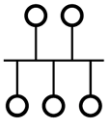

Orbix®

Mainframe Management User's Guide

Version 6.0, November 2003

Making Software Work Together™

Updated: 15-Dec-2003

M 3 1 7 2

Contents

List of Figures

Preface

Orbix Mainframe provides full integration with the IONA Orbix Management infrastructure, which provides support for enterprise-level management across different platform and programming language environments. IONA Administrator is a set of tools, integrated with IONA's Adaptive Runtime Technology, that enables seamless management of distributed enterprise applications.

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

Audience

Part 1 is aimed at OS/390 systems programmers managing distributed enterprise applications.

Part 2 is aimed at OS/390 application programmers writing distributed enterprise applications in C++ who wish to enable their applications for management by IONA Administrator. 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 OS/390 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.

Part 2, "Programmer's Guide"

This part is aimed at OS/390 application programmers writing distributed enterprise applications in C++ who wish to enable their applications for management by IONA Administrator. It explains how to enable CORBA C++ applications for management, and display them in IONA Administrator.

Related documentation

The Orbix Mainframe library includes the following related documentation:

- *Adminstrator's Guide*
- 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 `http://www.iona.com/support/docs/orbix/mainframe/6.0/index.xml`.

Additional resources

The IONA knowledge base contains helpful articles, written by IONA experts, about Orbix and other products. You can access the knowledge base at the following location:

`http://www.iona.com/support/knowledge_base/index.xml`

The IONA update center contains the latest releases and patches for IONA products:

`http://www.iona.com/support/updates/index.xml`

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 IONA Administrator

IONA Administrator is a set of tools that enables you to manage component-based distributed enterprise applications. This chapter introduces IONA Administrator and outlines typical administration tasks.

In this chapter

This chapter contains the following sections:

IONA Administrator

Overview

IONA Administrator is a set of IONA tools that enable you to manage and configure component-based distributed enterprise applications. It is integrated with IONA's *Adaptive Runtime Technology* (ART). This enables IONA Administrator to provide seamless management of IONA products and any applications developed using those products.

IONA Administrator is not aimed solely at any specific technology (for example, CORBA or Web services), but provides a generic management paradigm that enables the application to be managed without the administrator requiring knowledge of the technology used to create that application.

Scope of IONA Administrator

IONA Administrator enables 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 IONA web site:

`http://www.iona.com/products`

Assumptions

IONA Administrator does 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 IONA Administrator to manage any C++ system that has been enabled for management.

IONA Administrator Components

Overview

IONA Administrator includes the following main components:

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

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

IONA Administrator Web Console

The *IONA Administrator Web Console* provides a web browser interface to IONA Administrator. 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.

IONA Administrator Management Service

The *IONA Administrator 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 IONA Administrator Web Console and by the *IONA 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.

IONA Configuration Explorer

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

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.

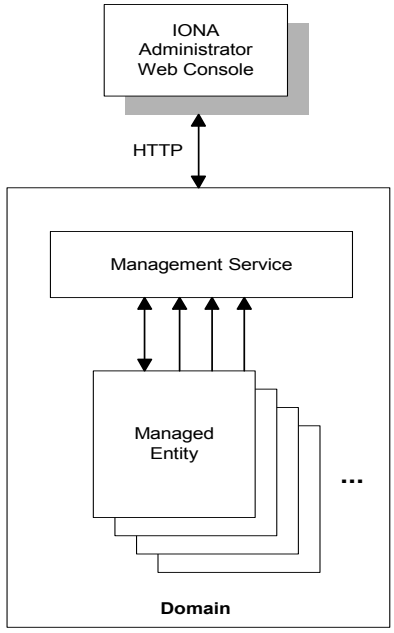

Figure 1: *IONA Administrator*

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

IONA 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. IONA Administrator therefore enables full instrumentation of server code. For information on how to write instrumentation code, see "Programmer's Guide" on page 23.

IONA Administrator Web Console

Overview

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

Figure 2 shows an example IONA Administrator Web Console interface.

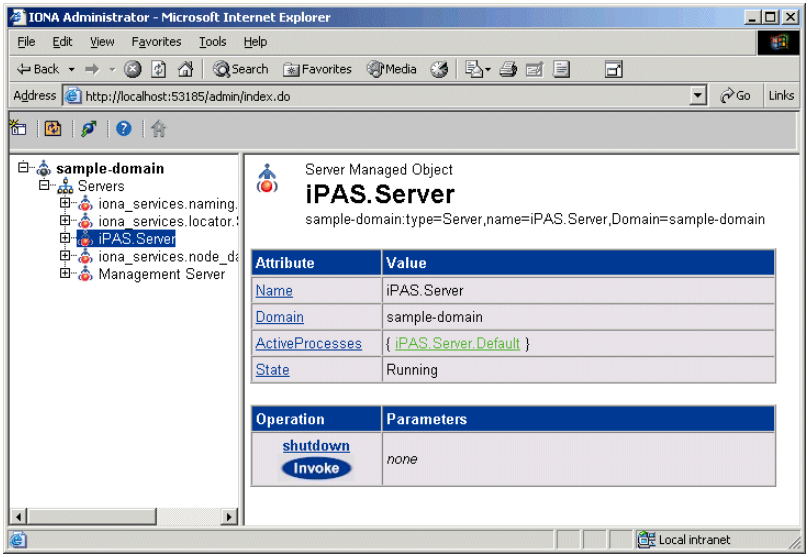

Figure 2: *IONA Administrator Web Console*

Multiple applications and domains

You can use one instance of the IONA 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 IONA Administrator 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.

Web Console architecture

Figure 3 gives an overview of this architecture. Each IONA 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.

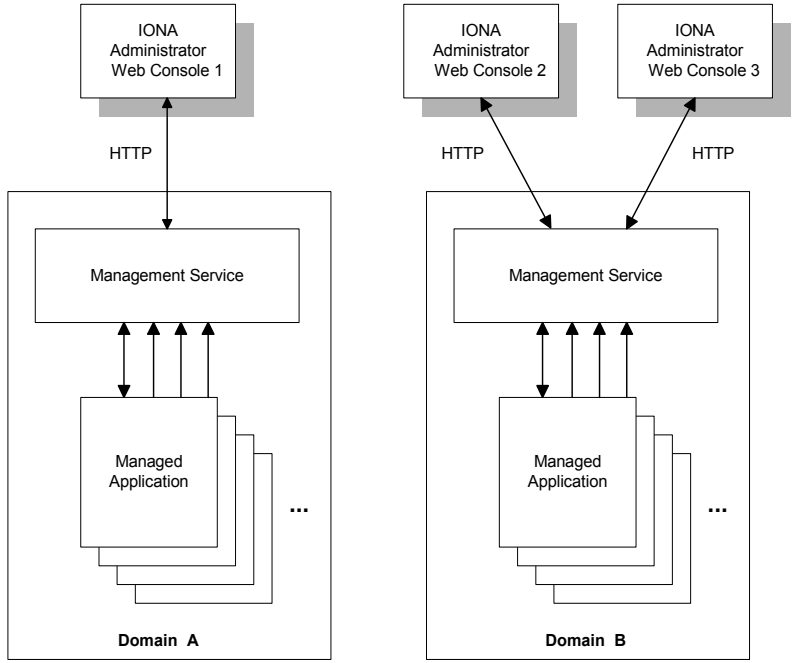

Figure 3: *IONA Administrator Web Console Architecture*

IONA Administrator Management Service

Overview

The IONA Administrator 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.

For more detailed information, see the *Management User's Guide* at `http://www.iona.com/support/docs/e2a/asp/6.0/admin.xml`.

IONA Configuration Explorer

Overview

The IONA 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.

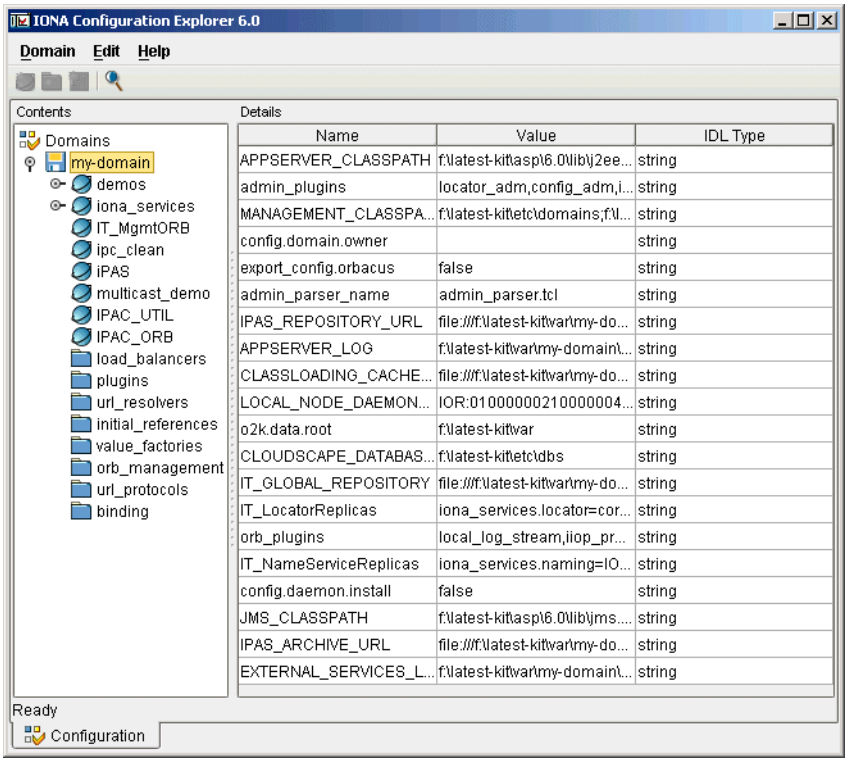

Figure 4: *IONA Configuration Explorer*

Multiple Domains

You can use a single instance of the IONA Configuration Explorer to manage configuration of multiple domains, both locally and on remote host machines. The IONA 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.

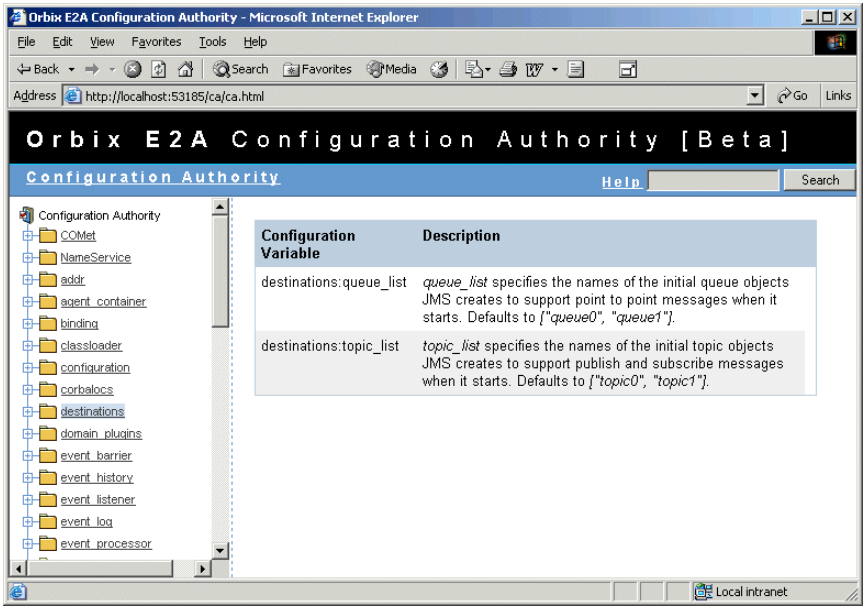

Figure 5: *Orbix Configuration Authority*

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.

For more detailed information about the Orbix Configuration Authority, see see the *Management User's Guide* at http://www.iona.com/support/docs/e2a/asp/6.0/admin.xml.

IONA Administrator Tasks

Overview

Typical Orbix management tasks that you can perform with IONA Administrator 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 IONA Administrator Web Console, see the *Management User's Guide* at `http://www.iona.com/support/docs/e2a/asp/6.0/admin.xml`.

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 IONA Administrator Web Console, see the *Management User's Guide* at `http://www.iona.com/support/docs/e2a/asp/6.0/admin.xml`.

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 IONA Administrator Web Console, see the *Management User's Guide* at http://www.iona.com/support/docs/e2a/asp/6.0/admin.xml.

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 http://www.iona.com/support/docs/e2a/asp/6.0/admin.xml.

Getting started

For details of how to get started with the IONA Administrator Web Console, see the *Management User's Guide* at http://www.iona.com/support/docs/e2a/asp/6.0/admin.xml.

CHAPTER 2

Managing Orbix Mainframe Services

Orbix Mainframe provides full integration with the IONA Orbix Management infrastructure. This allows Orbix servers running on the mainframe to be monitored from a centralized location, using IONA Administrator. This chapter provides details on Orbix Mainframe instrumentation and the configuration items involved in managing Orbix Mainframe services.

In this chapter

This chapter discusses the following topics:

Introduction

Overview

This section provides an introductory overview of how IONA Administrator components are used in the management of Orbix services running on OS/390.

Graphical overview

Figure 6 provides a graphical overview of how IONA Administrator components such as the Web Console and Management Service are used in the management of Orbix services on OS/390.

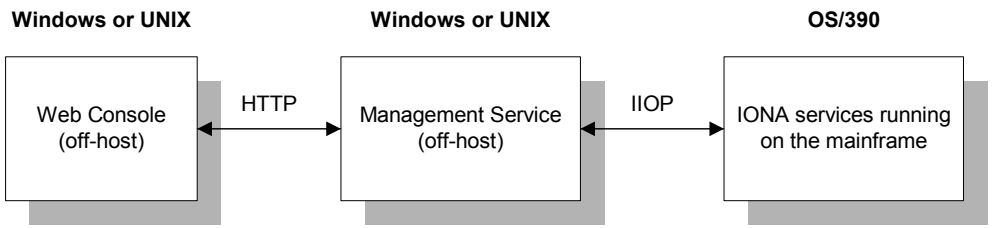

Figure 6: *IONA Administrator Integration with OS/390*

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 OS/390 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 23.

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/6.0/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 OS/390. 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 OS/390, 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 OS/390).

Configuration steps

The steps to enable the management of Orbix services on OS/390 are:

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

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

   The IOR 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";
```

Monitoring Orbix Services on OS/390

Overview

This section outlines the steps to monitor Orbix services on OS/390.

Steps

The steps to monitor Orbix services on OS/390 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 `http://www.iona.com/support/docs/e2a/asp/6.0/admin.xml`.

Part 2

Programmer's Guide

In this part

This part contains the following chapters:

CHAPTER 3

Introduction to Application Management

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

In this chapter

This chapter contains the following sections:

Introduction to IONA Administrator

Overview

IONA Administrator is a set of tools that enable administrators to configure, monitor and control distributed applications at runtime. Orbix provides seamless management of all IONA products, or any applications developed using those products, across different platform and programming language environments. IONA Administrator includes the following main components:

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

IONA Administrator Web Console

The *IONA Administrator Web Console* provides a web browser interface to IONA Administrator. 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 7.

IONA Administrator Management Service

The *IONA Administrator 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 IONA Administrator Web Console and by the *IONA Configuration Explorer*.

Note: Managed OS/390 applications can be written in C++. CORBA C++ applications use the management service process, `iona_services.management`.

IONA Configuration Explorer

The *IONA 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 7 shows how IONA Administrator interacts with managed applications to provide management capabilities.

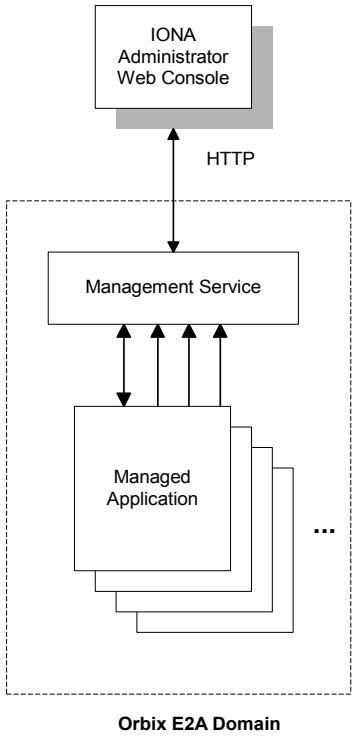

Figure 7: *IONA Administrator Components*

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 IONA Administrator, see the *Management User's Guide*.

Introduction to Java Management Extensions

Overview

Java Management Extensions (JMX) is a standards-based API from Sun 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, IONA Administrator).

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 8 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).

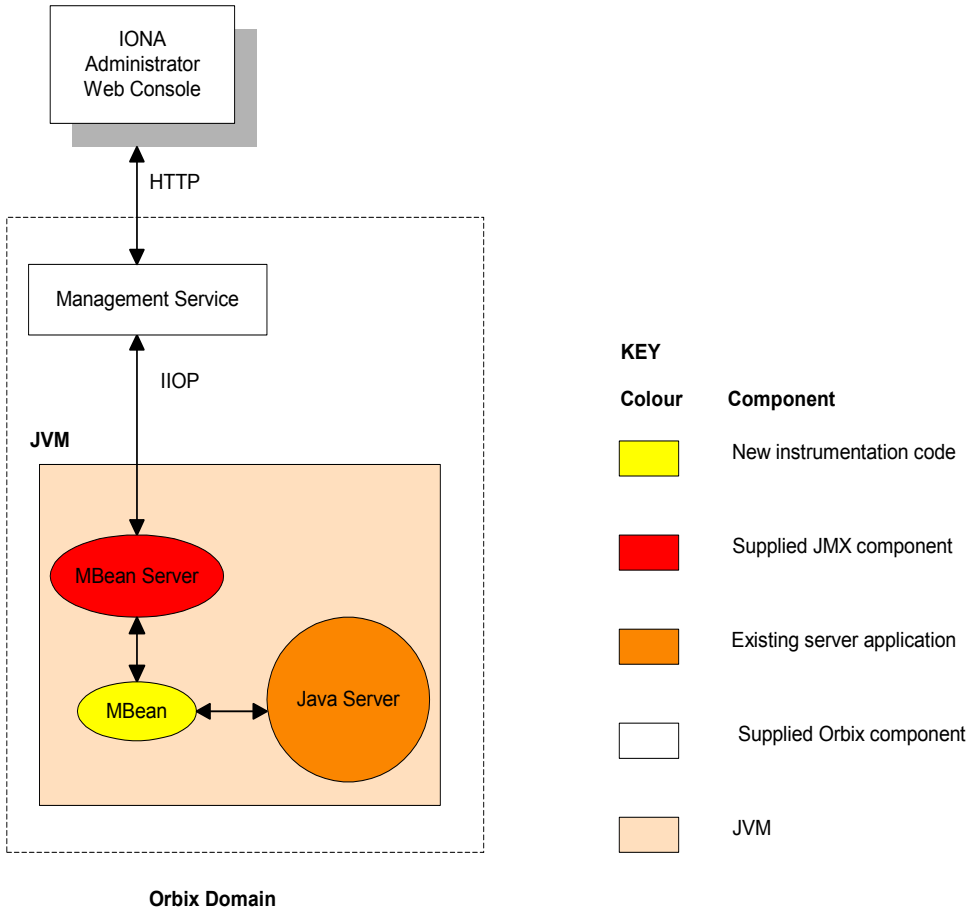

Figure 8: *JMX Management and Orbix*

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 8 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 Sun's JMX Instrumentation and Agent Specification, and Reference Implementation Javadoc. These documents are available online at:

`http://java.sun.com/products/JavaManagement/`

For information on how to integrate IONA Administrator with other general purpose management applications (for example, HP Openview™ or CA UniCenter™), 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. IONA'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 IONA Administrator. 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 8 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 IONA Administrator Web Console.

Figure 9 shows example parent–child relationships displayed in the left pane of the IONA Administrator Web Console.

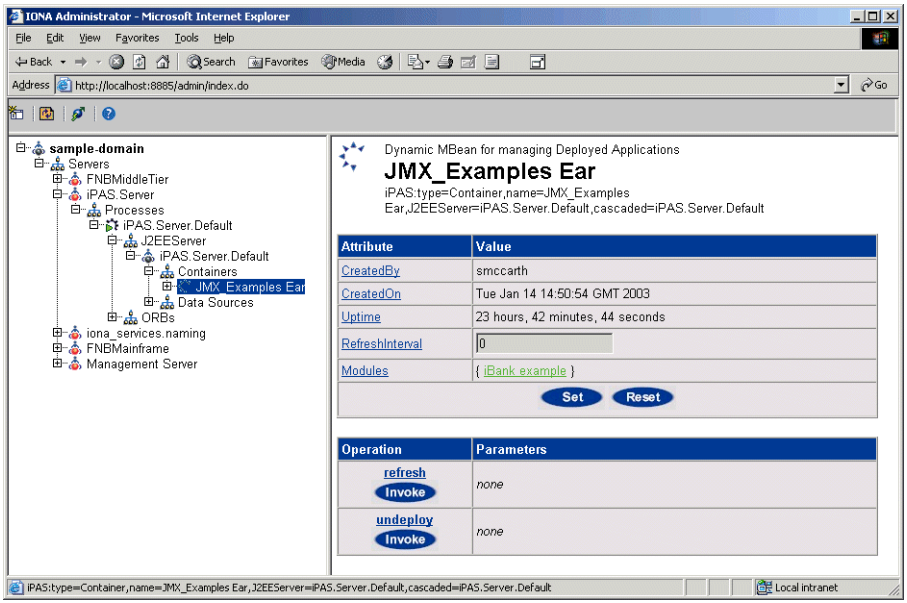

Figure 9: *Example Parent–Child Relationship*

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 37.

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 IONA Administrator. 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 IONA Administrator Web Console, as shown in Figure 9 on page 32.

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/6.0/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 4

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/6.0/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 IONA Administrator Console, shown in Figure 10.

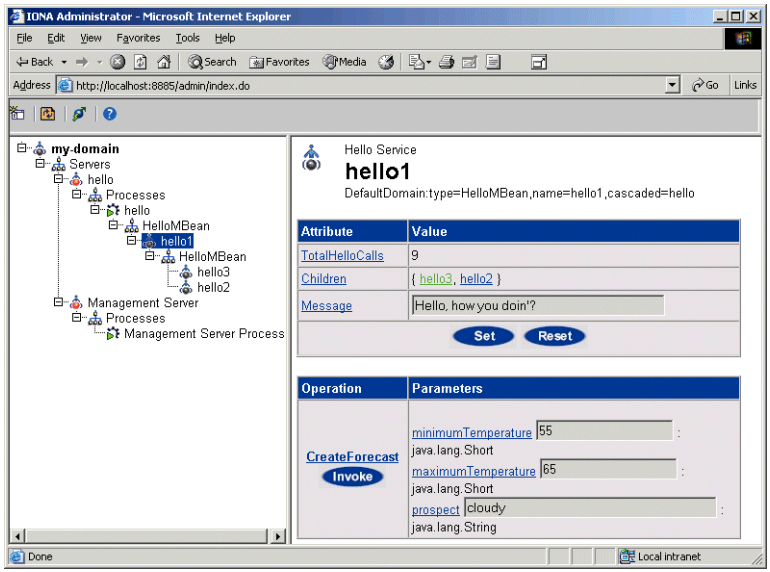

Figure 10: *Instrumented Plugin in IONA Administrator*

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/6.0/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 11 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.

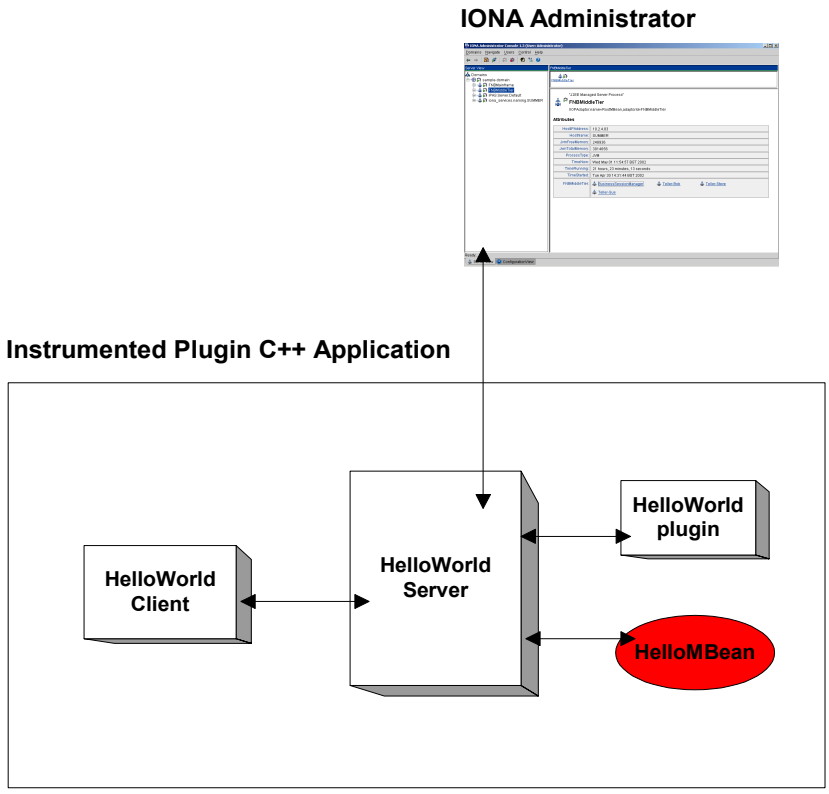

Figure 11: *Instrumented Plugin Application Overview*

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 IONA 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:

  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 IONA 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 IONA 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 56.

Further information

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

`http://java.sun.com/products/JavaManagement/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();
}
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 IONA 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 64.
2. The `HelloMBean()` destructor. For information about unregistering MBeans as managed entities, see "Removing your MBeans" on page 65.

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*

```
IT_Mgmt::ManagedEntityIdentifier HelloMBean::managed_entity_id()
   IT_THROW_DECL((CORBA::SystemException))
{
   return CORBA::string_dup(m_identity.c_str());
}
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 IONA 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 52. 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 10 on page 39, the **Children** attribute and its child MBeans (**hello3** and **hello2**) are displayed in the IONA 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 10 on page 39, shows an example text greeting for the **Message** attribute in the IONA 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 10 on page 39, shows example parameter values for the **CreateForecast** operation in the IONA 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%.
```

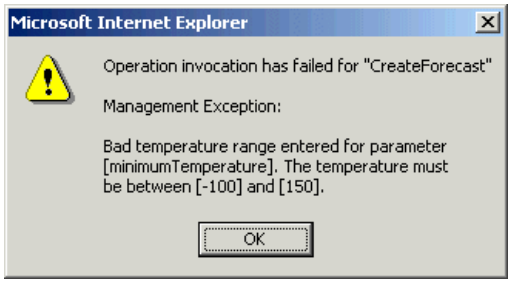

Figure 12: *Instrumented Plugin Custom Exception*

Get the MBean description

The following code shows how the MBean descriptions are obtained for display in the IONA 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);
}
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 IONA Administrator. 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 IONA 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 71.

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 49.
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 IONA 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 IONA Administrator Web Console. For more information about the process name, see "The Process MBean" on page 66.
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 IONA 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 IONA 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 13 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).

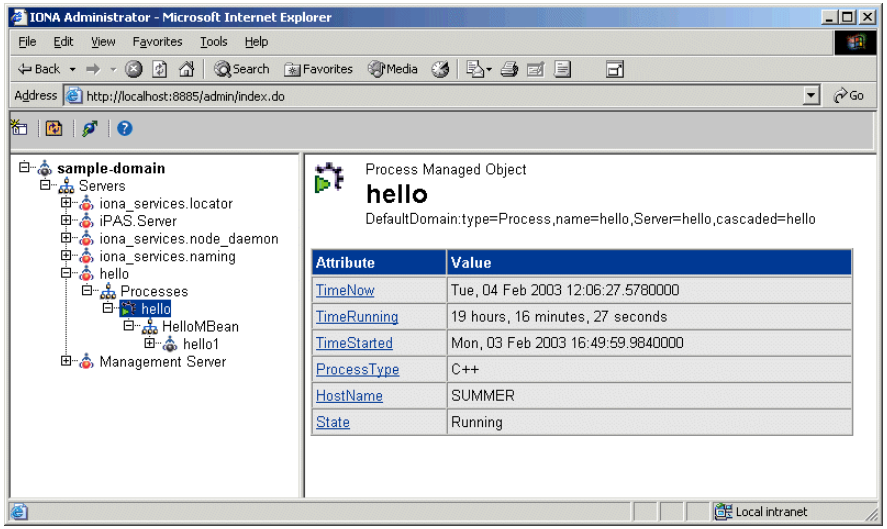

Figure 13: *Instrumented Plugin Process MBean*

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 64 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 IONA 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 14 shows a child MBean for the `instrumented_plugin` application (in this example, **hello3**).

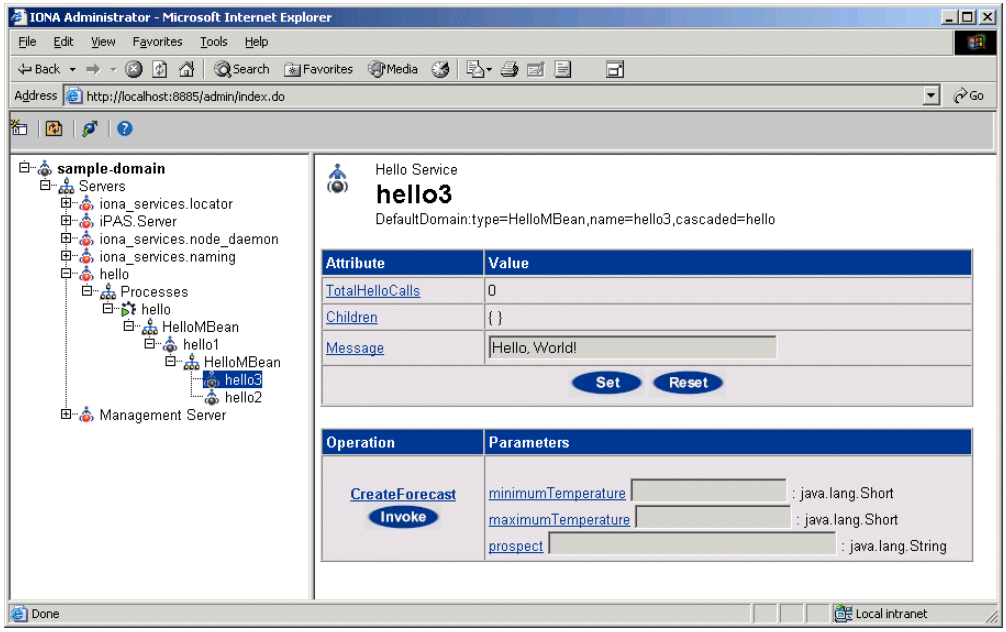

Figure 14: *Instrumented Plugin Child MBean*

APPENDIX A

MBean Document Type Definition

This appendix lists the contents of the mbean.dtd file used to generate the display of the IONA 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 IONA 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 IONA 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. Sun Microsystems' 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 http://java.sun.com/products/ejb/

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 IONA Administrator.

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. Sun'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 IONA Administrator.

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 IONA 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 IONA 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

P

R

S

V

W

X