Orbix Management Programme Guide

V6.3.14

Table of Contents

Preface

Orbix provides support for enterprise-level management across different platform and programming language environments. Orbix management tools enable administrators to manage distributed enterprise applications. This guide explains how programmers can enable applications to be managed by Orbix management tools (for example, Administrator).

Audience

This guide is aimed at developers writing distributed enterprise applications who wish to enable their applications for management by Orbix management tools. It assumes knowledge of either C++ or Java.

Related documentation

The document set for Orbix includes the following related documentation:

- *Management User's Guide*
- *Administrator's Guide*
- *CORBA Programmer's Guide, C++ Edition*
- *CORBA Programmer's Guide, Java Edition*

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 or Windows 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.

Introduction to Application Management

This section gives an overview of Orbix enterprise application management. It introduces the Orbix management tools, Sun's Java Management Extensions API, and the Orbix management API.

Note

The Administrator Web Console, Orbix Management Service and Orbix Configuration Explorer are no longer automatically installed. They are available as an optional component. To install, please download and extract the GUI components archive and follow the installation instructions.

The GUI components archive can be downloaded from our Software License & Download (SLD) site.

The GUI components have been separated from the product as they do not fully comply with the Section 508 and WCAG accessibility requirements and guidelines. It is intended that future product releases will include updated GUI components that comply with the relevant accessibility guidelines.

Introduction to Orbix Management Tools

Orbix management tools enable administrators to configure, monitor and control distributed applications at runtime. These tools provide seamless management of Orbix products, or any applications developed using those products, across different platform and programming language environments. 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 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 by the *Orbix Configuration Explorer*.

Note

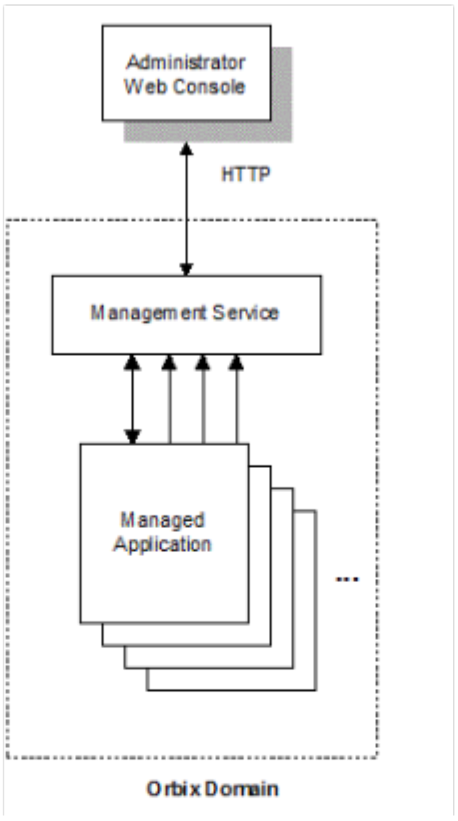
Managed applications can be written in C++ or Java. The same management service process (`iona_services.management`) can be used by Java and C++ applications.

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).

Figure 1 shows the Administrator Web Console, and how it interacts with managed applications to provide management capability.

Figure 1 Administrator Web Console



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.

Integrating with Enterprise Management Systems

Performance logging plugins enable Orbix to integrate effectively with Enterprise Management Systems (EMS), such as IBM Tivoli™, HP OpenView™, CA Unicenter™, or BMC Patrol™.

These 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.

Further information

For detailed information on using the Orbix management tools, and on how to configure EMS integration, see the *Orbix Management User's Guide*.

Introduction to Java Management Extensions

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 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, Administrator).

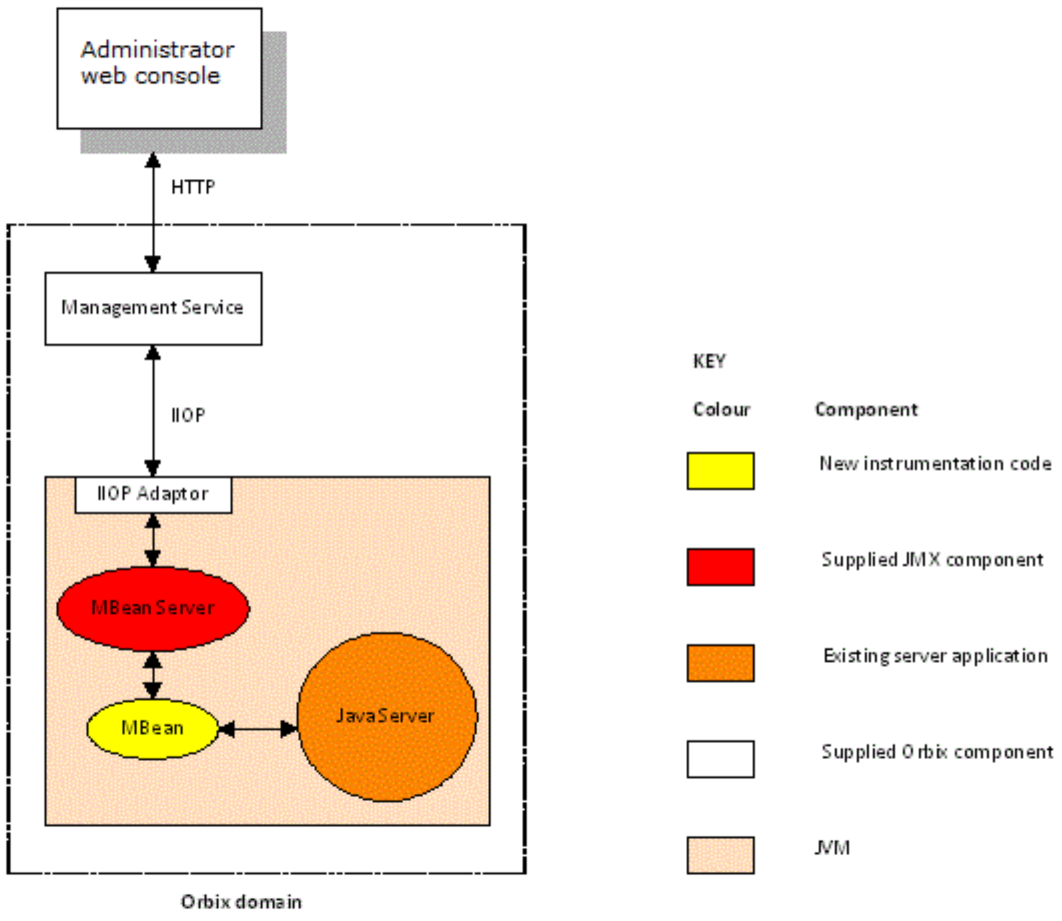
The MBean server

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 2 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 2 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 2 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 the JMX Instrumentation and Agent Specification, and Reference Implementation Javadoc. These documents are available online at:

```
[http://www.oracle.com/technetwork/java/javase/tech/
javamanagement-140525.html](http://www.oracle.com/technetwork/java/javase/
tech/javamanagement-140525.html)
```

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

Introduction to the Orbix management API

JMX does not specify how to remotely access MBeans using network protocols. The Orbix management API is used to enable remote communications for MBeans. This API also enables you to specify relationships between MBeans, and display MBeans in 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.

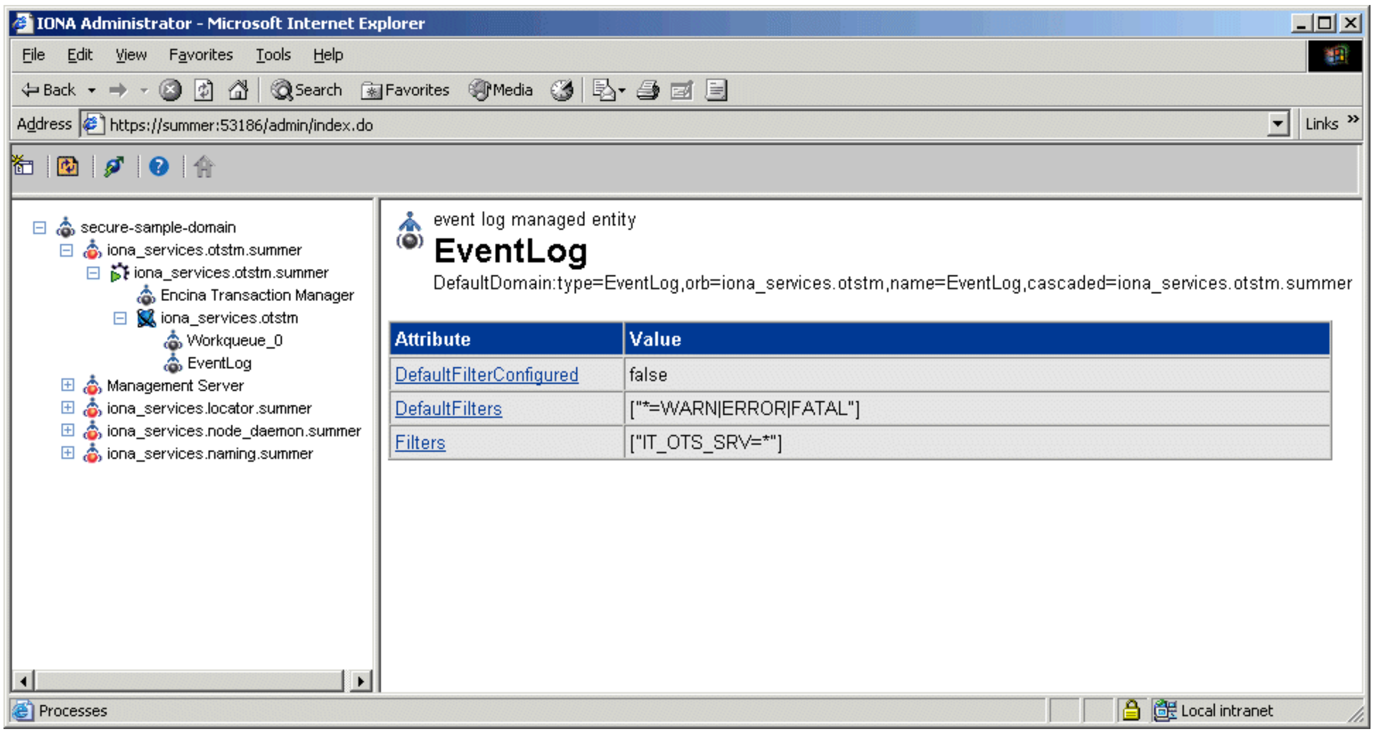
Figure 2 shows an example of this IIOP communication. This cross-platform API also enables communication for CORBA Java and C++ servers.

Defining MBean relationships

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

Figure 3 shows example parent–child relationships displayed in the left pane of the Administrator Web Console.

Figure 3 Example Parent–Child Relationship


IONA Administrator - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address https://summer:53186/admin/index.do
Links
secure-sample-domain
iona_services.otstm.summer
iona_services.otstm.summer
Encina Transaction Manager
iona_services.otstm
Workqueue_0
EventLog
Management Server
iona_services.locator.summer
iona_services.node_daemon.summer
iona_services.naming.summer
Processes
Local intranet

event log managed entity

EventLog

DefaultDomain:type=EventLog,orb=iona_services.otstm,name=EventLog,cascaded=iona_services.otstm.summer

Attribute	Value
DefaultFilterConfigured	false
DefaultFilters	["*=WARN|ERROR|FATAL"]
Filters	["IT_OTS_SRV=*"]

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 similar 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*, and the *Orbix Management Javadoc.*

Overview of Management Programming Tasks

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 the rest of this document.

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 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 Administrator Web Console, as shown in Figure 3.

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).

Further information

All of these management programming tasks are explained in detail, with examples, in these sections:

- CORBA Java Management CORBA Java management.
- CORBA C++ Management CORBA C++ management.

  It is not necessary to read one part before another. You can read these parts in any order.

CORBA Java Management

Instrumenting CORBA Java Applications

This section explains how to use the Java Management Extensions API and the Orbix Java Management API to enable an existing CORBA Java application for management. It uses a banking example application.

Step 1—Identifying Tasks to be Managed

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 user tasks
- New management tasks for administrators
- Planning your Programming Steps

Existing user tasks

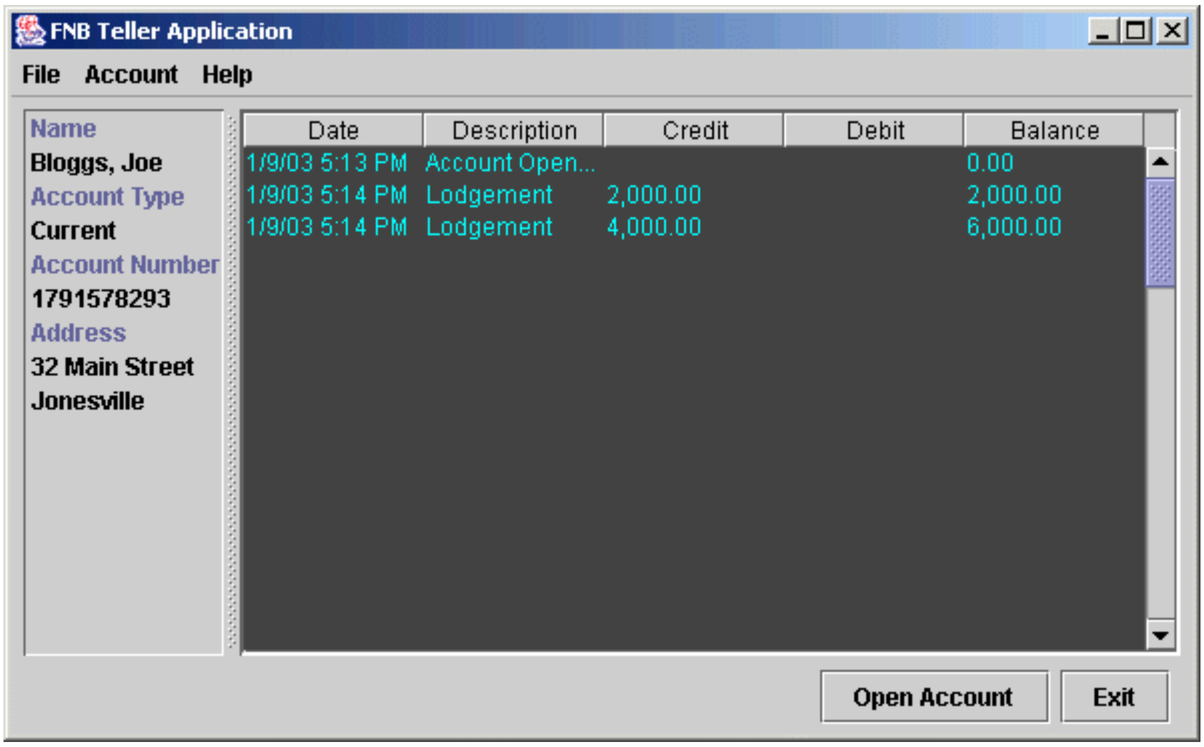
The First Northern Bank (FNB) example used in this chapter adds management capabilities to an existing CORBA Java banking application. This example application delivers standard banking services to customers.

The existing FNB application enables bank tellers to do the following:

- Log on and log off the system.
- Create a customer account.
- Lodge money into an account.
- Withdraw money from an account.

Figure 4 shows the user interface to these existing features.

Figure 4 Bank Teller Application



Name
Bloggs, Joe
Account Type
Current
Account Number
1791578293
Address
32 Main Street
Jonesville

Date	Description	Credit	Debit	Balance
1/9/03 5:13 PM	Account Open...			0.00
1/9/03 5:14 PM	Lodgement	2,000.00		2,000.00
1/9/03 5:14 PM	Lodgement	4,000.00		6,000.00

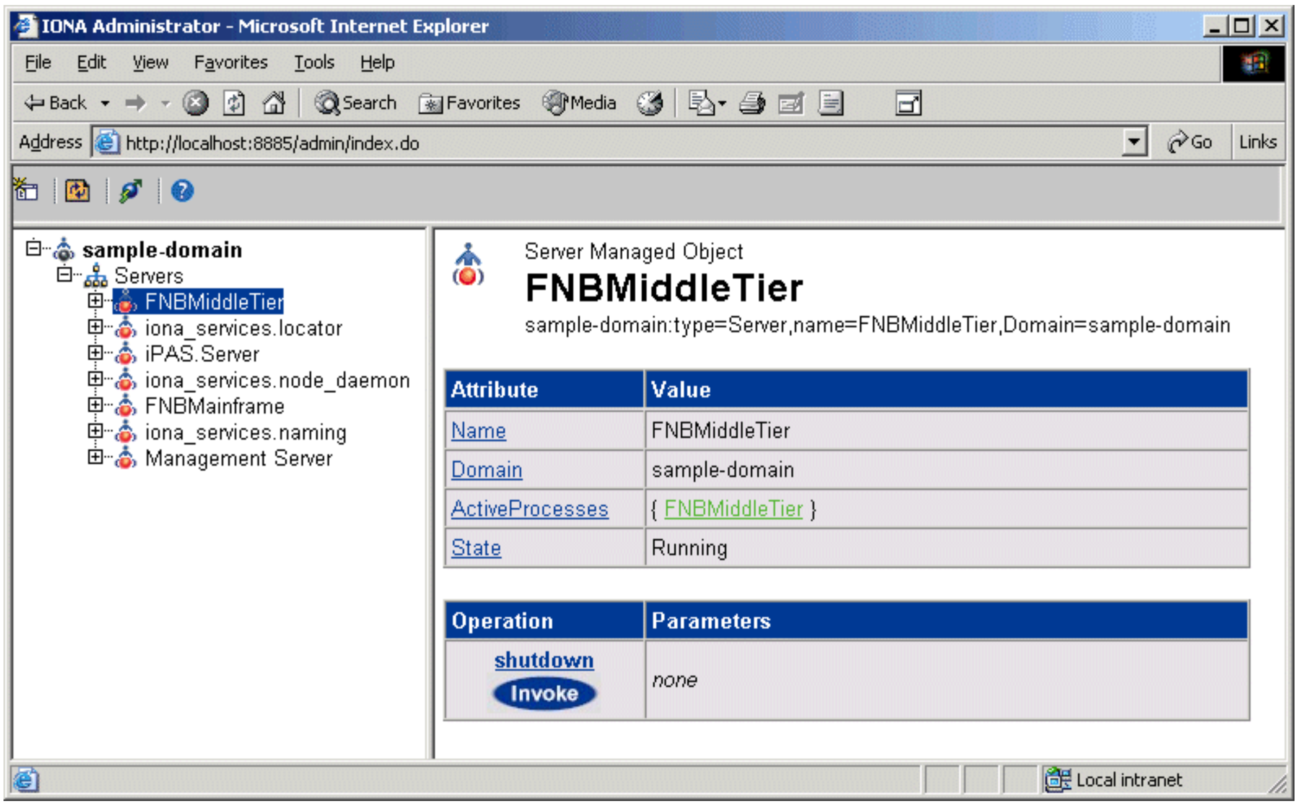
New management tasks for administrators

The new management instrumentation code added to FNB application enables administrators to do the following:

- Monitor the back-tier server
- Monitor customer accounts
- Unload account objects from memory
- Monitor the middle-tier server
- Monitor teller sessions
- Monitor bank tellers

Administrators can perform these tasks using the **Administrator Web Console**, shown in Figure 5.

Figure 5 Bank Example in Administrator


IONA Administrator - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8885/admin/index.do
Go Links
Local intranet

sample-domain
Servers
FNBMiddleTier
iona_services.locator
iPAS.Server
iona_services.node_daemon
FNBMainframe
iona_services.naming
Management Server

Server Managed Object
FNBMiddleTier
sample-domain:type=Server,name=FNBMiddleTier,Domain=sample-domain

Attribute	Value
Name	FNBMiddleTier
Domain	sample-domain
ActiveProcesses	{ FNBMiddleTier }
State	Running

Operation	Parameters
shutdown Invoke	*none*

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 your existing classes, and how much will be in new classes. Depending on the size of your application, you might wish to keep new instrumentation classes in a separate directory.

This chapter shows how JMX management code was added to the FNB CORBA Java application. It shows the standard programming steps. For example, defining and implementing MBeans, and registering and unregistering your MBeans with the MBean server.

Note

When instrumenting CORBA Java 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 Java implementation.

Step 2—Defining your MBeans

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 example MBean interfaces for the FNB application. It includes the following:

Rules for MBean interfaces

Each MBean object must implement an interface ending with the term `MBean` (for example, `BusinessSessionMBean`).

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

To expose management operations, you must declare an appropriate method in the MBean interface, and then implement it in the corresponding MBean class.

Example MBeans

Table 1 lists the example MBeans that are declared for the FNB application.

Table 1: FNB MBeans

MBean	Functionality
`AccountMgrMBean`	This back-tier MBean represents the bank account information managed by an administrator. For example, the number and type of accounts in the bank.
`CreditCardMBean`	This back-tier MBean represents credit card accounts.

MBean	Functionality
`BusinessSession ManagerMBean`	This middle-tier MBean represents the number of open bank teller sessions in the bank.
`BusinessSession MBean`	This middle-tier MBean represents the list of recent transactions for a particular bank teller session.

AccountMgrMBean interface

The interface for the `AccountMgrMBean` is defined as follows:

```
package bankobjects.management;
import javax.management.*;
import com.iona.management.jmx_iiop.*;
import com.iona.management.jmx_iiop.Public.*;
public interface AccountMgrMBean {
// attributes
public int getNumberOfAccounts();
public int getNumberOfCreditCards();
public int getNumberOfCurrentAccounts();
public int getNumberOfLoadedAccounts();
public ObjectName[] getActiveCreditCards();
// operations
public boolean unloadAccount (int accountNum);
}
```

CreditCardMBean interface

The interface for the CreditCardMBean is defined as follows:

```
package bankobjects.management;
import javax.management.*;
import com.iona.management.jmx_iiop.*;
import com.iona.management.jmx_iiop.Public.*;
public interface CreditCardMBean {
public int simpleOp ();
}
```

BusinessSessionManagerMBean interface

The interface for the BusinessSessionManagerMBean is as follows:

```
package fnbba.management;
import javax.management.*;
import com.iona.management.jmx_iiop.*;
import com.iona.management.jmx_iiop.Public.*;
public interface BusinessSessionManagerMBean {
public int getNumberOfOpenSessions ();
}
```

BusinessSessionMBean interface

The interface for the BusinessSessionMBean is defined as follows:

```
package fnbba.management;
import javax.management.*;
import com.iona.management.jmx_iiop.*;
import com.iona.management.jmx_iiop.Public.*;
public interface BusinessSessionMBean {
public String[] getRecentTransactionList();
}
```

MBean object names

MBean object names are used to uniquely identify an MBean. Object names are represented by the `javax.management.ObjectName` class, which extends the `java.lang.Object` class.

In the FNB example, the AccountMgrMBean interface declares the following `get()` method for the ActiveCreditCards attribute:

```
public ObjectName[] getActiveCreditCards();
```

This returns an array of MBean object names for the associated credit card accounts. The **getActiveCreditCards()** method is an example of using an object name to connect MBeans together.

Further information

For information about how to specify MBean object names, see Step 3—Implementing your MBeans.

For detailed information about the `ObjectName` class, see Oracle's JMX Reference Implementation Javadoc. This is available along with the source code from http://docs.oracle.com/javase/7/docs/api/javax/management/ObjectName.html.

Step 3—Implementing your MBeans

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

For example, this section shows interaction between an MBean (BusinessSessionManager) and the CORBA server implementation object (BusinessSessionManagerDelegate). The MBean's getNumberOfOpenSessions() method calls the implementation object's openSessions() method. This section includes the following:

- Example MBean implementation
- The management wrapper class
- Management wrapper implementation
- Identifying MBeans
- Further information

Example MBean implementation

The following code example shows the BusinessSessionManager implementation for the BusinessSessionManagerMBean:

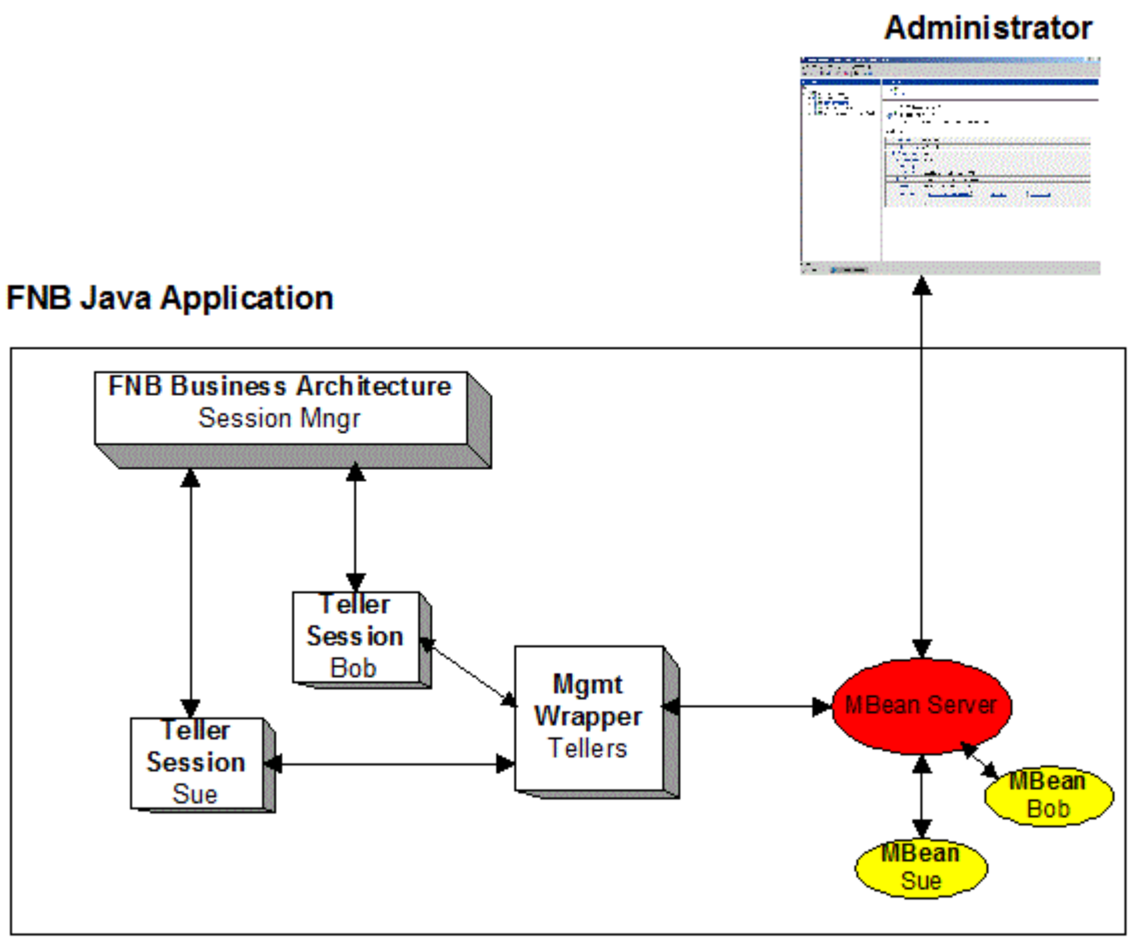
```
package fnbba.management;
import javax.management.*;
import com.iona.management.jmx_iiop.*;
import com.iona.management.jmx_iiop.Public.*;
public class BusinessSessionManager
implements BusinessSessionManagerMBean {
private ManagementWrapper mgmtWrapper = null;
private ObjectName myName = null;
private fnbba.BusinessSessionManagerDelegate myImpl = null;
public BusinessSessionManager (fnbba.BusinessSessionManagerDelegate myImpl){
this.myImpl = myImpl;
try { myName = new
ObjectName("FNBMiddleTier:name=BusinessSessionManager");
}
catch (Exception j) {}
public int getNumberOfOpenSessions()
{ return myImpl.openSessions(); }
public void remove ()
{ mgmtWrapper.removeMBean (myName);}
}
```

The management wrapper class

In this example, the MBean representing the bank teller BusinessSessionManager uses an underlying class (the ManagementWrapper class) to perform most of the work. The ManagementWrapper object creates the BusinessSessionMBeans for each bank teller session. It registers these beans with the MBean server, and then adds them to the Administrator Web Console display. A simplified overview is shown in Figure 6.

This is a typical MBean implementation, where the MBean uses the functionality of other application objects (in this case, the management wrapper) to provide the management capability. The management wrapper performs the core management tasks (for example, gaining access to the MBean server, and registering the MBean with the MBean server).

Figure 6 Bank Application Overview



Management wrapper implementation

The ManagementWrapper.instanc`e()` method that creates the MBean is defined as a `static` class method. This is because only one wrapper is required by each domain displayed by Administrator. For example, Figure 5 on page 16 shows the `FNBMiddleTier` node, which has a `FNBMiddleTier` MBean domain. Multiple wrappers representing multiple domains can be stored in an array of management wrappers. For example, you could add ATM support, which would use a separate management wrapper to manage the ATM sessions. For more information on MBean domain names, see Identifying MBeans.

The management wrapper code and the standard management tasks that it performs are explained in the sections that follow.

Identifying MBeans

An `ObjectName` must be a unique name in the MBean server. It includes an MBean domain name, separated from a list of name and value pairs by a colon. These name value pairs can be of any type or value. The syntax is:

`domain-name:name1=value1`,`name2=value2`,...

The object name used in the BusinessSessionManager example represents the following simple domain and name-value pair:

`FNBMiddleTier:name=BusinessSessionManager`

> **Note**
>
> The MBean domain name is not related to an Orbix configuration or location domain. This is purely a namespace for MBeans only.

Further information

For detailed information about the `ObjectName` class, see Oracle's JMX Reference Implementation Javadoc. This is available along with the source code from http://docs.oracle.com/javase/7/docs/api/javax/management/ObjectName.html

For another Java example, see the . This shows an MBean object name that specifies additional name-value pairs. This enables you to display more information in the Administrator Web Console.

Step 4—Gaining Access to an MBean Server

After defining and implementing your MBeans, you must gain access to an MBean Server. In the FNB example application, the MBean server is accessed by the management wrapper object. The management wrapper object performs the same tasks for different MBean implementations.

> **Note**
>
> You must explicitly load the management plugin (`it_mgmt`) for CORBA Java applications.

This section includes the following:

- Loading the Orbix management plugin
- Accessing the MBean server
- IT_IIOPAdaptorServer object
- Specifying an MBean object name

Loading the Orbix management plugin

You must first ensure that the Orbix management plugin (`it_mgmt`) is specified by your `orb_plugins` configuration variable in the appropriate configuration scope.

For example, the following settings are taken from the FNB configuration file:

```
FNBMiddleTier{
orb_plugins = ["it_mgmt", "iiop_profile", "giop", "iiop"];
};
FNBMainframe {
orb_plugins = ["it_mgmt", "iiop_profile", "giop", "iiop"];
};
```

Note

You must ensure that all settings are made in correct configuration scope (for example, **FNBMiddleTier**). Do not add the `it_mgmt` plugin to the `orb_plugins` variable in the global configuration scope.

Accessing the MBean server

The following code extract from the ManagementWrapper class shows how its constructor method accesses the default MBean server:

```
private ManagementWrapper (String ConfigDomainName) {
adaptorServer =
(IT_IIOPAdaptorServer)com.iona.management.jmx_iiop.IT_DynamicLoading.getDefault
try {
managedObjName = new ObjectName(ConfigDomainName);
mBeanServer = adaptorServer.getMBeanServer();
} catch (Exception ex) {
System.out.println("Unexpected exception while registering iBankMBean: " +
ex);
}
myConfigDomain = new String (ConfigDomainName);
processMBean =com.iona.management.jmx_iiop.IT_DynamicLoading.
getProcessObjectName();
}
```

IT_IIOPAdaptorServer object

In the ManagementWrapper class, the `IT_IIOPAdaptorServer` object is used to provide a reference to the MBean server. When you have accessed the default `MBeanServer` using the getMBeanServer() method, you can then register your MBeans with the MBean server.

For detailed reference information about `IT_IIOPAdaptorServer`, see the *Management Javadoc.*

Specifying an MBean object name

The ConfigDomainName parameter passed to ManagementWr`apper()` specifies the MBean object name used by the management wrapper, and which is displayed in Administrator as an MBean object. For example, the middle-tier `fnbba` server uses the following object name:

`FNBMiddleTier:name=FNBMiddleTier`

> **Note**
>
> The **ConfigDomainName** parameter is not related to the Orbix configuration or location domain. This is an MBean `ObjectName` domain is purely a namespace for MBeans only.

For more information, see Identifying MBeans.

The Process MBean

The process MBean is the starting point for navigation through a sever in the Administrator Web Console. In the console, application MBeans are displayed as nodes that are added to the process MBean in the navigation tree.

The ManagementWrapper obtains the process MBean's object name using the getProcessObjectName() method. This standard JMX call obtains the process MBean that will be used later to add the application MBean to the Administrator display. For more information, see Creating parent-child relationships.

Step 5—Registering your MBeans

After gaining access to the MBean server, you can then register your MBeans with the MBean server. Registering MBeans enables them to be monitored and controlled using Administrator. This section includes the following:

- Example MBean registration
- addMBean() implementation
- Registering MBeans
- Creating parent-child relationships

Example MBean registration

The following FNB example from the `BusinessSession` class first creates a MBean for a bank teller session, and then registers it with the MBean server. The MBean is registered using the management wrapper's `addMBean()` method:

```
public BusinessSession (fnbba.BusinessSessionDelegate myImpl,
String SessionName) {
this.myImpl = myImpl;
mgmtWrapper = ManagementWrapper.instance
("FNBMiddleTier:name=FNBMiddleTier");
try {
String t =new String ("FNBMiddleTier:name=" + SessionName);
myName = new ObjectName(t);
}
catch (Exception j) {}
mgmtWrapper.addMBean(this, myName);
}
```

addMBean() implementation

The `addMBean()` method is implemented in the `ManagementWrapper` class as follows:

```
public boolean addMBean (java.lang.Object mbean, ObjectName mbeanName )
{
System.out.println ("Registering mbean...");
try {
ObjectName tmpArray [] = new ObjectName [1];
tmpArray [0] = mbeanName;
mBeanServer.registerMBean(mbean, mbeanName);
          adaptorServer.createParentChildRelation(processMBean,tmpArray
);
}
catch ( Exception j ) {
System.err.println ("Exception in registering MBean " + j );
return false;
}
return true;
}
```

Registering MBeans

You can register MBean objects using either of the following approaches:

- Create the MBean object manually, and then register it with the MBean server. If you choose this approach, you must use the `new()` constructor and `registerMBean()` method.
- Create and register your MBean with the MBean server, using the `createMBean()` constructor. This registers the MBean automatically.

The FNB example uses the MBean server's `registerMBean()` method to register the MBean. The `registerMBean()` method takes two parameters:

- The MBean object instance (`mbean` in this example).
- An `ObjectName`, which is used to identify the MBean. The object name in this example is mbeanName. For more information on object names, see Identifying MBeans.

Creating parent-child relationships

The createParentChildRelation() method adds the MBean to the Process MBean. This is the starting point for navigation through a sever in the Administrator Web Console. The `createParentChildRelation()` method takes two parameters:

- The parent MBean `ObjectName`.
- The child MBean `ObjectName`.

For more information on the Process MBean and how it is displayed by Administrator, see Displaying CORBA Java Applications.

Step 6—Unregistering your MBeans

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

- Example MBean unregistration
- The unregisterMBean() method

Example MBean unregistration

To unregister an MBean, use the MBean server's `unregisterMBean()` method. In the FNB application, the `unregisterMBean()` method is called by the management wrapper's `removeMBean()` method. The following code extract is taken from the `BusinessSession` class:

```
public void remove ()
{
mgmtWrapper.removeMBean (myName);
}
```

The `removeMBean()` method is implemented in the management wrapper class as follows:

```
public boolean removeMBean (ObjectName mbean) throws Exception
{
mBeanServer.unregisterMBean (mbean);
return true;
}
}
```

The unregisterMBean() method

When the account's MBean has been unregistered, using the `unregisterMBean()` method, it is no longer displayed by the Administrator Web Console. All parent-child relationships between MBeans created using the `createParentChildRelation()` method are also removed.

The `unregisterMBean()` method takes an MBean object name as a parameter. For more information, see MBean object names.

Step 7—Connecting MBeans Together

Your application is 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 MBeans together. There are two different approaches:

- Connecting MBeans using a get() method
- Connecting MBeans using the createParentChildRelation() method

Connecting MBeans using a get() method

To connect two MBeans together using a `get()` method, you must create MBean methods that return MBean `ObjectName`s. For example, in the FNB application the `AccountMgr` MBean must be connected with the active `CreditCard` MBeans. The `AccountMgrMBean` interface declares the following `get()` method for the `ActiveCreditCards` attribute:

```
public ObjectName[] getActiveCreditCards();
```

This method returns an array of MBean object names for the associated credit card accounts. If this method returns object names that are already registered MBean names, these MBeans are displayed in the `ActiveCreditCards` attribute of the `CreditCard` MBean.

By using methods that return `ObjectName`s, you will see hyperlinks displayed in the details view on the right of the console. You can use these hyperlinks to navigate between managed objects like they are web pages. The navigation tree on the left is not affected.

Connecting MBeans using the createParentChildRelation() method

Using the `get()` method, hyperlinks between MBeans are displayed in the details view, on the right of the console. Alternatively, you can use `createParentChildRelation()` method to connect two MBeans together. This enables MBeans to appear as children of others in the tree view, on the left of the console.

The `createParentChildRelation()` method takes the parent and child MBeans as parameters, and is defined as follows:

```
public boolean createParentChildRelation(ObjectName parentObjName,
ObjectName[] childObjNames) throws
com.iona.common.management.relation.RelationServiceException
```

For an example of using this method, see addMBean() implementation.

Monitoring MBean Statistics

Optionally, you can also monitor statistics from MBeans in your own applications. The `it_mbean_monitoring` performance logging plug-in enables you to periodically harvest statistics associated with MBean attributes. This section includes the following:

- MBean monitoring
- Configuration steps
- Programming steps

MBean monitoring

The IT_MBeanMonitoring IDL interface provides the support for monitoring MBean statistics. This interface is defined as follows:

```
module IT_MBeanMonitoring
{
const string MANAGEMENT_MBEAN_MONITORING_INITIAL_REF =
"IT_MBeanMonitoringRegistration";
// Interface exceptions.
exception MBeanNotFound {};
exception MBeanAttributeNotFound {};
exception MBeanAttributeInvalidType {};
// IT_MBeanMonitoring::MBeanMonitoringRegistration
//
// An interface which provides a means to
// monitor and log statistics about mbeans
// registered with the management service.
local interface MBeanMonitoringRegistration
{
void monitor_attribute(
in string object_name,
in string attribute_name,
in string alias) raises ( MBeanNotFound,
MBeanAttributeNotFound, MBeanAttributeInvalidType);
void cancel_monitor(
in string object_name,
in string attribute_name,
in string alias) raises ( MBeanNotFound);
};
};
```

When the `it_mbean_monitoring` plug-in is included in your `orb_plugins` list, an initial reference is registered for the `IT_MBeanMonitoringRegistration` interface.

When you resolve on your application MBean, the IT_MBeanMonitoring API can be used to switch on, or turn off, monitoring of an application MBean. Statistics for user monitored MBeans will then appear in the performance logs.

Configuration steps

You must ensure that the `it_mbean_monitoring` plug-in is included in your `orb_plugins` list.

In addition, the following Orbix JAR file must be included on your classpath:

```
$IT_PRODUCT_DIR/lib/./art/java_management_logging/1.2/perf_logging.jar
```

Programming steps

This example assumes that you already have an MBean with an attribute that you want to be sampled and logged. For example, the MBean might track the memory currently being used by the process. The programming steps are as follows:

1. Import the following package:

```
import
com.iona.management.logging.IT_MBeanMonitoring.MBeanMonitoringRegistration;
```

2. To register your MBean with the `it _mbean_monitoring` plug-in, you must first resolve on the MBean monitoring initial reference:

```
// Resolve initial reference for MBeanMonitoringRegistration object.
MBeanMonitoringRegistration mbeanMonitoringRegistration =
(MBeanMonitoringRegistration)
orb.resolve_initial_references(IT_MBeanMonitoringRegistration);
```

3. You can then register the attribute to be monitored by specifying your MBean details to `monitor_attribute()`:

```
// Turn on monitoring for mbean attribute.
mbeanMonitoringRegistration.monitor_attribute("*mbean_name*","*attribute
name"*, "*mbean_friendly_name*");
```

The *mbean_friendly_name* is an alternative alias that will also appear in the log file.

Further information

For more details on Orbix performance logging, see the *Orbix Management User’s Guide*.

Displaying CORBA Java Applications

This section explains how to display CORBA applications in the Administrator Web Console in more detail. It explains the concept of the Process MBean, how to add MBeans to the navigation tree, and how to customize your icons.

Note

The Administrator Web Console is no longer automatically installed. It is available as an optional component. To install, please download and extract the GUI components archive and follow the installation instructions.

The GUI components archive can be downloaded from our Software License & Download (SLD) site.

The GUI components have been separated from the product as they do not fully comply with the Section 508 and WCAG accessibility requirements and guidelines. It is intended that future product releases will include updated GUI components that comply with the relevant accessibility guidelines.

Displaying MBeans

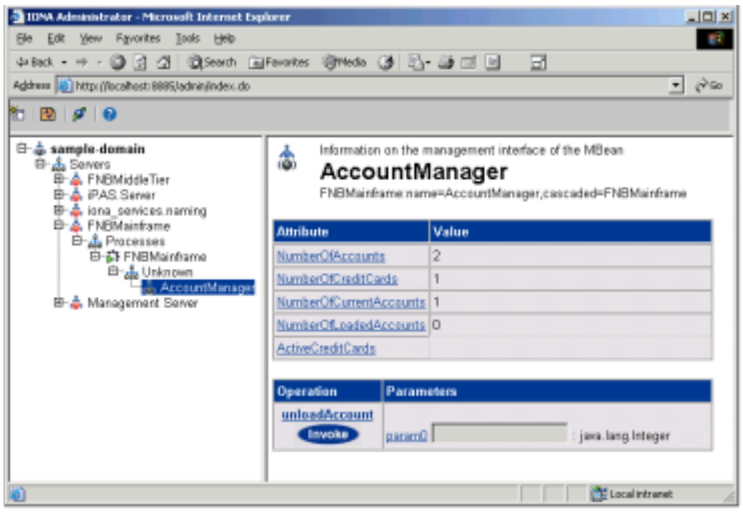
This section explains how MBeans are displayed by Administrator. It includes the following:

- Administrator Web Console
- The Process MBean
- Example Process MBean

Administrator Web Console

The Administrator Web Console is shown in Figure 7. This example shows the managed attributes and operations for the FNB `AccountManager` object. The attributes and operations displayed correspond to those declared in .

Figure 7 Account Manager Example


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
FNBMiddleTier
iPAS Server
iona_services.naming
FNBMainframe
Processes
FNBMainframe
Unknown
AccountManager
Management Server
Local intranet

Information on the management interface of the MBean

AccountManager

FNBMainframe:name=AccountManager,cascaded=FNBMainframe

Attribute	Value
NumberOfAccounts	2
NumberOfCreditCards	1
NumberOfCurrentAccounts	1
NumberOfLoadedAccounts	0
ActiveCreditCards	

Operation	Parameters
unloadAccount Invoke	param0 : java.lang.Integer

The Process MBean

shows how the `IT_IIOPAdaptorServer` object is used to access the default MBean server. When the `IT_IIOPAdaptorServer` instance is created, the Administrator Web Console creates an entry in the navigation tree. This entry represents the *Process MBean*, the first-level MBean that is exposed. The

Process MBean is the starting point for navigation through an application in the Administrator Web Console.

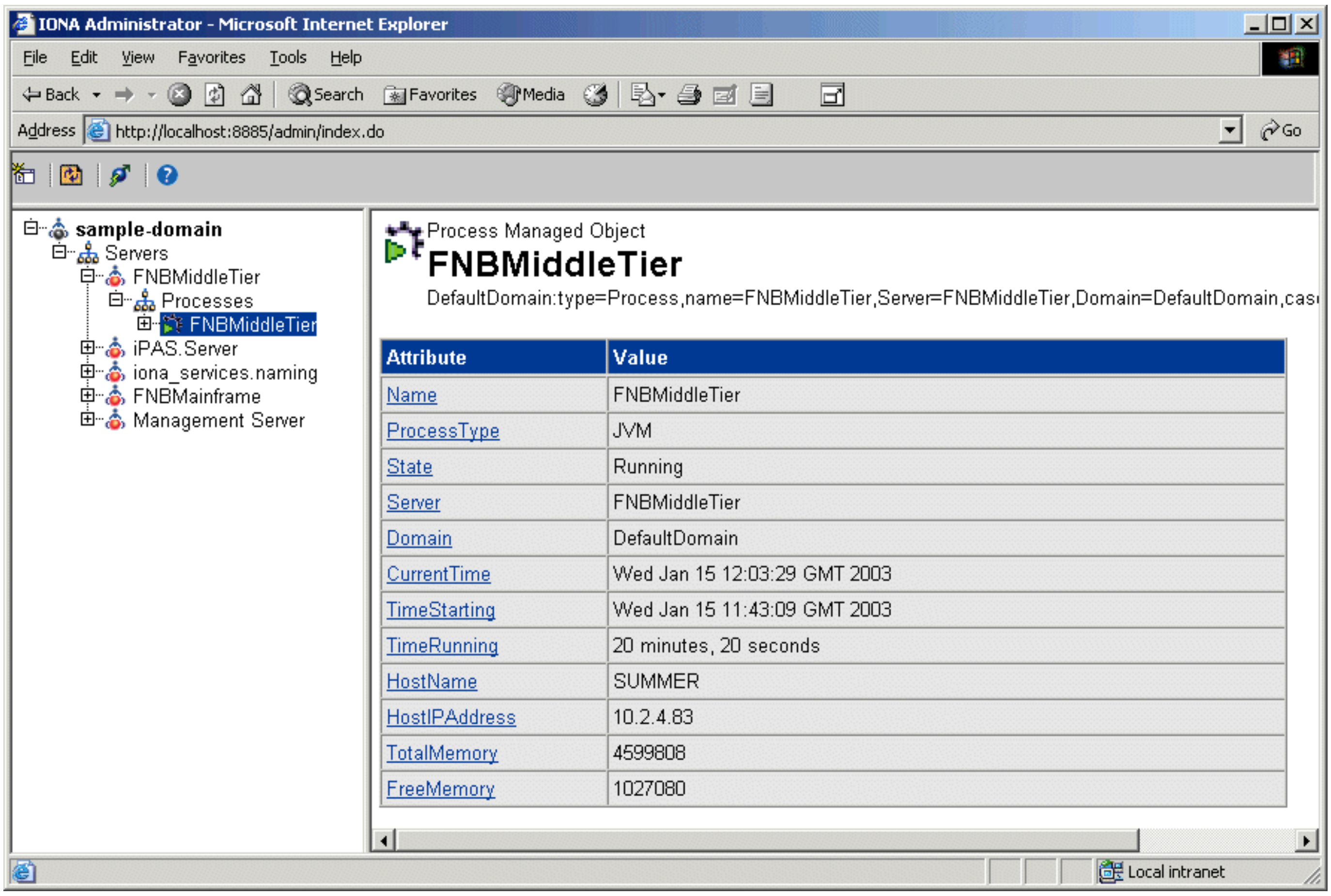
Example Process MBean

In Figure 8, the selected Process MBean in the navigation tree is `FNBMiddleTier`. The MBean's object name is displayed as:

```
DefaultDomain:type=Process,name=FNBMiddleTier,
Server=FNBMiddleTier,Domain=DefaultDomain,cascaded=FNBMiddleTier
```

The Process MBean has associated default attributes, displayed in the details panel (for example, process type, uptime, host, and so on).

Figure 8 Bank Process MBean


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
FNBMiddleTier
Processes
FNBMiddleTier
iPAS.Server
iona_services.naming
FNBMainframe
Management Server
Local intranet

Process Managed Object

FNBMiddleTier

DefaultDomain:type=Process,name=FNBMiddleTier,Server=FNBMiddleTier,Domain=DefaultDomain,cas

Attribute	Value
Name	FNBMiddleTier
ProcessType	JVM
State	Running
Server	FNBMiddleTier
Domain	DefaultDomain
CurrentTime	Wed Jan 15 12:03:29 GMT 2003
TimeStarting	Wed Jan 15 11:43:09 GMT 2003
TimeRunning	20 minutes, 20 seconds
HostName	SUMMER
HostIPAddress	10.2.4.83
TotalMemory	4599808
FreeMemory	1027080

Adding Application MBeans to the Tree

To display your application MBeans in the navigation tree of the Administrator Web Console, you must create a parent-child relationship between Process MBean and your application MBean.

To create parent-child relationships between your MBeans, use the `createParentChildRelation()` method. This section includes the following:

- Creating a parent-child relationship
- The createParentChildRelation() method

Creating a parent-child relationship

When create parent-child relationships your MBeans will be displayed as children of the Process MBean in the navigation tree, and as attributes in the details panel. Figure 8 shows the `FNBMiddleTier` Process MBean in the navigation tree, and its child MBeans listed details pane (for example, the `BusinessSessionManager` MBean).

The following code example shows how the `addMBean()` method is implemented in the `ManagementWrapper` class. This method calls the `createParentChildRelation()` method:

```
public boolean addMBean (java.lang.Object mbean, ObjectName mbeanName )
{
System.out.println ("Registering mbean...");
try {
ObjectName tmpArray [] = new ObjectName [1];
tmpArray [0] = mbeanName;
mBeanServer.registerMBean(mbean, mbeanName);
adaptorServer.createParentChildRelation(processMBean,tmpArray);
}
catch ( Exception j ) {
System.err.println ("Exception in registering MBean " + j );
return false;
}
return true;
}
```

The createParentChildRelation() method

The `createParentChildRelation()` method takes two parameters:

- The parent MBean `ObjectName` (in this case, the Process MBean).
- The child MBean `ObjectName` (in this case, an array of MBeans).

Note

MBeans must first be registered in order for them to appear when added to the Process MBean. For details of how to register MBeans, see Step 5—Registering your MBeans.

Customizing your Application MBean Icons

By default, when you add a new MBean, it is displayed using a default blue MBean icon. You can direct Administrator to use your own custom icons for your application MBeans.

The FNB example uses the default icons, and does not use custom icons. The examples in this section are taken from a demo application named iBank. The iBank example uses a bank icon to represent a `ManagediBank` MBean, and a cash icon to represent a `ManagediBankAccountMBean` MBean.

This section explains the following:

- Changing the admin.war file
- Changing the admin.war file
- Accessing your custom icons

Changing the admin.war file

You must first update the contents of the management web console by changing the `admin.war` file. The `admin.war` file can be found in the following directory:

<`install-dir`/asp/*version*/etc/admin/webapps

Note

You may want to make a backup copy of `admin.war` before removing it.

Under this directory, create a new directory called `admin`. Unjar `admin.war` into this directory, for example, using the following commands:

```
cd admin
jar xvf ../admin.war
rm ../admin.war
```

When you have changed the `admin.war` file you can then edit the `image_mapping.properties` file.

Updating your image mapping file

To use custom icons, you must update your `image_mapping.properties` file. This file is stored in your `resources` directory:

UNIX	**<`install-dir`>`/etc/opt/iona`/domains/my-domain/resources**
Windows	<`install-dir`>`\etc\`domains\my-domain\resources

For example, the `image_mapping.properties` file lists all the iBank MBeans; and for each MBean there are several entries. The following entries are for `Banking Servers` type, which contains the `ManagediBank` MBean:

```
# Type = BankingServer
examples.ejb.management.ibank.ManagediBank.small =
                                              resources/images/bank16.gif
examples.ejb.management.ibank.ManagediBank.large =
                                              resources/images/bank32.gif
examples.ejb.management.ibank.ManagediBank.text = "iBank"
BankingServer.small=bank16.gif
BankingServer.large=bank32.gif
BankingServer.text=Banking Server
BankingServer.type=Banking Servers
```

These entries specify the images for a small icon (16x16), a larger icon (32x32), the text displayed for the icon, and its type or group (**BankingServer**).

In the first three entries in this example, the first part of the property name denotes the classname of the MBean. For example, "`examples.ejb.management.ibank.ManagediBank`".

In the remaining entries, the first part of the property name denotes the type of the property (for example, BankingServer). This is the type in which the MBean is grouped and displayed in the console.

Accessing your custom icons

To access your new icons, simply copy them into your `resources/images` subdirectory.

When you are happy with the results you, may want to `jar` your `.war` file again. You can do this from within the `admin` directory, for example, using the following command:

```
jar cvf ../admin.war .
cd ..
rm -rf admin
```

You must clear out the classloading cache to see your changes take effect. You can do this by stopping the management service and removing the contents of the cache, for example, as follows:

```
rm -rf <install-dir>/var/mydomain/dbs/mgmt/cache/CJMP/*
```

CORBA C++ Management

Instrumenting CORBA C++ Applications

This section 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.

Note

The Administrator Web Console is no longer automatically installed. It is available as an optional component. To install, please download and extract the GUI components archive and follow the installation instructions.

The GUI components archive can be downloaded from our Software License & Download (SLD) site.

The GUI components have been separated from the product as they do not fully comply with the Section 508 and WCAG accessibility requirements and guidelines. It is intended that future product releases will include updated GUI components that comply with the relevant accessibility guidelines.

Step 1—Identifying Tasks to be Managed

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
```

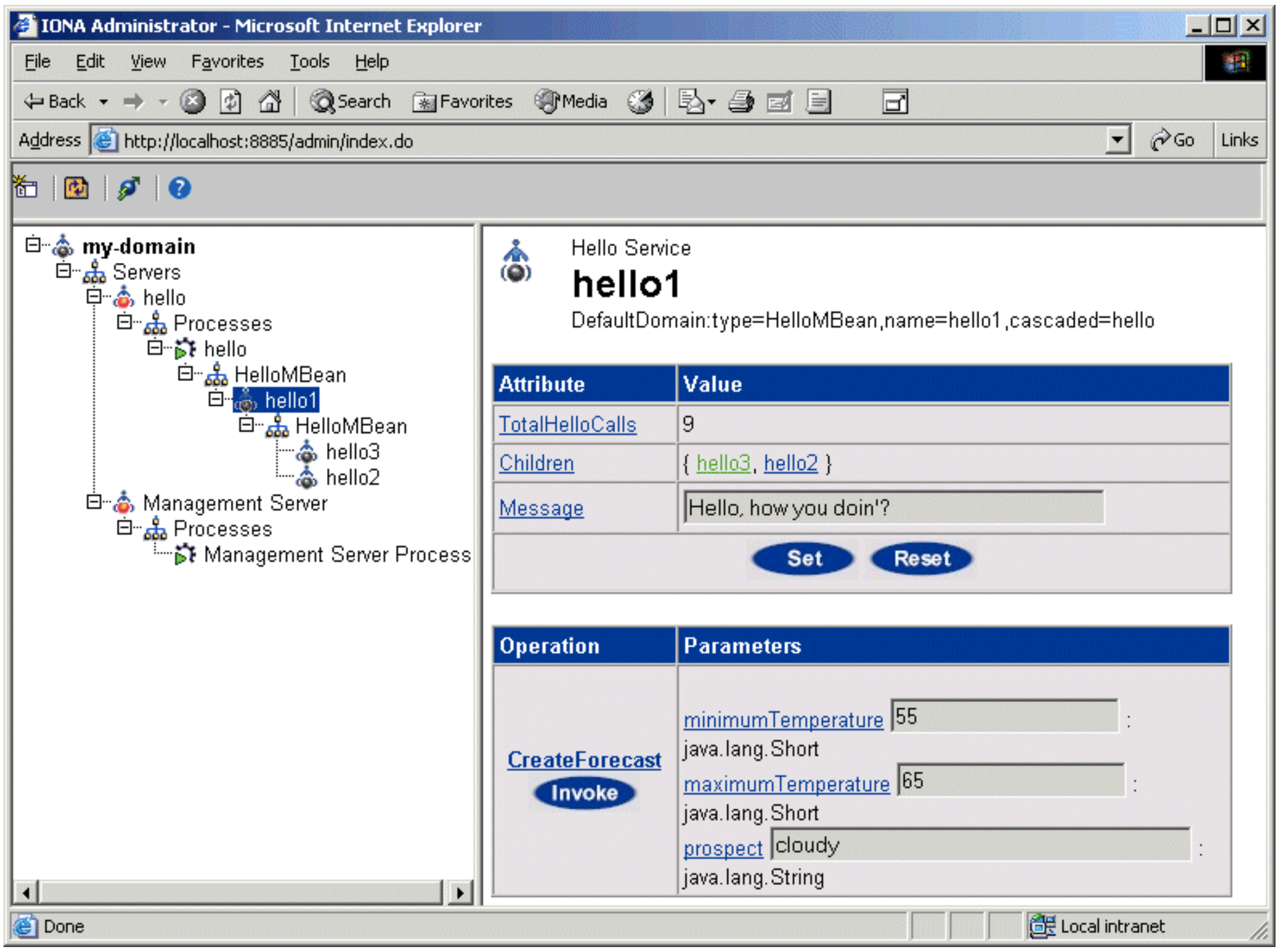
New management tasks

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 Console, shown in Figure 9.

Figure 9 Instrumented Plugin in Administrator


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

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

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

Set Reset

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

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

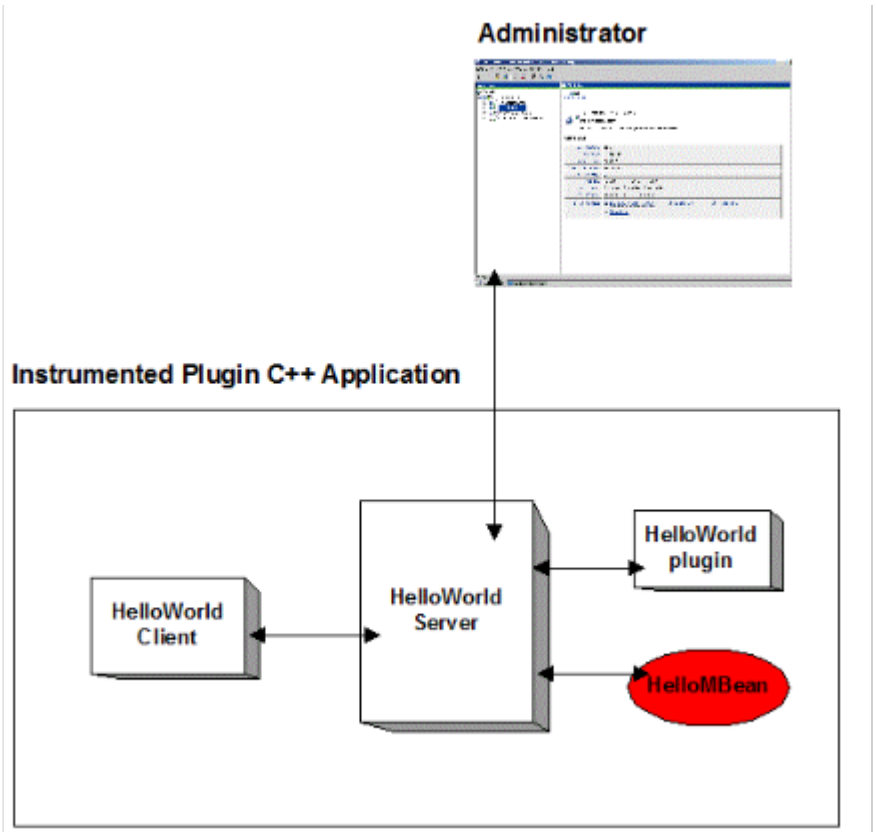
  For larger applications, it is advised that you to store your management code in a separate `management` directory. This will make your application more modular, and easier to understand.

Instrumented plugin overview

Figure 10 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 10 Instrumented Plugin Application Overview



Step 2—Defining your MBeans

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:

```
Example1Hello 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](#example-mbean-declaration)`	`public virtual IT_Mgmt::ManagedEntity,`
	`public virtual IT_CORBA::RefCountedLocalObject`
	`{`
	`public:`
	`HelloMBean (`
	`HelloWorldImpl * orb_info,`
	`const char * name`
	`);`
	`virtual ~HelloMBean();`

`[2](#example-mbean-declaration)`	`IT_Mgmt::ManagedEntityIdentifier`
	`managed_entity_id()`
	`IT_THROW_DECL((CORBA::SystemException));`

`[3](#example-mbean-declaration)`	`char* entity_type()`
	`IT_THROW_DECL((CORBA::SystemException));`

`[4](#example-mbean-declaration)`	`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](#example-mbean-declaration)`	`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:

```
Example2HelloMBean Private Declaration
```

	`private:`
`[1](#example-private-description)`	`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;`
	`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](#example-private-description)`	`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.

Further information

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

```
[http://www.oracle.com/technetwork/java/javase/tech/
javamanagement-140525.html](http://www.oracle.com/technetwork/java/javase/
tech/javamanagement-140525.html)
```

Step 3—Implementing your MBeans

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:

```
Example3MBean Constructor and Destructor
```

`[1](#write-the-mbean-constructor-and-destructor)`	`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";`

	`m_identity = "DefaultDomain";`
	`//m_identity = m_domain.c_str();`
	`m_identity +=`
	`":type=HelloMBean,name=";`
	`m_identity += name;`
	`initialize_attributes();`
	`initialize_operations();`
	`}`

`[2](#write-the-mbean-constructor-and-destructor)`	`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.
2. The HelloMBean() destructor. For information about unregistering MBeans as managed entities, see Removing your MBeans.

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:

```
Example4Managed Entity ID and Type
```

`[1](#get-the-managed-entity-id-and-entity-type)`	`IT_Mgmt::ManagedEntityIdentifier` `HelloMBean::managed_entity_id()` `IT_THROW_DECL((CORBA::SystemException))` `{` `return CORBA::string_dup(m_identity.c_str());` `}`
`[2](#get-the-managed-entity-id-and-entity-type)`	`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:

```
Example5Gettiing Managed Attributes
```

`[1](#get-the-managed-attributes)` `[\#p_d1862e1_d1863e1_d1864e1_d1865e1](#get-the-managed-attributes)`	`CORBA::Any*` `HelloMBean::get_mgmt_attribute(const char* key)` `IT_THROW_DECL((CORBA::SystemException,` `IT_Mgmt::AttributeUnknown))` `{`
`[2](#get-the-managed-attributes)` `[\#p_d1862e1_d1863e1_d1864e1_d1865e1](#get-the-managed-attributes)`	`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](#get-the-managed-attributes)`	`else if (strcmp(key, m_children_name) == 0)` `{` `IT_Locker<IT_Mutex> lock(m_mutex);` `HelloWorldImpl::HelloWorldList children =` `m_hello->get_children();`

```
CORBA::AnySeq
children_seq(children.size());
children_seq.length(children.size());
HelloWorldImpl::HelloWorldList::iterat
or 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. For detailed information about the CORBA::Any type, see the *Orbix CORBA Programmer's Guide* (C++ version).

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 9 on page 40, 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:

```
Example6Setting Managed Attributes
```

`[1](#set-the-managed-attributes)` `[get_the_managed_attributes_1609.md\#p_d1862e1_d1863e1_d1864e1_d1865e1](#get-the-managed-attributes)`	`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;`

`[2](#set-the-managed-attributes) [get_the_managed_attributes_1609.md\#p_d1862e1_d1863e1_d1864e1_d1865e1](#get-the-managed-attributes)`	`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 *Orbix CORBA Programmer's Guide* (C++ version).

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 9 on page 40, 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:

```
Example7Invoke Operations
```

`[1](#invoke-the-managed-operations)`	`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](#invoke-the-managed-`
`operations)`
	`validate_create_forecast_parameters(in_param`
	`eters);`
	`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);`
	`}`

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

`[3](#invoke-the-managed-`
`operations)`
	`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 *Orbix CORBA Programmer's Guide* (C++ version).

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 9 on page 40, 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:

```
Example8Throwing 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;
```

```
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

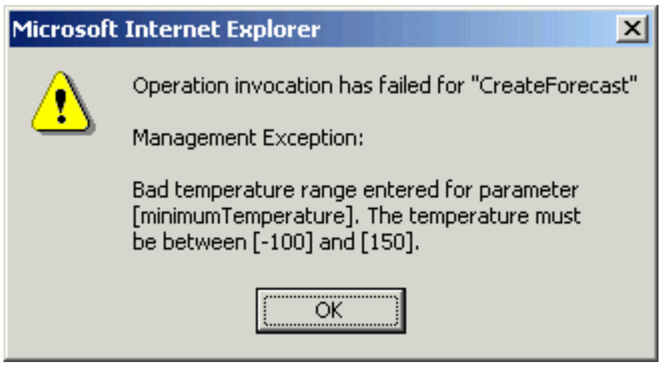
You can specify custom messages using the `exception-ia.properties` file, which is located in the following directory:

```
<install-dir>\e2a\etc\domains\sample-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 11 Instrumented Plugin Custom Exception



Operation invocation has failed for "CreateForecast"

Management Exception:

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

Get the MBean description

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

```
Example9Getting the MBean description
```

`[1](#get-the-mbean-description)`	`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](#get-the-mbean-description)

```
void HelloMBean::initialize_attributes()
{
m_hit_count_name = "TotalHelloCalls";
HelloAttribute total_hits =
{
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](#get-the-mbean-description)`	`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;` `}`
	`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 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 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 MBean Document Type Definition.

Step 4—Initializing the Management Plugin

After defining and implementing your MBeans, you should then initialize 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:

```
Example10Management Initialization
```

	`void`
	`HelloWorldImpl::initialize_management()`
	`IT_THROW_DECL(())`
	`{`

`[1](#example-management-initialization)`	`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](#example-management-initialization)`	`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.
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

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:

```
Example11Creating an MBean
```

	`void HelloWorldImpl::initialize_management()` `IT_THROW_DECL(())` `{` `.` `.` `.` `// Create and register the Hello MBean` `IT_Mgmt::ManagedEntity_var hello_mbean_ref;`
`[1](#creating-an-example-mbean)`	`hello_mbean_ref = m_hello_mbean_servant =` `new HelloMBean(this,m_name.in());` `instrument->new_entity(hello_mbean_ref);` `if (m_is_parent)` `{`
`[2](#creating-an-example-mbean)`	`//Get the Process ObjectName` `process_object_name = instrument->get_process_object_name();`
`[3](#creating-an-example-mbean)`	`// 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.

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

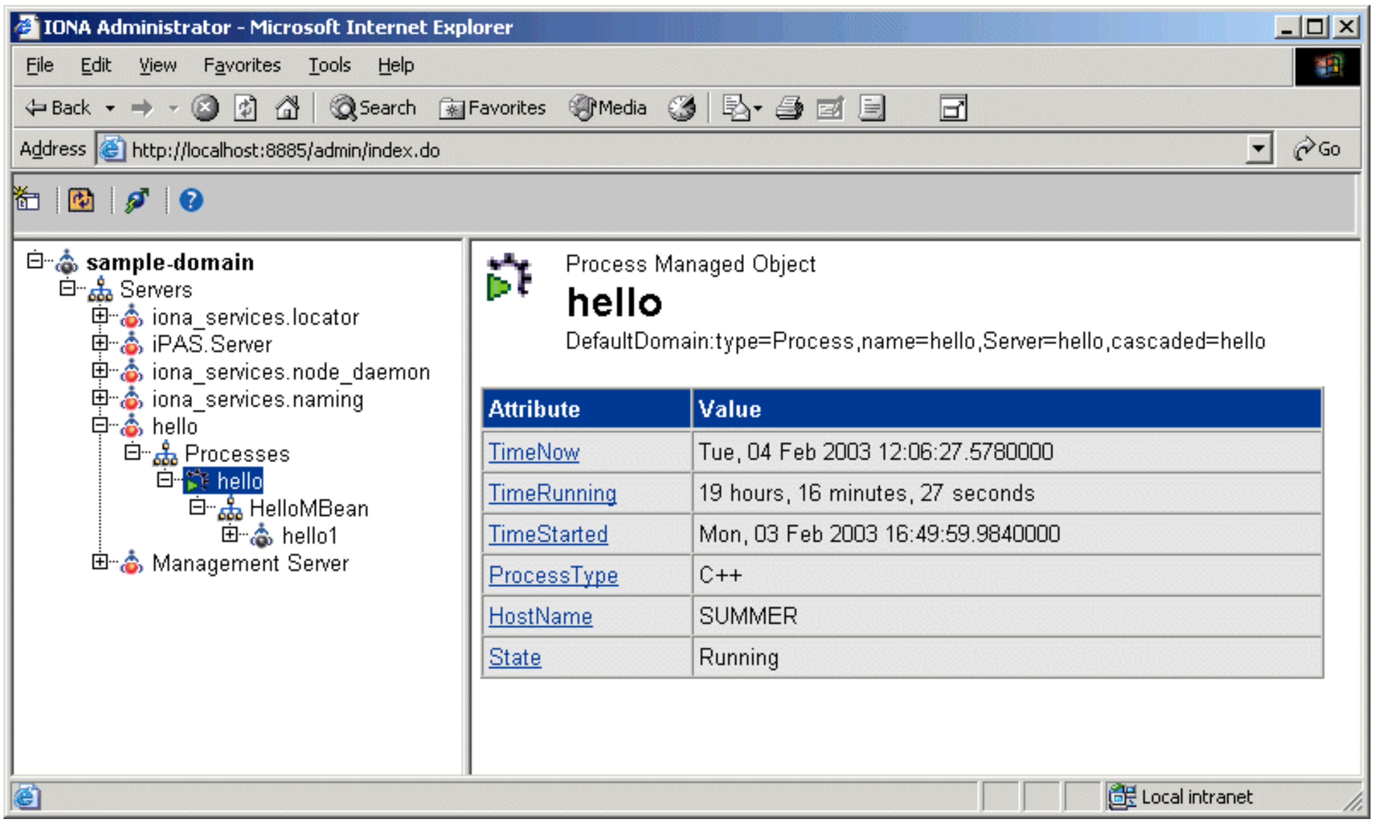
Applications are displayed in the Administrator Web Console as a series of related or connected MBeans, which can be monitored by administrators. This section explains how to connect your application MBeans together.

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 12 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 12 Instrumented Plugin Process MBean


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

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 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.

```
Example12Creating Child MBeans
```

```
void
HelloWorldImpl::add_child(HelloWorldImpl
*child)
IT_THROW_DECL(())
{
// Lock mutex
try
{
```

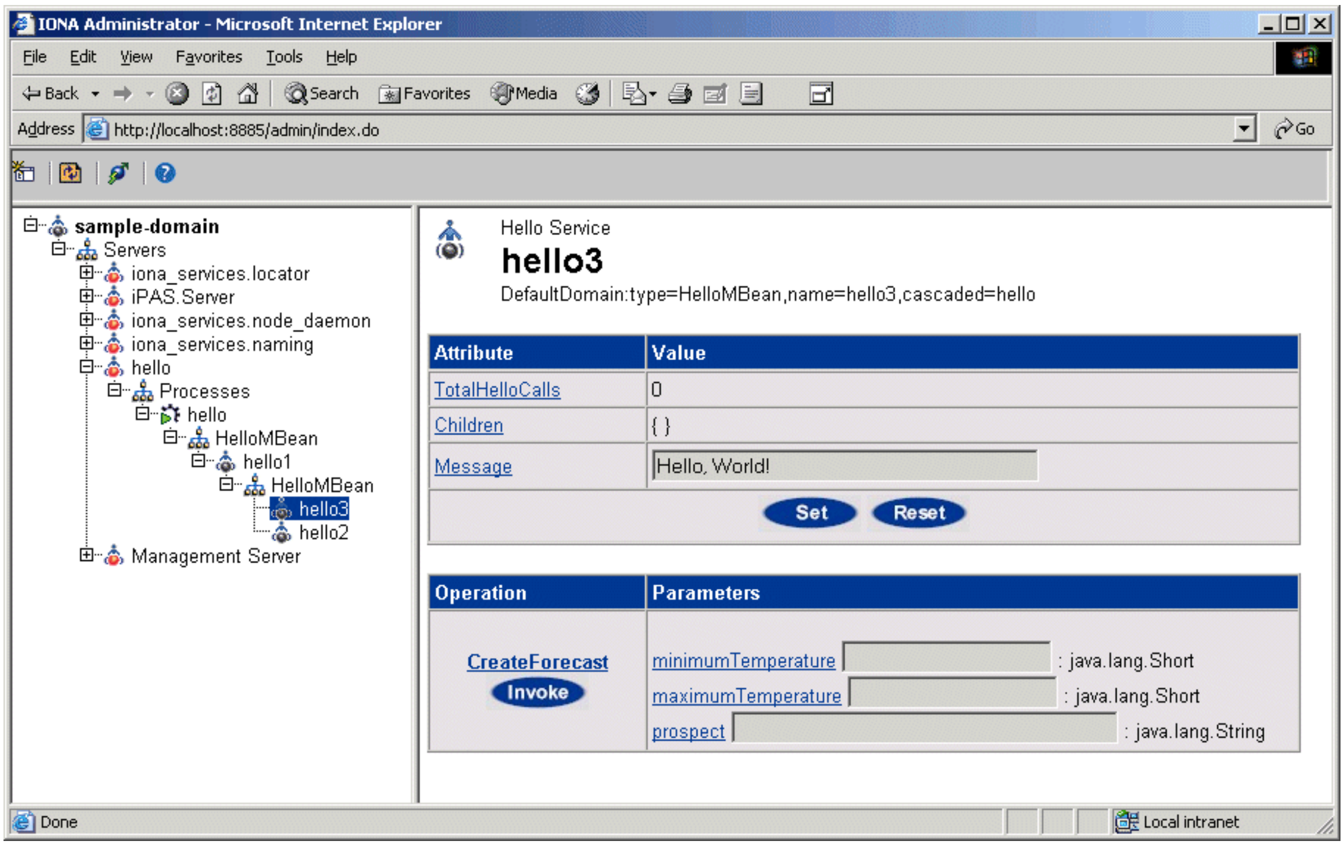
`[1](#creating-parent-child-relationships)`	`CORBA::Object_var obj;` `obj = m_orb->resolve_initial_references("IT_Instrumentation");` `IT_Mgmt::Instrumentation_var instrument;` `instrument = IT_Mgmt::Instrumentation::_narrow(obj);` `if (CORBA::is_nil(instrument))` `{` `throw IT_String("Instrumentation reference is nil");` `}` `CORBA::String_var my_name, child_name;`
`[2](#creating-parent-child-relationships)`	`my_name = m_hello_mbean_servant->managed_entity_id();` `IT_Mgmt::ManagedEntity_var childMBean = child->get_mbean();` `child_name = childMBean->managed_entity_id();`
`[3](#creating-parent-child-relationships)`	`instrument->create_parent_child_relationship(` `my_name.in(),` `child_name.in()` `);`
`[4](#creating-parent-child-relationships)`	`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.

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

   Figure 13 Instrumented Plugin Child MBean


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

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

Done Local intranet

Monitoring MBean Statistics

Optionally, you can also monitor statistics from MBeans in your own applications. The `it_mbean_monitoring` performance logging plug-in enables you to periodically harvest statistics associated with MBean attributes. This section includes the following:

- MBean monitoring
- Programming steps

MBean monitoring

The IT_MBeanMonitoring IDL interface provides the support for monitoring MBean statistics. This interface is defined as follows:

```
module IT_MBeanMonitoring
{
const string MANAGEMENT_MBEAN_MONITORING_INITIAL_REF =
"IT_MBeanMonitoringRegistration";
// Interface exceptions.
exception MBeanNotFound {};
exception MBeanAttributeNotFound {};
exception MBeanAttributeInvalidType {};
// IT_MBeanMonitoring::MBeanMonitoringRegistration
//
// An interface which provides a means to
// monitor and log statistics about mbeans
// registered with the management service.
local interface MBeanMonitoringRegistration
{
void monitor_attribute(
in string object_name,
in string attribute_name,
in string alias) raises ( MBeanNotFound,
MBeanAttributeNotFound, MBeanAttributeInvalidType);
void cancel_monitor(
in string object_name,
in string attribute_name,
in string alias) raises ( MBeanNotFound);
};
};
```

When the `it_mbean_monitoring` plug-in is included in your `orb_plugins` list, an initial reference is registered for the `IT_MBeanMonitoringRegistration` interface.

When you resolve on your application MBean, the IT_MBeanMonitoring API can be used to switch on, or turn off, monitoring of an application MBean. Statistics for user monitored MBeans will then appear in the performance logs.

Programming steps

This example assumes that you already have an MBean with an attribute that you want to be sampled and logged. For example, the MBean might track the memory currently being used by the process. The programming steps are as follows:

1. Include the following header files:

```
#include <orbix_pdk/mbean_monitoring_registration.hh>
```

2. To register your MBean with the `it_mbean_monitoring` plug-in, you must first resolve on the MBean monitoring initial reference:

```
try {
Object_var obj = orb->resolve_initial_references(
                IT_MBeanMonitoring::MANAGEMENT_MBEAN_MONITORING_INITIAL_REF
);
m_mbean_monitoring_registration =
MBeanMonitoringRegistration::_narrow(obj);
}
catch(const ORB::InvalidName&)
{
...
```

`}`

3. You can then register the attribute to be monitored by specifying your MBean details in a call to `monitor_attribute()`:

```
try {
m_mbean_monitoring_registration->monitor_attribute( "mbean_name",
"attribute_name", "mbean_friendly_name");
}
catch (...)
{
// do nothing.
}
```

The *mbean_friendly_name* is an alternative alias that will also appear in the log file.

Further information

For more details on Orbix performance logging, see the *Orbix Management User's Guide*.

MBean Document Type Definition

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

Note

The Administrator Web Console is no longer automatically installed. It is available as an optional component. To install, please download and extract the GUI components archive and follow the installation instructions .

The GUI components archive can be downloaded from our Software License & Download (SLD) site.

The GUI components have been separated from the product as they do not fully comply with the Section 508 and WCAG accessibility requirements and guidelines. It is intended that future product releases will include updated GUI components that comply with the relevant accessibility guidelines.

The MBean Document Type Definition File

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 are 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.

`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 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 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 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.

`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/

Notices

Copyright

Trademarks

Examples

License agreement

Corporate information

Rocket Software, Inc. develops enterprise infrastructure products in four key areas: storage, networks, and compliance; database servers and tools; business information and analytics; and application development, integration, and modernization.

Website: www.rocketsoftware.com

Contacting Technical Support

The Rocket Community is the primary method of obtaining support. If you have current support and maintenance agreements with Rocket Software, you can access the Rocket Community and report a problem, download an update, or read answers to FAQs. To log in to the Rocket Community or to request a Rocket Community account, go to www.rocketsoftware.com/support. In addition to using the Rocket Community to obtain support, you can use one of the telephone numbers that are listed above or send an email to support@rocketsoftware.com.

Rocket Global Headquarters
77 4th Avenue, Suite 100
Waltham, MA 02451-1468
USA

Country and Toll-free telephone number

To contact Rocket Software by telephone for any reason, including obtaining pre-sales information and technical support, use one of the following telephone numbers.

- United States: 1-855-577-4323
- Australia: 1-800-823-405
- Belgium: 0800-266-65
- Canada: 1-855-577-4323
- China: 400-120-9242
- France: 08-05-08-05-62
- Germany: 0800-180-0882
- Italy: 800-878-295
- Japan: 0800-170-5464
- Netherlands: 0-800-022-2961
- New Zealand: 0800-003210
- South Africa: 0-800-980-818
- United Kingdom: 0800-520-0439