platforms, using dynamic MBeans could make it more difficult to achieve this goal. The examples cited in this book use standard MBeans only.

Further information

For more information about JMX, see Oracle's JMX Instrumentation and Agent Specification, and Reference Implementation Javadoc. These documents are available online at:

https://www.oracle.com/java/technologies/javase/javamanagement.html

For information on how to integrate the Orbix management tools with other general purpose management applications (for example, HP Openview^{TM} or CA UniCenter^{TM}), see the "SNMP Integration" chapter in the *Management User's Guide*.

Introduction to the Orbix Management API

Overview

JMX does not specify how MBeans communicate at the network protocol level. Micro Focus's Orbix Management API is used to enable network communications for MBeans. This API also enables you to specify relationships between MBeans, and display MBeans in the Orbix management tools. This section includes the following:

- "The IIOP Adaptor".
- "Defining MBean relationships".
- "C++ Instrumentation".

The IIOP Adaptor

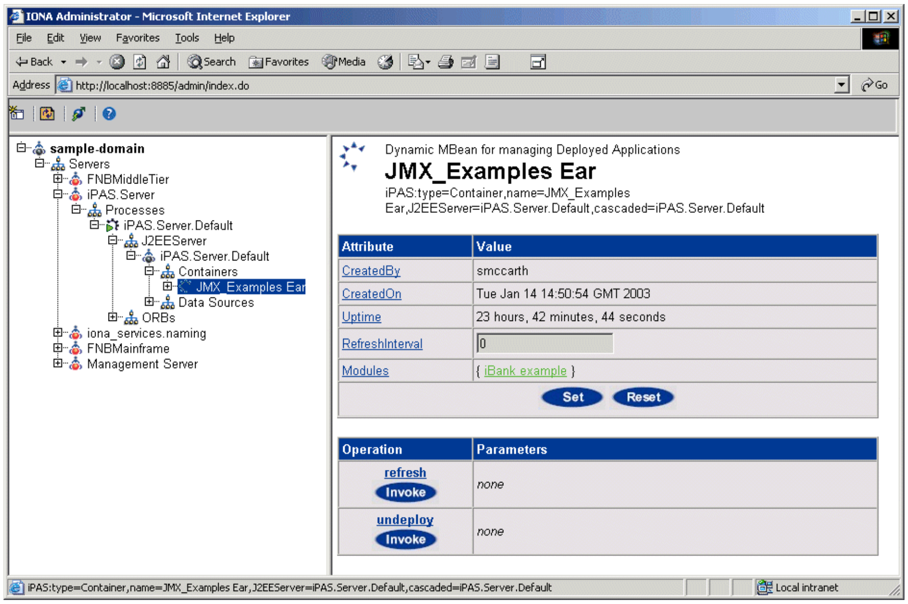
The Orbix Management API enables network communication between the MBean server and the management service over IIOP (Internet Inter-ORB Protocol). This is performed using an IIOP adapter, which is contained in the ORB plugin for the management service.

Figure 15 shows a J2EE example of this IIOP communication. This cross-platform API also enables communication for CORBA Java and C++ servers.

Defining MBean relationships

The Orbix Management API also enables you to specify hierarchical parent–child relationships between MBeans. For example, you might want to show relationships between your application server and its lower-level processes. These relationships can then be displayed in the Administrator Web Console.

Figure 16 shows example parent–child relationships displayed in the left pane of the Administrator Web Console.



Dynamic MBean for managing Deployed Applications

JMX_Examples Ear

iPAS:type=Container,name=JMX_Examples Ear,J2EEServer=iPAS.Server.Default,cascaded=iPAS.Server.Default

Attribute	Value
CreatedBy	smccarth
CreatedOn	Tue Jan 14 14:50:54 GMT 2003
Uptime	23 hours, 42 minutes, 44 seconds
RefreshInterval	0
Modules	{ iBank example }

Operation	Parameters
refresh	*none*
undeploy	*none*

iPAS:type=Container,name=JMX_Examples Ear,J2EEServer=iPAS.Server.Default,cascaded=iPAS.Server.Default

Figure 16: *Example Parent–Child Relationship*

C++ Instrumentation

The concept of an MBean is a Java term that comes from JMX. The C++ version of the Orbix Management API uses the generic concept of a *Managed Entity* instead of an MBean. A C++ Managed Entity is functionally equivalent to the Java MBean. It acts as a handle to your application object, and enables the object to be managed.

The C++ version of the Orbix Management API is defined in IDL (Interface Definition Language).

For more details of the Orbix Management API, see the *Orbix Management IDLdoc.*

Overview of Management Programming Tasks

Overview

This section gives an overview of the typical management programming tasks. These include the following:

- "Identifying tasks to be managed".
- "Writing your MBeans".
- "Registering your MBeans with the MBean server".
- "Unregistering your MBeans".
- "Defining relationships between MBeans".

These tasks are explained in more detail in "Instrumenting CORBA C++ Applications" on page 63.

Identifying tasks to be managed

Before adding any management code to an application, you must decide on the application tasks that you wish the administrator to manage.

Deciding which tasks should be managed varies from application to application. This depends on the nature of the application, and on the type of runtime administration that is required. Typical managed tasks include monitoring the status of an application (for example, whether it is active or inactive), and controlling its operation (for example, starting or stopping the application).

Writing your MBeans

When you have decided which parts of your application need to be managed, you can define and implement MBeans to satisfy your management objectives. Each MBean object must implement an interface ending with the term `MBean` (for example, `CarMBean`).

To expose its attributes, an MBean interface must declare a number of get and set operations. If get operations are declared only, the MBean attributes are read-only. If set operations are declared, the MBean attributes are writable.

Registering your MBeans with the MBean server

Registering application MBeans with the MBean server enables them to be monitored and controlled by the Orbix management tools. Choosing when to register or expose your MBeans varies from application to application. However, there are two stages when all applications create and register MBeans:

During application initialization. During any application initialization sequence, a set of objects is created that represents the core functionality of the application. After these objects are created, MBeans should also be created and registered, to enable basic management of that application.

During normal application runtime. During normal application runtime, new objects are created as a result of internal or external events (for example, an internal timer, or a request from a client). When new objects are created, corresponding MBeans can be created and registered, to enable management of these new application components. For example, in a bank example when a new account is created, a new account MBean would be also be created and registered with the MBean server.

Unregistering your MBeans

You might wish to unregister an MBean in response to an administrator's interaction with the system. For example, if a bank teller session is closed, it would be appropriate to unregister a corresponding session MBean. This ensures that the MBean will no longer be displayed as part of the application that is being managed.

Defining relationships between MBeans

You can use the Orbix Management API to define parent–child relationships between MBeans. These relationships are then displayed in the Administrator Web Console, as shown in Figure 16 on page 58.

Parent-child relationships are no longer displayed in the console when the MBean is unregistered by the application (for example, if a bank account is closed).

Instrumentation demonstration

An instrumentation demonstration is provided in the UNIX System Services component of your Orbix Mainframe installation, as follows (where *install_dir* represents the full path to your Orbix Mainframe installation on UNIX System Services):

```
install_dir/asp/Version/demos/corba/pdk/instrumented_plugin
```

This instrumentation demonstration illustrates how to use the main Management APIs and how to write your own Generic Service application. You can use an ORB plug-in approach to build the Management code, to instrument existing services such as the CICS and IMS server adapters.

CHAPTER 5

Instrumenting CORBA C++ Applications

This chapter explains how to use the Orbix C++ Management API to enable an existing CORBA C++ application for management. It uses the CORBA instrumented_plugin demo as an example.

In this chapter

This chapter contains the following sections:

Step 1—Identifying Tasks to be Managed

Overview

Before adding management code to an application, you must decide on the tasks in your application that you wish to be managed by a system administrator. Only then should you start thinking about adding management instrumentation code to your existing application. This section includes the following:

- "Existing functionality".
- "New management tasks".
- "Planning your programming steps".
- "Location of the management code".

Existing functionality

The `instrumented_plugin` example adds management capability to an existing CORBA C++ application. This is a simple "Hello World" application, where the client application reads the server's object reference from a file.

For details of how to run the instrumented plugin application, see the `README_CXX.txt` file in the following Orbix directory:

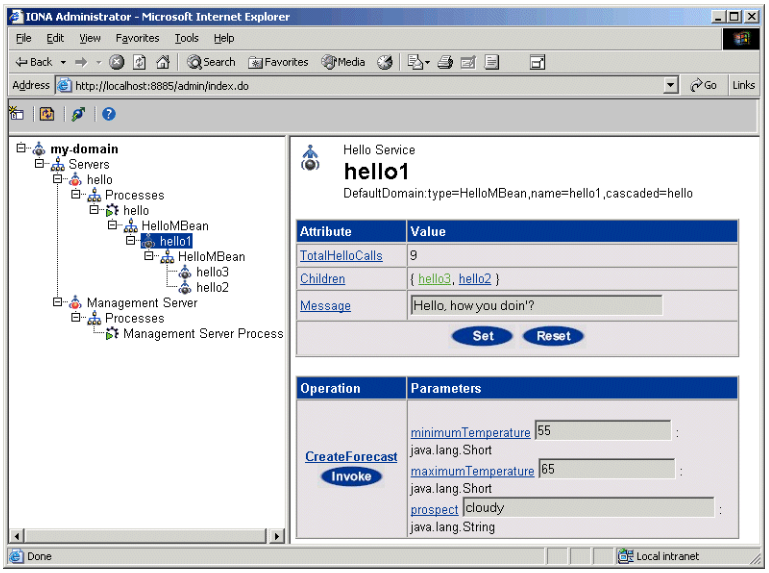
```
<install-dir>/asp/Version/demos/corba/pdk/instrumented_plugin
```

New management tasks

The new management instrumentation code added to `instrumented_plugin` application enables administrators to perform the following additional tasks:

- Monitor the status of the `Hello` server (active or inactive).
- Monitor the number of times that the client reads the server's object reference.
- Set a hello text message.
- Invoke a weather forecast with specified text values.
- Shutdown the `Hello` server.

Administrators can perform these tasks using the Administrator Web Console, shown in Figure 17.


IONA Administrator - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8885/admin/index.do
Go Links
my-domain
Servers
hello
Processes
hello
HelloMBean
hello1
HelloMBean
hello3
hello2
Management Server
Processes
Management Server Process
Set
Reset
Invoke
Done
Local intranet

Hello Service

hello1

DefaultDomain:type=HelloMBean,name=hello1,cascaded=hello

Attribute	Value
TotalHelloCalls	9
Children	{ hello3, hello2 }
Message	Hello, how you doin'?

Operation	Parameters
CreateForecast	minimumTemperature 55 : java.lang.Short maximumTemperature 65 : java.lang.Short prospect cloudy : java.lang.String

Figure 17: *Instrumented Plugin in Administrator Web Console*

Planning your programming steps

When you have identified your management tasks, you should think carefully about how exactly you wish to add the new management code to your existing application. For example, how much of the new code you will add to existing files, and how much will be in new files.

In the `instrumented_plugin` example, the instrumentation code is part of the service and is initialized when the service is initialized. For larger applications, you might wish to keep new instrumentation files in a separate directory.

This chapter explains how Orbix C++ management code was added to the instrumented_plugin application, and shows the standard programming steps. For example, defining and implementing your MBeans, and defining relationships between MBeans.

Note: When instrumenting CORBA C++ servers, you do not need to make any changes to the CORBA IDL. You can enable your application for management simply by adding new MBean instrumentation code to your CORBA C++ implementation files.

Location of the management code

You should first decide where you wish to store your new management code. All source code for the instrumented_plugin application is stored in the following directory:

<install-dir>/asp/*Version*/demos/corba/pdk/instrumented_plugin/

The management code for the CORBA C++ server is stored in the following directory:

.../instrumented_plugin/cxx_server

The following files are discussed in detail in this chapter

- hello_mbean.h
- hello_mbean.cxx
- hello_world_impl.cxx

For larger applications, it is advised that you to store your management code in a separate management directory. This will make your application more modular, and easier to understand.

Instrumented plugin overview

Figure 18 shows the main components of the instrumented_plugin application. In this simple example, there is only one C++ MBean, the HelloBean.

Most of the key management programming tasks in this example are performed in the HelloWorld server implementation (hello_world_impl.cxx). For example, management initialization, creating the MBean, and displaying MBeans in the navigation tree of the console. The server implementation interacts with the MBean implementation to perform these tasks.

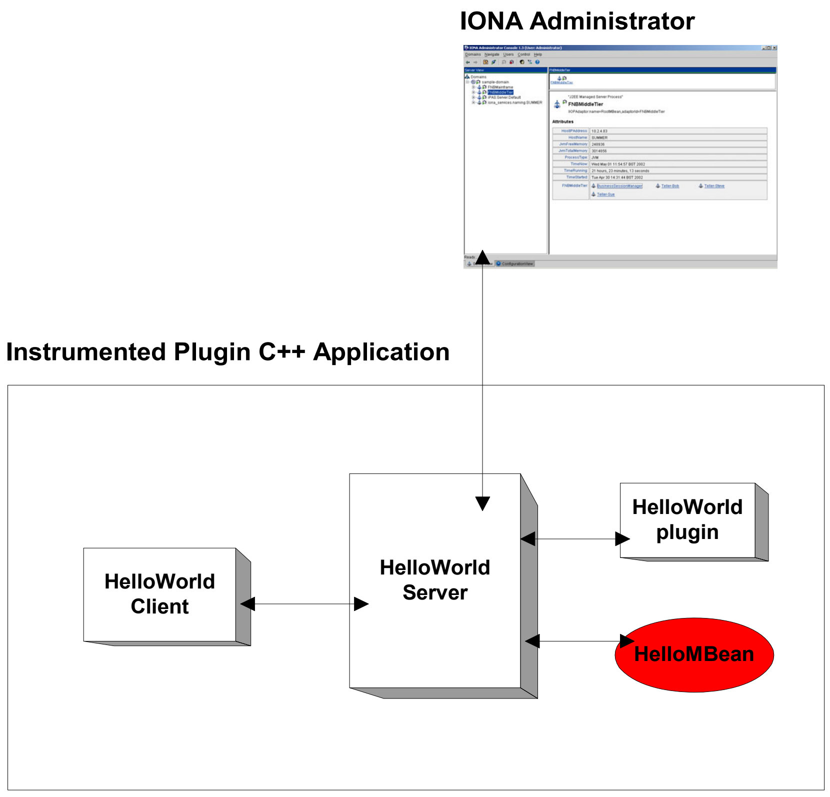


Figure 18: *Instrumented Plugin Application Overview*

Step 2—Defining your MBeans

Overview

When you have planned which parts of your application need to be managed, you can then define MBeans to satisfy your management objectives. This section shows how to define an example MBean header file for the `instrumented_plugin` application. This section includes the following:

- "Managed Entities and MBeans".
- "Rules for MBean declarations".
- "Example MBean declaration".
- "Example private description".
- "Further information".

Managed Entities and MBeans

The C++ version of the Orbix Management API is based around the concept of a *Managed Entity*. This is similar to the JMX MBeans that are used by Java Programmers. A managed entity acts as a handle to your application object, and enables the object to be managed. The terms managed entity and MBean are used interchangeably in this document.

The Orbix C++ Management API is defined in CORBA IDL (Interface Definition Language). For full details of the Orbix Management API, see the *Orbix Management IDLdoc*.

Rules for MBean declarations

The following rules apply for C++ MBeans:

- Each MBean object must implement the declaration defined for it in a C++ header file (in this example, `hello_mbean.h`).
- The following two operations must be declared and implemented:
    - `get_mgmt_attribute()`
    - `set_mgmt_attribute()`

  (although their implementation may be empty). These are the only two operations for getting and setting all MBean attributes. The name of the attribute is passed as a parameter, and the operation determines whether to get or set the attribute.

- The `invoke_method()` operation must also be declared and implemented (although its implementation may be empty).

You must declare all these methods in the MBean header file, and then implement them in the corresponding MBean implementation file (in this example, `hello_mbean.cxx`).

Example MBean declaration

The header file for the `instrumented_plugin` application is `hello_mbean.h`. It includes the following Hello MBean declaration:

Example 1: *Hello MBean Declaration*

```
    #ifndef _HELLO_MBEAN_H_
    #define _HELLO_MBEAN_H_

    #include <omg/orb.hh>
    #include <orbix_pdk/instrumentation.hh>
    #include <orbix/corba.hh>
    #include <it_dsa/string.h>
    #include <it_dsa/list.h>
    #include <it_ts/mutex.h>

    class HelloWorldImpl;

    class HelloMBean :
1       public virtual IT_Mgmt::ManagedEntity,
        public virtual IT_CORBA::RefCountedLocalObject {

     public:

        HelloMBean (
            HelloWorldImpl * orb_info,
            const char * name
        );

        virtual ~HelloMBean();

2       IT_Mgmt::ManagedEntityIdentifier managed_entity_id()
            IT_THROW_DECL((CORBA::SystemException));

3      char* entity_type() IT_THROW_DECL((CORBA::SystemException));
```

Example 1: *Hello MBean Declaration*

```
4      CORBA::Any* get_mgmt_attribute(const char*  key)
            IT_THROW_DECL((CORBA::SystemException,
            IT_Mgmt::AttributeUnknown));

        void set_mgmt_attribute(
           const char*  key, const CORBA::Any &  new_value)
            IT_THROW_DECL((CORBA::SystemException,
            IT_Mgmt::AttributeUnknown, IT_Mgmt::AttributeReadOnly,
            IT_Mgmt::AttributeValueInvalid));

        CORBA::Any* invoke_method (const char*  method_name,
          const IT_Mgmt::ArgumentSeq&  in_parameters,
            IT_Mgmt::ArgumentSeq_out out_parameters)
            IT_THROW_DECL((CORBA::SystemException,
            IT_Mgmt::MethodUnknown, IT_Mgmt::InvocationFailed ));

5      IT_Mgmt::ManagedEntityDescription get_description()
          IT_THROW_DECL((CORBA::SystemException));

       struct HelloParam
        {
            const char *name;
            const char *type;
            const char *description;
        };

       typedef IT_List<HelloParam> HelloParamList;
 .
 .
 .
```

This hello_mbean.h code example is described as follows:

1. The HelloMBean class implements the IT_Mgmt::ManagedEntity IDL interface. All entities that need to be managed must derive from this interface. The C++ implementation of the IT_Mgmt::ManagedEntity IDL interface is equivalent to a Java MBean.
2. The IT_Mgmt::ManagedEntityIdentifier managed_entity_id() operation is used to uniquely identify the managed entity.
3. The entity_type() operation returns a string indicating the type. This demo uses HelloMBean, which is the C++ classname. The naming service, for example, uses NamingMBean.

4. The `get_mgmt_attribute()`, `set_mgmt_attribute()`, and `invoke_method()` operations all use the `CORBA::Any` type to access managed entity attributes and operations.

   The `CORBA::Any` type enables you to specify values that can express any IDL type. For detailed information about the `CORBA::Any` type, see the *CORBA Programmer's Guide* (C++ version).

5. The `get_description()` operation returns an XML description of the managed entity. This is used to display information about the managed entity in the Administrator Web Console. This is described in more detail in the next topic.

Example private description

The `hello_mbean.h` file also includes the following privately declared information:

Example 2: *HelloMBean Private Declaration*

```
private:

1   struct HelloAttribute
      {
          const char * name;
          const char * type;
          const char * description;
          IT_Bool      access;
      };
      typedef IT_List<HelloAttribute> HelloAttributeList;

    struct HelloOperation
      {
          const char * name;
          const char * return_type;
          const char * description;
          HelloParamList params;
      };

      typedef IT_List<HelloOperation> HelloOperationList;

      void initialize_attributes();

      void initialize_operations();

      IT_String get_attributes_XML() const;
```

Example 2: *HelloMBean Private Declaration*

```
    IT_String get_attribute_XML(HelloAttribute att) const;

    IT_String get_operations_XML() const;

    IT_String get_operation_XML(HelloOperation op) const;

    IT_String get_param_XML(HelloParam param) const;

2   IT_Bool validate_create_forecast_parameters(
        const IT_Mgmt::ArgumentSeq&  in_parameters)
        throw (IT_Mgmt::InvocationFailed);

    void throw_wrong_num_parameters()
        throw (IT_Mgmt::InvocationFailed);

    void throw_invalid_parameter(const char *param_name)
        throw (IT_Mgmt::InvocationFailed);

    void throw_bad_temp_range( const char *paramName,
        CORBA::Short minVal, CORBA::Short maxVal)
        throw (IT_Mgmt::InvocationFailed);

    void throw_max_must_be_greater_than_min()
         throw (IT_Mgmt::InvocationFailed);

    HelloAttributeList      m_attribute_list;
    HelloOperationList      m_operation_list;
    IT_String               m_identity;
    IT_String               m_domain;
    IT_String               m_class_name;
    IT_String               m_type;
    IT_String               m_name;
    IT_Mutex                m_mutex;

    // Attribute names
    const char*             m_hit_count_name;
    const char*             m_children_name;
    const char*             m_message_name;

    // Operation names
    const char*             m_create_forecast_name;

    HelloWorldImpl*         m_hello;
};
```

1. This privately declared information is used to display descriptions of managed attributes and operations in the Administrator Web Console. For example, the `initialize_attributes()` function uses a `HelloAttribute` structure to define a single attribute. An instance of this attribute and anything else that you declare are pushed on to a a list. This list is then processed by `get_attributes_XML()` and by `get_attribute_XML()` to generate the description for display in the Administrator Web Console.
2. These operations all throw `IT_Mgmt` management exceptions. You also can specify custom management exceptions. For more information, see "Throw the managed exceptions" on page 82.

Further information

C++ Managed entities are similar to the JMX MBeans that are used by Java Programmers. For information about Java MBeans see:

https://docs.oracle.com/javase/tutorial/jmx/mbeans/index.html

Step 3—Implementing your MBeans

Overview

After defining your MBean interfaces, you must provide an MBean implementation. MBean implementation objects interact with the application they are designed to manage, enabling monitoring and control.

For example, this section shows the interaction between an MBean (`HelloMBean`) and the CORBA server implementation object (`HelloWorldImpl`). This section shows example code extracts from the MBean implementation file (`hello_mbean.cxx`). It includes the following steps:

1. "Write the MBean constructor and destructor".
2. "Get the managed entity ID and entity type".
3. "Get the managed attributes".
4. "Set the managed attributes"
5. "Invoke the managed operations".
6. "Throw the managed exceptions".
7. "Get the MBean description".

Write the MBean constructor and destructor

The `HelloMBean` constructor and destructor are shown in the following extract from `hello_mbean.cxx`:

Example 3: *MBean Constructor and Destructor*

1
```
HelloMBean::HelloMBean (
     HelloWorldImpl * hello, const char *name) : m_hello(0)
{
    assert(hello != 0);
    hello->_add_ref();
    m_hello = hello;
    m_domain = m_hello->get_domain_name();
    m_class_name = "com.iona.hello.HelloMBean";
    m_type = "HelloMBean";
    m_name = "HelloService";
```

Example 3: *MBean Constructor and Destructor*

```
    m_identity = "DefaultDomain";
    //m_identity = m_domain.c_str();
    m_identity += ":type=HelloMBean,name=";
    m_identity += name;
    initialize_attributes();
    initialize_operations();
}
```
2
```
HelloMBean::~HelloMBean()
{
    m_hello->_remove_ref();
}
```

This code extract is explained as follows:

1. The `HelloMBean()` constructor specifies all the key information used to identify the MBean, and display it in the Administrator Web Console. For example, this includes its domain name, a Java-style class name (`com.iona.hello.HelloMBean`), and a managed entity ID. For information about registering MBeans as managed entities, see "Creating an example MBean" on page 90.
2. The `HelloMBean()` destructor. For information about unregistering MBeans as managed entities, see "Removing your MBeans" on page 91.

Get the managed entity ID and entity type

The managed entity ID and type uniquely identify the managed entity. The following code extract shows how to obtain the managed entity ID and its type:

Example 4: *Managed Entity ID and Type*

1
```
IT_Mgmt::ManagedEntityIdentifier HelloMBean::managed_entity_id()
   IT_THROW_DECL((CORBA::SystemException))
{
   return CORBA::string_dup(m_identity.c_str());
}
```
2
```
char* HelloMBean::entity_type()
    IT_THROW_DECL((CORBA::SystemException))
{
    return CORBA::string_dup(m_type.c_str());
}
```

This code extract is explained as follows:

1. The ID returned by `managed_entity_id()` is a string that includes the domain, type, and name, at minimum. These are the keys that are looked up in the MBean by the management service. The actual values are decided by the developer.

   This example uses the `DefaultDomain` for the first string (the domain). You can specify your own domain name instead. The rest of the name value pairs follow, and are separated by commas, for example:

   `"DefaultDomain:type=HelloMBean,name=HelloService"`

   Note: The domain name part of the managed entity ID is not related to an Orbix configuration or location domain. It is a namespace for managed entities only. For example, in a banking application your IDs might use a `BankingApp` domain.

2. The `entity_type()` operation returns a string indicating the type of the managed entity. The entity type is formatted in a dotted Java-style notation, which can be used by the Administrator Web Console to display icons for an MBean. For example, this demo uses the `com.iona.hello.HelloMBean` type.

Get the managed attributes

The following code extract shows how to get managed MBean attributes:

Example 5: *Getting Managed Attributes*

```
1 CORBA::Any* HelloMBean::get_mgmt_attribute(const char*  key)
      IT_THROW_DECL((CORBA::SystemException,
      IT_Mgmt::AttributeUnknown))
      {
2       CORBA::Any_var retval = new CORBA::Any;
        if (strcmp(key, m_hit_count_name) == 0)
        {
          IT_Locker<IT_Mutex> lock(m_mutex);
          *retval <<= m_hello->total_hits();
          return retval._retn();
        }
3       else if (strcmp(key, m_children_name) == 0)
        {
          IT_Locker<IT_Mutex> lock(m_mutex);
          HelloWorldImpl::HelloWorldList children =
          m_hello->get_children();
```

Example 5: *Getting Managed Attributes*

```
        CORBA::AnySeq children_seq(children.size());
        children_seq.length(children.size());
        HelloWorldImpl::HelloWorldList::iterator iter =
        children.begin();

        for (int i = 0; i < children.size();i++, iter++)
        {
          IT_Mgmt::ManagedEntity_var mbean = (*iter)->get_mbean();
          children_seq[i] <<= mbean.in();
        }
       *retval <<= children_seq;
       return retval._retn();
    }

    else if (strcmp(key, m_message_name) == 0)
    {
        IT_Locker<IT_Mutex> lock(m_mutex);
        CORBA::String_var message = m_hello->get_message();
        *retval <<= message.in();
        return retval._retn();
    }
    else
    {
    throw new IT_Mgmt::AttributeUnknown();
    }
}
```

This code extract is explained as follows:

1. The `get_mgmt_attribute()` operation is the only operation used for getting all MBean attributes. The name of the attribute is passed in and the operation determines whether to get the attribute.
2. The `CORBA::Any` type enables you to specify values that can express any IDL type. For details of managed attribute types, see "Permitted types" on page 78. For detailed information about the `CORBA::Any` type, see the *CORBA Programmer's Guide, C++*.
3. This `get_mgmt_attribute()` implementation supports complex attribute types by also getting the attributes of child MBeans.

   In the `instrumented_plugin` example, the children attribute of the Hello MBean gets a list of references to child MBeans.

For example, in Figure 17 on page 65, the **Children** attribute and its child MBeans (**hello3** and **hello2**) are displayed in the Administrator Web Console.

Permitted types The following basic types are permitted for managed attributes:

```
CORBA::Short
CORBA::Long
CORBA::LongLong
CORBA::Float
CORBA::Double
CORBA::Boolean
CORBA::Octet
CORBA::String,
CORBA::WString.
```

In addition, you can use `ManagedEntity` references to connect one Managed Entity and another. These will be displayed as hyperlinks on the web console. Finally, you can use `CORBA::AnySeq` to create lists of any of the permitted types already listed.

Set the managed attributes

The following code extract shows how to set managed MBean attributes:

Example 6: *Setting Managed Attributes*

1
```
void HelloMBean::set_mgmt_attribute(const char*  key,
    const CORBA::Any &  new_value
   IT_THROW_DECL((CORBA::SystemException,
    IT_Mgmt::AttributeUnknown, IT_Mgmt::AttributeReadOnly,
   IT_Mgmt::AttributeValueInvalid ))
  {
    if (strcmp(key, m_message_name) == 0)
    {
        CORBA::TypeCode_var tc(new_value.type());
        CORBA::TCKind kind = tc->kind();

        if (kind != CORBA::tk_string)
        {
         throw new IT_Mgmt::AttributeValueInvalid();
        }
        const char *new_message;
        new_value >>= new_message;
```

Example 6: *Setting Managed Attributes*

2
```
            m_hello->set_message(new_message);
        }
        else if (strcmp(key, m_hit_count_name) == 0)
        {
            throw new IT_Mgmt::AttributeReadOnly();
        }
        else if (strcmp(key, m_children_name) == 0)
        {
            throw new IT_Mgmt::AttributeReadOnly();
        }
        else
        {
            throw new IT_Mgmt::AttributeUnknown();
        }
}
```

This code extract is explained as follows:

1. The `set_mgmt_attribute()` operation is the only operation used for setting all MBean attributes. The name of the attribute is passed in and the operation determines whether to set the attribute.

   The `CORBA::Any` type enables you to specify values that can express any IDL type. For detailed information about the `CORBA::Any` type, see the *CORBA Programmer's Guide, C++*.

2. The `set_message()` function enables you to set the text message for the hello greeting that is returned by the Hello object. For example, Figure 17 on page 65, shows an example text greeting for the **Message** attribute in the Administrator Web Console.

Invoke the managed operations

The following code extract shows how to invoke MBean operations:

Example 7: *Invoke Operations*

```
1   CORBA::Any* HelloMBean::invoke_method(const char*  method_name,
        const IT_Mgmt::ArgumentSeq&  in_parameters,
        IT_Mgmt::ArgumentSeq_out out_parameters)
        IT_THROW_DECL((CORBA::SystemException,IT_Mgmt::MethodUnknown
        IT_Mgmt::InvocationFailed))
       {
        CORBA::Any_var retval = new CORBA::Any;
        if (strcmp(method_name,m_create_forecast_name) == 0)
        {
            IT_Locker<IT_Mutex> lock(m_mutex);

            out_parameters = new IT_Mgmt::ArgumentSeq(0);
            out_parameters->length(0);

            CORBA::String_var forecast;
            CORBA::Short min_temp, max_temp;
            const char *prospect;

            if (in_parameters.length() != 3)
            {
                throw_wrong_num_parameters();
            }

2       validate_create_forecast_parameters(in_parameters);

            in_parameters[0].value >>= min_temp;
            if (min_temp < COLDEST_MIN_TEMP || min_temp >
            HOTTEST_MAX_TEMP)
            {
              throw_bad_temp_range("minimumTemperature",
              COLDEST_MIN_TEMP,HOTTEST_MAX_TEMP);
            }

            in_parameters[1].value >>= max_temp;
            if (max_temp < COLDEST_MIN_TEMP || max_temp >
             HOTTEST_MAX_TEMP)
             {
              throw_bad_temp_range("maxmimumTemperature",
              COLDEST_MIN_TEMP, HOTTEST_MAX_TEMP);
             }
```

Example 7: *Invoke Operations*

```
            in_parameters[2].value >>= prospect;
            if (max_temp < min_temp)
            {
               throw_max_must_be_greater_than_min();
            }

3           m_hello->set_forecast_parameters(
                min_temp,
                max_temp,
                prospect
            );

            forecast = m_hello->get_forecast();
           *retval <<= forecast.in();
            return retval._retn();
        }
        else
        {
           throw new IT_Mgmt::MethodUnknown();
        }
}
```

This code extract is explained as follows:

1. The `invoke_method()` operation is the only operation used for invoking all MBean operations. The name of the operation is passed in and the `invoke_method()` operation determines whether to invoke the operation.

   The `CORBA::Any` type enables you to specify values that can express any IDL type. For detailed information about the `CORBA::Any` type, see the *CORBA Programmer's Guide, C++*.

2. In this example, the `validate_create_forecast_parameters()` function checks that the weather forecast values entered are of the correct type (`short` or `string`). The rest of the code checks that the temperature values entered do not fall outside the range of the predeclared `const` values:

```
static const CORBA::Short COLDEST_MIN_TEMP = -100;
static const CORBA::Short HOTTEST_MAX_TEMP = 150;
```

3. The `set_forecast_parameters()` and `get_forecast()` functions enable you to create and invoke your own weather forecast. Figure 17 on page 65, shows example parameter values for the **CreateForecast** operation in the Administrator Web Console. This operation takes the following parameters:
   - `min_temp` (short)
   - `max_temp` (short)
   - `prospect` (string)

Throw the managed exceptions

Before throwing management exceptions, you must first declare them in your MBean implementation file, for example:

```
static const char *BAD_TEMP_RANGE_EX =
  "com.iona.demo.pdk.instrumentedplugin.BadTempRange";
static const char *MAX_MUST_BE_GREATER_THAN_MIN_EX =
 "com.iona.demo.pdk.instrumentedplugin.MaxMustBeGreaterThanMin";
static const char *INVALID_PARAM_EX_PARAM_NAME = "paramName";
static const char *BAD_TEMP_RANGE_EX_PARAM_NAME = "paramName";
static const char *BAD_TEMP_RANGE_EX_MIN_VAL = "minVal";
static const char *BAD_TEMP_RANGE_EX_MAX_VAL = "maxVal";
```

The following code shows two example functions that are used to throw management exceptions:

Example 8: *Throwing Management Exceptions*

```
void HelloMBean::throw_bad_temp_range(
    const char *paramName,
    CORBA::Short minVal,
    CORBA::Short maxVal) throw (IT_Mgmt::InvocationFailed)
{
    IT_Mgmt::InvocationFailed ex;
    IT_Mgmt::InvocationError err;
    IT_Mgmt::PropertySeq_var properties = new
        IT_Mgmt::PropertySeq(3);
    properties->length(3);
    properties[0].name = BAD_TEMP_RANGE_EX_PARAM_NAME;
    properties[0].value <<= paramName;
    properties[1].name = BAD_TEMP_RANGE_EX_MIN_VAL;
    properties[1].value <<= minVal;
    properties[2].name = BAD_TEMP_RANGE_EX_MAX_VAL;
    properties[2].value <<= maxVal;
```

Example 8: *Throwing Management Exceptions*

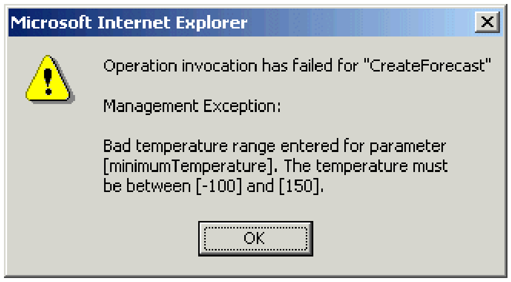
```
    err.id = (const char *) BAD_TEMP_RANGE_EX;
    err.error_params = properties;
    ex.error_details = err;

    throw IT_Mgmt::InvocationFailed(ex);
}

void HelloMBean::throw_max_must_be_greater_than_min()
   throw (IT_Mgmt::InvocationFailed)
{
    IT_Mgmt::InvocationFailed ex;
    IT_Mgmt::InvocationError err;

    err.id = (const char *) MAX_MUST_BE_GREATER_THAN_MIN_EX;
    ex.error_details = err;

    throw IT_Mgmt::InvocationFailed(ex);
}
```

Custom exception messages You can specify custom messages using the `exception-ia.properties` file, which is located in the following off-host directory:

`install-dir/conf/domains/default-domain/resources`

For example, the entry in this file for the `throw_bad_temp_range()` operation is as follows:

```
com.iona.demo.pdk.instrumentedplugin.BadTempRange=Bad
   temperature range entered for parameter %paramName%. The
   temperature must be between %minVal% and %maxVal%.
```



Operation invocation has failed for "CreateForecast"

Management Exception:

Bad temperature range entered for parameter [minimumTemperature]. The temperature must be between [-100] and [150].

Figure 19: *Instrumented Plugin Custom Exception*

Get the MBean description

The following code shows how the MBean descriptions are obtained for display in the Administrator Web Console:

Example 9: *Getting the MBean Description*

```
1   IT_Mgmt::ManagedEntityDescription HelloMBean::get_description()
       IT_THROW_DECL((CORBA::SystemException))
    {
        IT_String xml_str =
        "<?xml version=\"1.0\"?>"
        "<?rum_dtd version=\"1.0\" ?>"
        "<mbean>"
            "<class_name>";
                xml_str += m_class_name;
                xml_str +=
            "</class_name>"
            "<domain>";
                xml_str += m_domain;
                xml_str +=
            "</domain>"
            "<type>";
                xml_str += m_type;
                xml_str +=
            "</type>"
            "<identity>";
                xml_str += m_identity;
                xml_str +=
            "</identity>"
            "<description>";
                xml_str += "Hello Service";
                xml_str +=
            "</description>";
            xml_str += get_attributes_XML();
            xml_str += get_operations_XML();
            xml_str += "</mbean>";

        return CORBA::string_dup(xml_str.c_str());
    }
2   void HelloMBean::initialize_attributes()
    {
        m_hit_count_name = "TotalHelloCalls";

        HelloAttribute total_hits =
        {
```

Example 9: *Getting the MBean Description*

```
        m_hit_count_name, "long",
        "The total number of successful calls to
         HelloWorld::request_number() "
         "since the Hello Service started",
         IT_FALSE
    };
    m_attribute_list.push_back(total_hits);

    m_children_name = "Children";

    HelloAttribute children =
    {
        m_children_name, "list",
        "The list of children of this MBean",
        IT_FALSE
    };

    m_attribute_list.push_back(children);

    m_message_name = "Message";

    HelloAttribute message =
    {
        m_message_name, "string",
        "Message that this object emits",
        IT_TRUE
    };

    m_attribute_list.push_back(message);
}
```

3
```
IT_String HelloMBean::get_attributes_XML() const
{
    IT_String xml_str("");

    HelloAttributeList::const_iterator iter =
         m_attribute_list.begin();
    while (iter != m_attribute_list.end())
    {
        xml_str += get_attribute_XML(*iter);
        iter++;
    }
    return xml_str;
}
```

Example 9: *Getting the MBean Description*

```
IT_String HelloMBean::get_attribute_XML
   (HelloAttribute att) const
{
    IT_String xml_str =
    "<managed_attribute>"
        "<name>";
            xml_str += att.name;
            xml_str +=
        "</name>"
        "<type>";
            xml_str += att.type;
            xml_str +=
        "</type>"
        "<description>";
            xml_str += att.description;
            xml_str +=
        "</description>"
        "<property>"
            "<name>Access</name>"
            "<value>";
                xml_str += att.access ? "ReadWrite" : "Read";
                xml_str +=
            "</value>"
        "</property>"
    "</managed_attribute>";
  return xml_str;
}
.
.
.
```

This code extract is explained as follows:

1. The `get_description()` operation returns an XML string description of the managed entity, which is displayed by the Orbix management tools. This description normally includes the managed entity's attributes and operations (with parameters and return types). This string must be exact in order to parse correctly. This code example includes the `class_name`, `domain` and `type` attributes in the description.
2. The rest of the functions are local to this particular implementation, and are not defined in IDL. The `initialize_attributes()` function uses a locally-defined structure (`HelloAttribute`) to define a single

attribute. `HelloAttribute` is declared in `hello_mbean.h`. An instance of this attribute and anything else that you declare are pushed on to a list, including child MBeans.

3. The `HelloAttributeList` is then processed by `get_attributes_XML()` and by `get_attribute_XML()` to generate the description for display in the Administrator Web Console.

   There are similar functions for displaying the operations and their parameters in the console (`get_operation_XML()`, `get_operations_XML()` and `get_param_XML()`).

For full details of the `mbean.dtd` file used to display the XML string description, see Appendix A on page 97.

Step 4—Initializing the Management Plugin

Overview

After defining and implementing your MBeans, you should then initialize the the management plugin in your server implementation. The `instrumented_plugin` example adds the additional instrumentation code to the existing server implementation file.

Alternatively, for a larger application, you could create a separate instrumentation class, which is called by your server implementation.

Example management initialization

The following code extract is also from the server implementation file (`hello_world_impl.cxx`). It shows how the management plugin is initialized in the `instrumented_plugin` application:

Example 10: *Management initialization*

```
void HelloWorldImpl::initialize_management() IT_THROW_DECL(())
 {
1  if (!m_config->get_string("domain_name", m_domain_name))
   {
     cerr << "Couldn't get domain_name from config" << endl;
     m_domain_name = "<unknown domain>";
   }
  try
  {
   CORBA::Object_var obj;
   CORBA::String_var process_object_name;

2  obj = m_orb->resolve_initial_references("IT_Instrumentation");
  IT_Mgmt::Instrumentation_var instrument;
  instrument = IT_Mgmt::Instrumentation::_narrow(obj);

  if (CORBA::is_nil(instrument))
   {
    throw IT_String("Instrumentation reference is nil");
   }
.
.
.
```

This `hello_world_impl.cxx` code extract is described as follows:

1. The `get_string()` operation obtains the managed entity domain name. For more information, see "Get the managed entity ID and entity type" on page 75.
2. Like any other Orbix service, the management service must be initialized by your server implementation. The `resolve_initial_references()` operation obtains a reference to the management instrumentation interface, `IT_Instrumentation`. This is then narrowed to the `IT_Mgmt::Instrumentation` type.

   A managed entity must be registered with the instrumentation interface to be displayed in the Administrator Web Console.

Step 5—Creating your MBeans

Overview

After initializing the management service plugin, you can then create your MBeans in your server implementation. This section includes the following:

- "Creating an example MBean".
- "Removing your MBeans".

Creating an example MBean

The following is a continuation of the example in the last section, taken from the server implementation file. It shows how the MBean is created for the `instrumented_plugin` application:

Example 11: *Creating an MBean*

```
   void HelloWorldImpl::initialize_management()
     IT_THROW_DECL(())
   {
      .
      .
      .
       // Create and register the Hello MBean
       IT_Mgmt::ManagedEntity_var hello_mbean_ref;

1      hello_mbean_ref = m_hello_mbean_servant =
                                new HelloMBean(this,m_name.in());
       instrument->new_entity(hello_mbean_ref);

       if (m_is_parent)
       {

2       //Get the Process ObjectName
        process_object_name = instrument->get_process_object_name();

3       // Add the MBean as a child of the Process MBean.
        instrument->create_parent_child_relationship(
           process_object_name,
           hello_mbean_ref->managed_entity_id()
       );
     }
   .
   .
   }
```

This `hello_world_impl.cxx` code extract is described as follows:

1. You must create the MBean using the `new()` method, and register it as a managed entity using the `new_entity()` operation.
2. This gets the string that specifies the process object. The process object is displayed as the parent of the `HelloMBean` in the navigation tree of the Administrator Web Console. For more information about the process name, see "The Process MBean" on page 92.
3. This creates a parent-child relationship between your MBean and the Process MBean. The `create_parent_child_relationship()` operation takes two parameters:
   - The parent MBean name (in this case, the Process MBean).
   - The child MBean name (in this case, a reference to the `HelloMBean`).

   Creating a parent-child relationship adds the MBean to the navigation tree of the console.

Removing your MBeans

You might wish to remove an MBean in response to an administrator's interaction with the system. For example, in a banking application, if an account is deleted from the bank, it would be appropriate to remove the corresponding MBean for the account.

Removing an MBean unregisters it as a managed entity. This ensures that the MBean will no longer be displayed as part of the managed application.

To remove an MBean, use the `remove_entity()` operation. When the account's MBean has been removed, it is no longer displayed in the Administrator Web Console. The `remove_entity()` operation takes the managed entity name as a parameter.

The `instrumented_plugin` application is a simple example that does not remove any MBeans.

Further information

For full details of the Orbix Management API, see the *Orbix Management IDLdoc*.

Step 6—Connecting MBeans Together

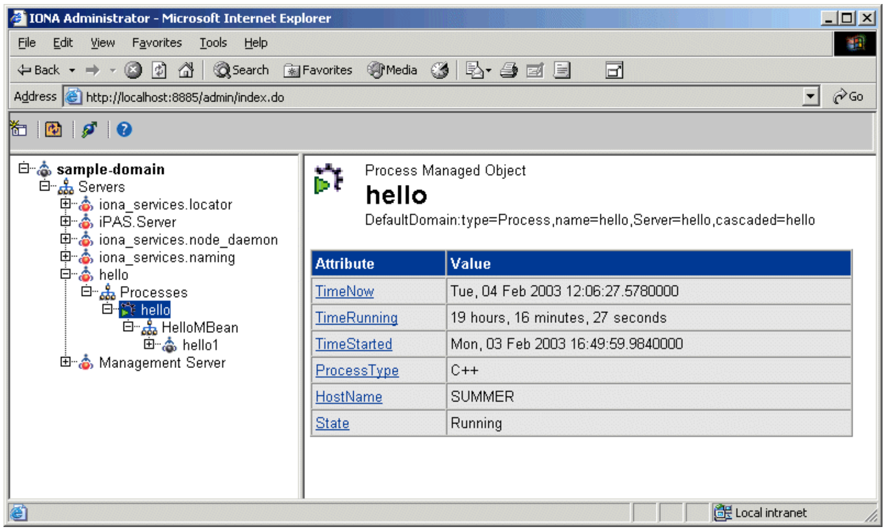
Overview

Applications are displayed in the Administrator Web Console as a series of related or connected MBeans, which can be monitored by administrators. This section explains how to connect your application MBeans together.

The Process MBean

The management service plugin creates a *Process MBean* when it is first loaded. A Process MBean is the default starting point in the console for navigation within a managed process. In the `instrumented_plugin` application, the `HelloMBean` is a child of the Process MBean.

Figure 20 shows the Process MBean for the `instrumented_plugin` application. The Process MBean has associated default attributes, displayed in the details pane (for example, process type, time running, hostname, and so on).


IONA Administrator - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8885/admin/index.do
Go
sample-domain
Servers
iona_services.locator
iPAS.Server
iona_services.node_daemon
iona_services.naming
hello
Processes
hello
HelloMBean
hello1
Management Server
Local intranet

Process Managed Object

hello

DefaultDomain:type=Process,name=hello,Server=hello,cascaded=hello

Attribute	Value
TimeNow	Tue, 04 Feb 2003 12:06:27.5780000
TimeRunning	19 hours, 16 minutes, 27 seconds
TimeStarted	Mon, 03 Feb 2003 16:49:59.9840000
ProcessType	C++
HostName	SUMMER
State	Running

Figure 20: *Instrumented Plugin Process MBean*

Creating parent–child relationships

Use the `create_parent_child_relationship()` operation to connect two MBeans together. This enables MBeans to appear as children of others in the navigation tree on the left of the console.

“Creating an example MBean” on page 90 shows how to use this operation to add your application MBean as a child of the Process MBean. In Example 12, the `add_child()` function shows how to add further child MBeans created by your application to the navigation tree.

Example 12: *Creating Child MBeans*

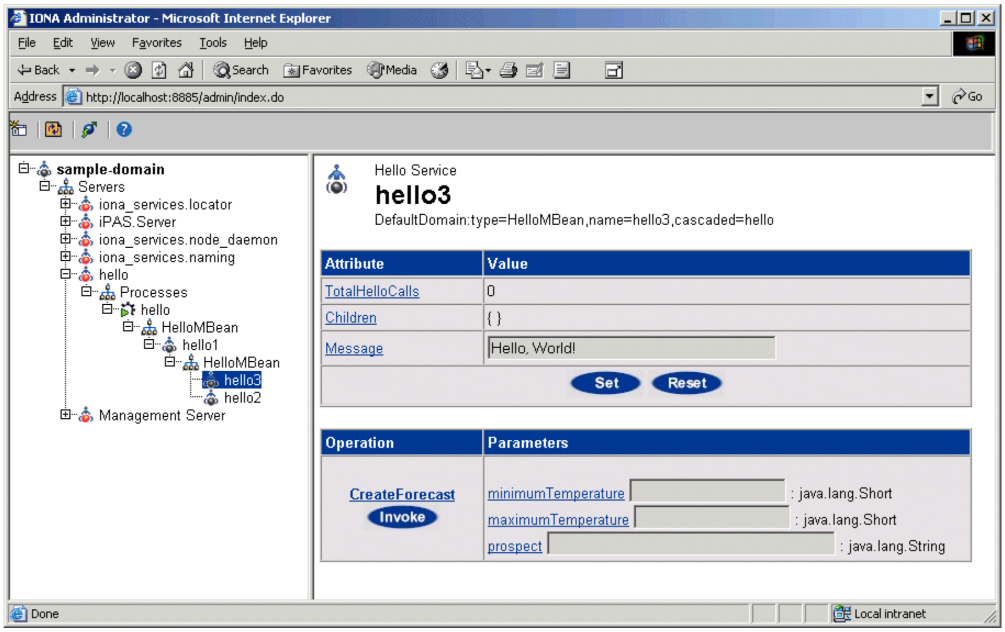
```
    void HelloWorldImpl::add_child(HelloWorldImpl *child)
        IT_THROW_DECL(())
    {
      // Lock mutex
     try
     {
1     CORBA::Object_var obj;
      obj = m_orb->resolve_initial_references("IT_Instrumentation");
      IT_Mgmt::Instrumentation_var instrument;
      instrument =  IT_Mgmt::Instrumentation::_narrow(obj);

      if (CORBA::is_nil(instrument))
        {
         throw IT_String("Instrumentation reference is nil");
         }

      CORBA::String_var my_name, child_name;

2     my_name = m_hello_mbean_servant->managed_entity_id();

      IT_Mgmt::ManagedEntity_var childMBean = child->get_mbean();

      child_name = childMBean->managed_entity_id();

3     instrument->create_parent_child_relationship(
        my_name.in(),
        child_name.in()
      );
```

Example 12: *Creating Child MBeans*

```
4      m_children.push_front(child);
    }
    catch(IT_Mgmt::ManagementBindFailed& ex)
     {
     cerr << "Management bind failed: " << ex << endl;
     m_is_managed = IT_FALSE;
     }
    .
    .
    .
  }
```

This `hello_world_impl.cxx` code extract is described as follows:

1. The `resolve_initial_references()` operation obtains a reference to the management instrumentation interface, `IT_Instrumentation`. This is then narrowed to the `IT_Mgmt::Instrumentation` type. All managed entities must be registered with the instrumentation interface to be displayed in the Administrator Web Console.
2. The `managed_entity_id()` operation is used to uniquely identify the managed entity.
3. The `create_parent_child_relationship()` operation takes the parent MBean and the child MBean as parameters.
4. This adds the child MBean to the list of MBeans. These steps add the child MBean to the tree for display in console. For example, Figure 21 shows a child MBean for the `instrumented_plugin` application (in this example, **hello3**).


IONA Administrator - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8885/admin/index.do
Go
sample-domain
Servers
iona_services.locator
iPAS.Server
iona_services.node_daemon
iona_services.naming
hello
Processes
hello
HelloMBean
hello1
HelloMBean
hello3
hello2
Management Server
Done
Local intranet

Hello Service

hello3

DefaultDomain:type=HelloMBean,name=hello3,cascaded=hello

Attribute	Value
TotalHelloCalls	0
Children	{ }
Message	Hello, World!
	Set Reset

Operation	Parameters
CreateForecast Invoke	minimumTemperature : java.lang.Short maximumTemperature : java.lang.Short prospect : java.lang.String

Figure 21: *Instrumented Plugin Child MBean*

APPENDIX A

MBean Document Type Definition

This appendix lists the contents of the mbean.dtd file used to generate the display of the Administrator Web Console.

In this appendix

This appendix contains the following section:

The MBean Document Type Definition File

Overview

The mbean.dtd file used to generate the XML used in the display of the Administrator Web Console. For example, the `get_description()` operation returns an XML string description of the managed entity, which is then displayed by the console. This description normally includes the managed entity's attributes and operations (with parameters and return types).

mbean.dtd contents

The contents of the mbean.dtd file is as follows:

```
<!-- MBean is the top level element -->
<!ELEMENT mbean (class_name, domain, identity, agent_id,
   description, notification_listener*, notification_filter*,
   notification_broadcaster*, constructor*, operation*,
   managed_attribute*)>

<!-- IMMEDIATE MBEAN PROPERTIES -->
<!ELEMENT class_name (#PCDATA)>
<!ELEMENT domain (#PCDATA)>
<!ELEMENT identity (#PCDATA)>
<!ELEMENT agent_id (#PCDATA)>

<!-- COMMON ELEMENT TYPES  -->

<!-- type = void | byte| char | double | float | long | longlong
    | short | boolean | string | list | ref | UNSUPPORTED -->
<!ELEMENT type (#PCDATA)>

<!ELEMENT name (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT param (name, type, description)>

<!-- NOTIFICATION details - note no recipients are shown for the
   broadcasts -->
<!ELEMENT notification_listener EMPTY>
<!ELEMENT notification_filter EMPTY>
<!ELEMENT notification_broadcaster EMPTY>
```

```
<!-- CONSTRUCTORS -->
<!ELEMENT constructor (name, description, param*)>

<!-- OPERATIONS -->
<!ELEMENT operation (name, type, description, param*)>

<!-- MANAGED ATTRIBUTES -->
<!ELEMENT managed_attribute (name, type, description,
   property*)>

<!-- PROPERTIES -->
<!-- name = Access -->
<!ELEMENT property (name, value)>
<!-- value = ReadWrite | ReadOnly | INACCESSIBLE -->
<!ELEMENT value (#PCDATA)>
```

Glossary

Administration
All aspects of installing, configuring, deploying, monitoring, and managing a system.

Application Server
A software platform that provides the services and infrastructure required to develop and deploy middle-tier applications. Middle-tier applications perform the business logic necessary to provide web clients with access to enterprise information systems. In a multi-tier architecture, an application server sits beside a web server or between a web server and enterprise information systems. Application servers provide the middleware for enterprise systems.

CORBA
Common Object Request Broker Architecture. An open standard that enables objects to communicate with one another regardless of what programming language they are written in, or what operating system they run on.

Configuration
A specific arrangement of system elements and settings.

Controlling
The process of modifying the behavior of running software components, without stopping them.

Details Pane
The display pane on the right hand side of the Administrator Web Console user interface.

Deployment
The process of distributing a configuration or system element into an environment.

Domain
An abstract grouping of managed server processes and hosts within a physical location. Processes within a domain share the same configuration and distributed application infrastructure. A domain is equivalent to an Orbix configuration domain.

EJB

Enterprise Java Beans. Oracle's architecture for the development and deployment of reusable, object-oriented, middle-tier components. EJBs can be either session beans or entity beans. EJB enables the implementation of a multi-tier, distributed object architecture. See https://www.oracle.com/java/technologies/javaee/enterprise-javabeans-technology.html

Event

An occurrence of interest, which is emitted from a managed entity.

Host

Generic term used to describe a computer, which runs parts of a distributed application.

Installation

The placement of software on a computer. Installation does not include Configuration unless a default configuration is supplied.

Instrumentation

Code instructions that monitor specific components in a system (for example, instructions that output logging information on screen.) When an application contains instrumentation code, it can be managed using a management tool such as the Orbix management tools.

Invocation

A request issued on an already active software component.

JRE

Java Runtime Environment. A subset of the Java Development Kit required to run Java programs. The JRE consists of the Java Virtual Machine, the Java platform core classes and supporting files. It does not include the compiler or debugger.

JMX

Java Management Extensions. Oracle’s standard for distributed management solutions. JMX provides tools for building distributed, Web-based solutions for managing devices, applications and service-driven networks.

Managed Application

An abstract description of a distributed application, which does not rely on the physical layout of its components.

Managed Entity

A generic manageable component (C++ or Java). Managed entities include managed domains, servers, containers, modules, and beans.

A managed entity acts as a handle to your application object, and enables the object to be managed. The terms managed entity and MBean are used interchangeably in this document.

Managed Server

A set of replicated managed processes. A managed process is a physical process which contains an ORB and which has loaded the management plugin. The managed server can be an EJB application server, CORBA server, or any other instrumented server that can be managed by the Orbix management tools.

Managed Process.

A physical process which contains an ORB and which has loaded the management plugin.

Management

To direct or control the use of a system or component. Sometimes used in a more general way meaning the same as Administration.

MBean

A JMX term used to describe a generic manageable object.

An MBean acts as a handle to your application object, and enables the object to be managed. The terms managed entity and MBean are used interchangeably in this document.

Monitoring

Observing characteristics of running instances of software components. Monitoring does not change a system.

Navigation Tree

The tree on the left hand side of the Administrator Web Console.

Node

A node represents a host machine on which the product is installed. The management service and managed servers are deployed on nodes.

ORB

CORBA Object Request Broker. This is the key component in the CORBA architecture model. It acts as the middleware between clients and servers.

Process

This is the operating system execution environment in which system and application programs execute. A Java Virtual Machine (JVM) is a special type of process that runs Java programs. A process that is not running Java programs is referred to as a standard or C++ process.

Process MBean

The is the first-level MBean that is exposed for management of an application. It is the starting point for navigation through an application in the Administrator Web Console

Resource

This represents shared data or services provided by a server. Examples of J2EE resources include JDBC, JNDI, JMS, JCA, and so on. Examples of CORBA resources include naming service, implementation repository, trading service, notification service, etc.

Server

This is a collection of one or more processes on the same or different nodes that execute the same programs. The processes in a server are tightly coupled, and provide equivalent service. This means that the calling client does not care which process ends up servicing the request.

Runtime Administration, Runtime Management

Encompasses the running, monitoring, controlling and stopping of software components.

SNMP

Simple Network Management Protocol. The Internet standard protocol developed to manage nodes on an IP network. It can be used to manage and monitor all sorts of devices (for example, computers, routers, and hubs)

Starting

The process of activating an instance of a deployed software component.

Stopping

The process of deactivating a running instance of a software component.

Web Services

Web services are XML-based information exchange systems that use the Internet for direct application-to-application interaction. These systems can include programs, objects, messages, or documents.

Web Services Container

A Web services container provides an environment for deploying and running Web services. A Web services container is typically deployed and runs in an application server.

XML

Extensible Markup Language. XML is a simpler but restricted form of Standard General Markup Language (SGML). The markup describes the meaning of the text. XML enables the separation of content from data. XML was created so that richly structured documents could be used over the web. See `http://www.w3.org/XML/`

Index