Artix™

IBM Tivoli Integration Guide

Version 4.1, September 2006

Making Software Work Together™

Updated: September 22, 2006

Contents

List of Figures

Preface

What is covered in this book

IONA's products support integration with Enterprise Management Systems such as IBM Tivoli™, BMC Patrol™, CA WSDM™, and HP OpenView™. This book explains how to integrate Artix and Orbix with IBM Tivoli.

Who should read this book

This book is aimed at system administrators using IBM Tivoli to manage distributed enterprise environments, and developers writing distributed enterprise applications. Administrators do not require detailed knowledge of the technology that is used to create distributed enterprise applications.

This book assumes that you already have a good working knowledge of the IBM Tivoli Management Framework and IBM Tivoli Monitoring (formerly known as Distributed Monitoring).

Organization of this book

This book contains the following chapters:

- Chapter 1 introduces Enterprise Management Systems, and IONA's integration with IBM Tivoli.
- Chapter 2 describes how to configure your IONA product for integration with IBM Tivoli.

- Chapter 3 describes how to configure your IBM Tivoli environment for integration with IONA products.
- Chapter 4 describes how to extend your integration from a test environment into a production environment.
- Chapter 5 explains how to perform common tasks such as tracking server metrics or starting a server.
- Appendix A lists the contents of the IONA Tivoli resource model, describing its thresholds, events and parameters.

The Artix Library

The Artix documentation library is organized in the following sections:

- Getting Started
- Designing Artix Solutions
- Configuring and Managing Artix Solutions
- Using Artix Services
- Integrating Artix Solutions
- Integrating with Management Systems
- Reference
- Artix Orchestration

Getting Started

The books in this section provide you with a background for working with Artix. They describe many of the concepts and technologies used by Artix. They include:

- Release Notes contains release-specific information about Artix.
- Installation Guide describes the prerequisites for installing Artix and the procedures for installing Artix on supported systems.
- Getting Started with Artix describes basic Artix and WSDL concepts.
- Using Artix Designer describes how to use Artix Designer to build Artix solutions.
- Artix Technical Use Cases provides a number of step-by-step examples of building common Artix solutions.

Designing Artix Solutions

The books in this section go into greater depth about using Artix to solve real-world problems. They describe how to build service-oriented architectures with Artix and how Artix uses WSDL to define services:

- Building Service-Oriented Infrastructures with Artix provides an overview of service-oriented architectures and describes how they can be implemented using Artix.
- Writing Artix Contracts describes the components of an Artix contract. Special attention is paid to the WSDL extensions used to define Artix-specific payload formats and transports.

Developing Artix Solutions

The books in this section how to use the Artix APIs to build new services:

- Developing Artix Applications in C++ discusses the technical aspects of programming applications using the C++ API.
- Developing Advanced Artix Plug-ins in C++ discusses the technical aspects of implementing advanced plug-ins (for example, interceptors) using the C++ API.
- Developing Artix Applications in Java discusses the technical aspects of programming applications using the Java API.

Configuring and Managing Artix Solutions

This section includes:

- Configuring and Deploying Artix Solutions explains how to set up your Artix environment and how to configure and deploy Artix services.
- Managing Artix Solutions with JMX explains how to monitor and manage an Artix runtime using Java Management Extensions.

Using Artix Services

The books in this section describe how to use the services provided with Artix:

- Artix Router Guide explains how to integrate services using the Artix router.
- Artix Locator Guide explains how clients can find services using the Artix locator.
- Artix Session Manager Guide explains how to manage client sessions using the Artix session manager.
- Artix Transactions Guide, C++ explains how to enable Artix C++ applications to participate in transacted operations.

- Artix Transactions Guide, Java explains how to enable Artix Java applications to participate in transacted operations.
- Artix Security Guide explains how to use the security features in Artix.

Integrating Artix Solutions

The books in this section describe how to integrate Artix solutions with other middleware technologies.

- Artix for CORBA provides information on using Artix in a CORBA environment.
- Artix for J2EE provides information on using Artix to integrate with J2EE applications.

For details on integrating with Microsoft's .NET technology, see the documentation for Artix Connect.

Integrating with Management Systems

The books in this section describe how to integrate Artix solutions with a range of enterprise and SOA management systems. They include:

- IBM Tivoli Integration Guide explains how to integrate Artix with the IBM Tivoli enterprise management system.
- BMC Patrol Integration Guide explains how to integrate Artix with the BMC Patrol enterprise management system.
- CA-WSDM Integration Guide explains how to integrate Artix with the CA-WSDM SOA management system.
- AmberPoint Integration Guide explains how to integrate Artix with the AmberPoint SOA management system.

Reference

These books provide detailed reference information about specific Artix APIs, WSDL extensions, configuration variables, command-line tools, and terms. The reference documentation includes:

- Artix Command Line Reference
- Artix Configuration Reference
- Artix WSDL Extension Reference
- Artix Java API Reference
- Artix C++ API Reference
- Artix .NET API Reference
- Artix Glossary

Artix Orchestration

These books describe the Artix support for Business Process Execution Language (BPEL), which is available as an add-on to Artix. These books include:

- Artix Orchestration Release Notes
- Artix Orchestration Installation Guide
- Artix Orchestration Administration Console Help.

Getting the Latest Version

The latest updates to the Artix documentation can be found at http://www.iona.com/support/docs.

Compare the version dates on the web page for your product version with the date printed on the copyright page of the PDF edition of the book you are reading.

Searching the Artix Library

You can search the online documentation by using the **Search** box at the top right of the documentation home page:

http://www.iona.com/support/docs

To search a particular library version, browse to the required index page, and use the **Search** box at the top right, for example:

http://www.iona.com/support/docs/artix/4.0/index.xml

You can also search within a particular book. To search within a HTML version of a book, use the **Search** box at the top left of the page. To search within a PDF version of a book, in Adobe Acrobat, select **Edit**|**Find**, and enter your search text.

Artix Online Help

Artix Designer and Artix Orchestration Designer include comprehensive online help, providing:

- Step-by-step instructions on how to perform important tasks
- A full search feature
- Context-sensitive help for each screen

There are two ways that you can access the online help:

- Select **Help|Help Contents** from the menu bar. The help appears in the contents panel of the Eclipse help browser.
- Press **F1** for context-sensitive help.

In addition, there are a number of cheat sheets that guide you through the most important functionality in Artix Designer and Artix Orchestration Designer. To access these, select **Help|Cheat Sheets**.

Artix Glossary

The Artix Glossary is a comprehensive reference of Artix terms. It provides quick definitions of the main Artix components and concepts. All terms are defined in the context of the development and deployment of Web services using Artix.

Additional Resources

The IONA Knowledge Base (http://www.iona.com/support/knowledge_base/index.xml) contains helpful articles written by IONA experts about Artix and other products.

The IONA Update Center (http://www.iona.com/support/updates/index.xml) contains the latest releases and patches for IONA products.

If you need help with this or any other IONA product, go to IONA Online Support (http://www.iona.com/support/index.xml).

Comments, corrections, and suggestions on IONA documentation can be sent to docs-support@iona.com .

Document Conventions

Typographical conventions

This book uses the following typographical conventions:

`Fixed width`

Fixed 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 `IT_Bus::AnyType` class.

Constant width paragraphs represent code examples or information a system displays on the screen. For example:

```
#include <stdio.h>
```

Fixed width italic	Fixed width 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/`*`YourUserName`*
Italic	Italic words in normal text represent *emphasis* and introduce *new terms*.
Bold	Bold words in normal text represent graphical user interface components such as menu commands and dialog boxes. For example: the **User Preferences** dialog.

Keying Conventions

This book uses the following keying conventions:

No prompt	When a command's format is the same for multiple platforms, the command prompt is not shown.
`%`	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 MS-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.
`|`	In format and syntax descriptions, a vertical bar separates items in a list of choices enclosed in `{}` (braces). In graphical user interface descriptions, a vertical bar separates menu commands (for example, select **File**|**Open**).

CHAPTER 1

Integrating with IBM Tivoli™

This chapter introduces the integration of IONA products with the IBM Tivoli™ Enterprise Management System (EMS).

In this chapter

This chapter contains the following sections:

Introduction

Overview

IONA's products support integration with Enterprise Management Systems such as IBM Tivoli. This section includes the following topics:

- "The application life cycle".
- "Enterprise Management Systems".
- "IONA EMS integration".
- "IONA Tivoli integration tasks".
- "Integration overview".

The application life cycle

Most enterprise applications go through a rigorous development and testing process before they are put into production. When applications are in production, developers rarely expect to manage those applications. They usually move on to a new project while the day-to-day running of the applications are managed by a production team. In some cases, the applications are deployed in a data center that is owned by a third party, and the team that monitors the applications belong to a different organization.

Enterprise Management Systems

Different organizations have different approaches to managing their production environment, but most will have at least one *Enterprise Management System* (EMS).

For example, the main Enterprise Management Systems include IBM Tivoli, HP OpenView™, and BMC Patrol™. These systems are popular because they give a top-to-bottom view of every part of the IT infrastructure. This means that if an application fails because the `/tmp` directory fills up on a particular host, for example, the disk space is reported as the fundamental reason for the failure. The various application errors that arise are interpreted as symptoms of the underlying problem with disk space. This is much better than being swamped by an event storm of higher level failures that all originate from the same underlying problem. This is the fundamental strength of integrated management.

IONA EMS integration

IONA's Orbix and Artix products are designed to integrate with Enterprise Management Systems. IONA's common management instrumentation layer provides a base that can be used to integrate with any EMS.

In addition, IONA provides packaged integrations that provide out-of-the-box integration with major EMS products. This guide describes IONA's integration with the IBM Tivoli products.

IONA Tivoli integration tasks

The IONA Tivoli integration performs key enterprise management tasks (for example, posting an event if a server dies). This enables automated recovery actions to be taken.

The IONA Tivoli integration also tracks key server metrics (for example, number of invocations received; and average, maximum and minimum response times). Events can be generated when any of these parameters go out of bounds.

In addition, you can also perform an extensible set of actions on servers. The default actions are start, stop and restart.

Integration overview

In the IONA Tivoli integration, these key server performance metrics are logged by the IONA performance logging plugins. Log file interpreting utilities are then used to analyze the logged data. Figure 1 shows a simplified overview of the IONA Tivoli integration at work. In this example, a restart command is issued to an unresponsive server (for example, locator or naming service).

The IONA performance logging plugins collect data relating to server response times and log it periodically in the performance logs. The IONA Tivoli resource model executes periodically on each host and uses the IONA log file interpreter to collect and summarize the logged data. It compares the response times and other values against user defined thresholds. If these values exceed the threshold, an event is fired. This event can be used to trigger an option from the Tivoli task library to restart the unresponsive server.

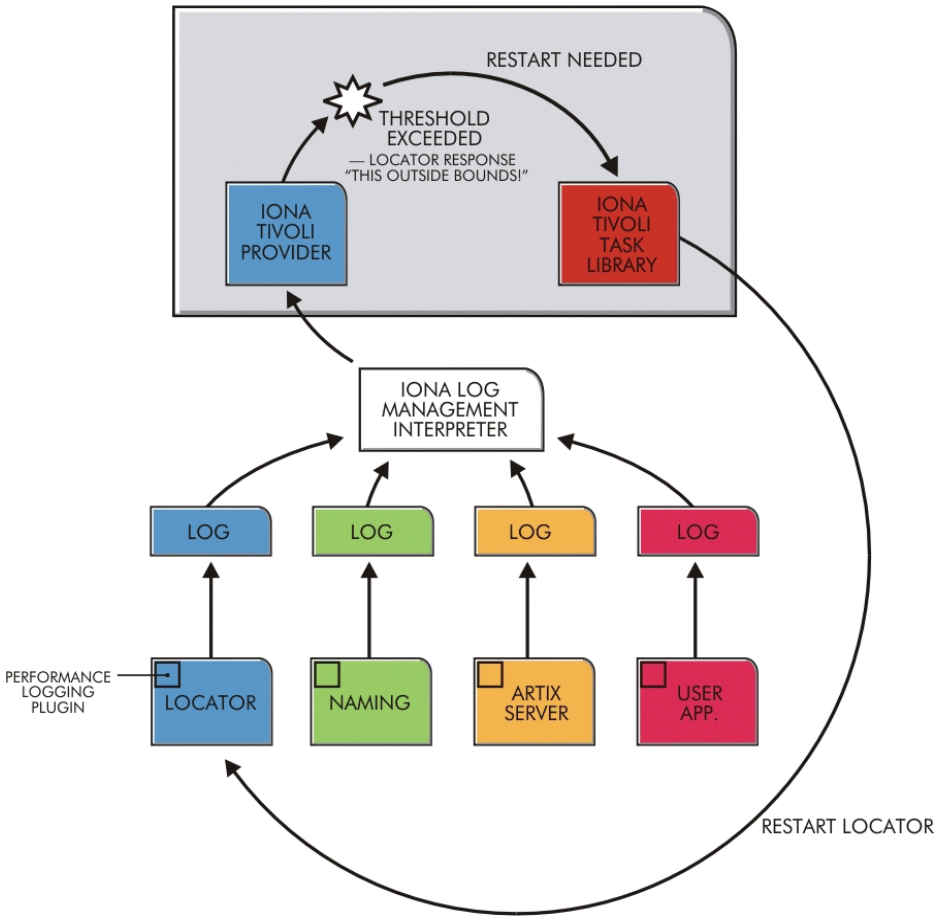

Figure 1: *Overview of the IONA Tivoli Integration*

The IONA Tivoli Integration

Overview

This section describes the requirements and main components of IONA's Tivoli integration. This section includes the following topics:

- "IONA requirements".
- "Tivoli requirements".
- "Main components".
- "IONA Tivoli resource model".
- "IONA Tivoli task library".
- "Integration and setup utilities".
- "Example IONA Tivoli deployment".

IONA requirements

IONA's Artix and Orbix products are fully integrated with IBM Tivoli. You must have at least one of the following installed:

- Artix 2.0.1 or higher.
- Orbix 6.1 or higher.

Tivoli requirements

IONA's products are fully integrated with IBM Tivoli Management Framework and IBM Tivoli Monitoring.

To use the IONA Tivoli integration, you must have at least the following versions installed:

- IBM Tivoli Management Framework 4.1 or higher.
- IBM Tivoli Monitoring 5.1.1 (Fix Pack 04) or higher.

Main components

The IONA Tivoli integration package contains three main parts:

- A Tivoli Monitoring resource model.
- A Tivoli task library.
- Integration and setup utilities.

IONA Tivoli resource model

For an introduction to Tivoli resource models, see the IBM *Tivoli Monitoring User Guide*. The IONA Tivoli resource model enables Tivoli to track key attributes of Artix and Orbix services and customer-built servers that are based on Artix and Orbix. These attributes include:

- Server liveness.
- Number of incoming invocations received by the server.
- Maximum, average, and minimum response times of the server.

The resource model defines events that fire when a server's liveness cannot be verified, or when any of the other attribute values go beyond thresholds that can be set by the user.

The IONA Tivoli resource model is described in detail in Appendix A.

IONA Tivoli task library

The IONA Tivoli task library contains a set of tasks that can be used to configure and check the IONA Tivoli integration.

This task library can also be used to start, stop, or restart monitored servers. It can also be extended to perform any number of actions on a monitored server. These actions can be performed automatically as a result of receiving an event. For example, if an event fires to indicate that a server is no longer alive, you can configure Tivoli to use the IONA Tivoli task library to issue a restart for that server.

Integration and setup utilities

Both the IONA Tivoli resource model and task library must be installed and configured to work correctly. The IONA Tivoli integration package contains a number of setup utilities that help you achieve this task. These utilities are described in detail in "Configuring your Tivoli Environment" on page 33.

Example IONA Tivoli deployment

The high-level overview in Figure 2 shows a typical deployment of an IONA Tivoli integration. This deployment is explained as follows:

1. The IONA Tivoli resource model and task library are installed on the Tivoli region server.
2. The administrator customizes a monitoring profile based on the IONA Tivoli resource model.
3. The monitoring profile is distributed through the gateways to each of the Tivoli endpoints (managed hosts). In this example, there are three Tivoli endpoints—two based on Windows, and one on Solaris.

4. The monitoring profile executes inside the Tivoli Monitoring Agent, periodically checking the status and response times of the IONA services and IONA-based applications.

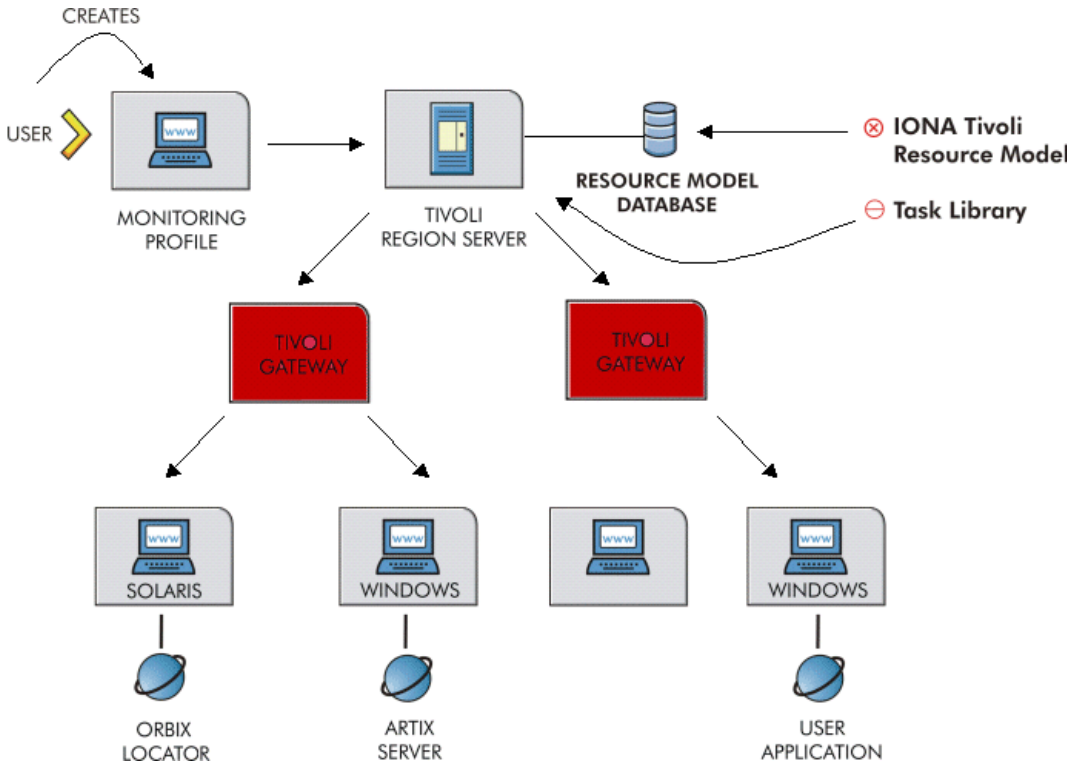

Figure 2: *Example IONA Tivoli Deployment*

These steps are explained in more detail in "Configuring your Tivoli Environment" on page 33 and "Extending to a Production Environment" on page 45.

CHAPTER 2

Configuring your IONA Product

This chapter explains the steps that you need to perform in Artix or Orbix so that they can be managed using IBM Tivoli.

In this chapter

This chapter contains the following sections:

Setting up your Artix Environment

Overview

The best way to learn how to use the IONA Tivoli integration is to start with a host that has both Tivoli and Artix installed. This section explains the configuration steps in your Artix environment. It includes the following topics:

Enabling management

You can use the **Artix Designer** GUI tool to enable management for your Artix applications. The **Artix Tools** dialog shown in Figure 3 allows you to do this.

Note: Before enabling management, you must have first generated a service plug-in (see Using Artix Designer).

To enable management, perform the following steps

1. Select **Artix Designer**|**Artix Tools**|**Artix Tools**.
2. In the **Artix Tools** window, create a new container deployment launch configuration.
3. In the **Advanced QoS Options** tab, select the **Enable reporting to a third-party management application** checkbox.
4. Enter an **Output file location**.

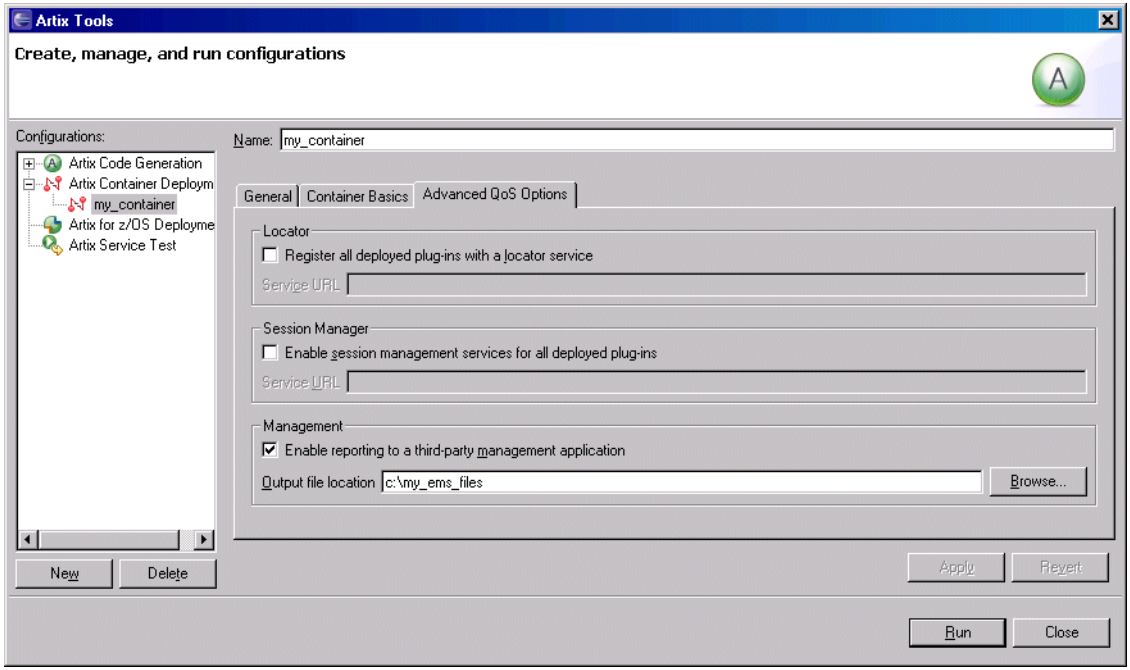

Figure 3: *Enabling Management in Artix Designer*

Generating EMS configuration files

When you click **Run**, this generate two files that are used to configure the IBM Tivoli integration:

- `servers.conf`
- `server_commands.txt`

These files are generated in the **Output file location** specified in Figure 3.

For more information on using Artix GUI tools, see Using Artix Designer.

The servers.conf file

When you open the `servers.conf` file, you will see an entry such as the following:

```
myapplication, 1, /Path/to/MyProject/log/myapplication_perf.log
```

This example entry instructs Tivoli to track the `myapplication` server. It reads performance data from the following log file:

`/Path/to/MyProject/log/myapplication_perf.log`

There will be one of these files for each application that you want to monitor. The IONA Tivoli resource model uses the `servers.conf` file to locate these logs and then scans the logs for information about the server's key performance indicators.

The server_commands.txt file

When you open the `server_commands.txt` file, you will see entries like the following:

```
myapplication,start=/Path/to/MyProject/bin/start_myapplication.sh
myapplication,stop=/Path/to/MyProject/bin/stop_myapplication.sh
myapplication,restart=/Path/to/MyProject/bin/restart_myapplication.sh
```

Each entry in this file references a script that can be used to stop, start or restart the `myapplication` server. For example, when the IONA Tivoli task library receives an instruction to start `myapplication`, it looks up the `server_commands.txt` file, and executes the script referenced in this entry.

Stopping Artix applications on Windows

On Windows, stop scripts are not generated by default. While it is straightforward to terminate a process on UNIX by sending it a kill signal, there is no straightforward equivalent on most Windows platforms.

On Windows XP, you can use the `taskkill` command in your stop scripts. On older versions of Windows, you can write your own stop scripts based on a variety of methods. There are many options for implementing a stop script including adding a Web service interface to control the shutdown of your server, or simply making use of a utility such as `pskill` from `www.sysinternals.com`.

See also, the following article on the Microsoft support pages:

How to terminate an application cleanly in Win32
(http://support.microsoft.com/default.aspx?scid=kb;EN-US;q178893)

Further information

For details of how to manually configure servers to use the performance logging, see "Configuring an Artix Production Environment" on page 46.

For a complete explanation of performance logging configuration, see Configuring and Deploying Artix Solutions.

Setting up your Orbix Environment

Overview

The best way to learn how to use the IONA Tivoli integration is to start with an Orbix installation on a host that is also a Tivoli endpoint. This section explains the configuration steps in your Orbix environment. It includes the following:

- "Creating an Orbix configuration domain".
- "Generating EMS configuration files".
- "Configuring performance logging".
- "Tivoli configuration files".
- "The servers.conf file".
- "The server_commands.txt file".
- "Further information".

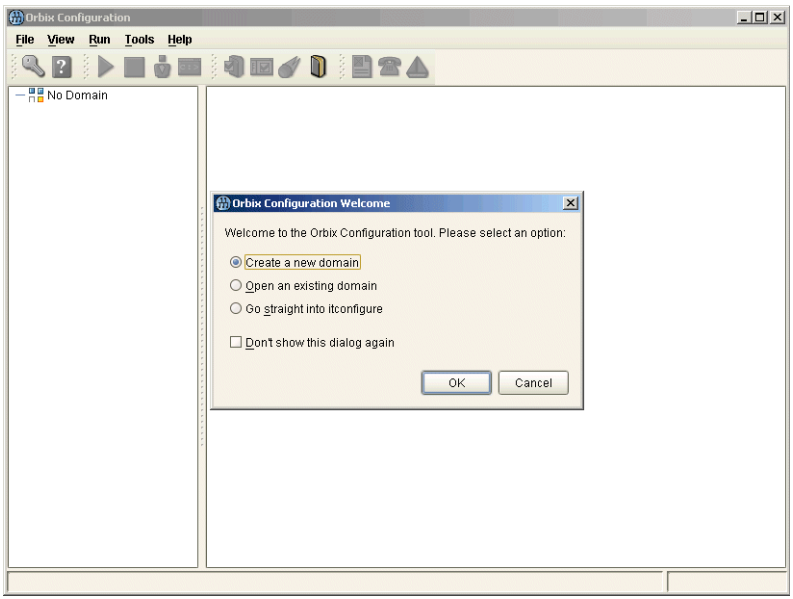

Figure 4: *Orbix Configuration GUI*

Creating an Orbix configuration domain

You must first create the Orbix configuration domain that you want to monitor using the **Orbix Configuration** GUI.

To start the **Orbix Configuration** GUI, enter `itconfigure` on the command line. The first screen is shown in Figure 4.

Generating EMS configuration files

To generate Tivoli agent configuration files, perform the following steps:

1. Click **Go straight into itconfigure** in the welcome dialog.
1. Select **File**|**New**|**Expert** from the GUI main menu. This displays the **Domain Details** screen, as shown in Figure 5.
2. Select the **Generate EMS Configuration Files** checkbox. This will generate configuration files required for your IONA Tivoli integration.

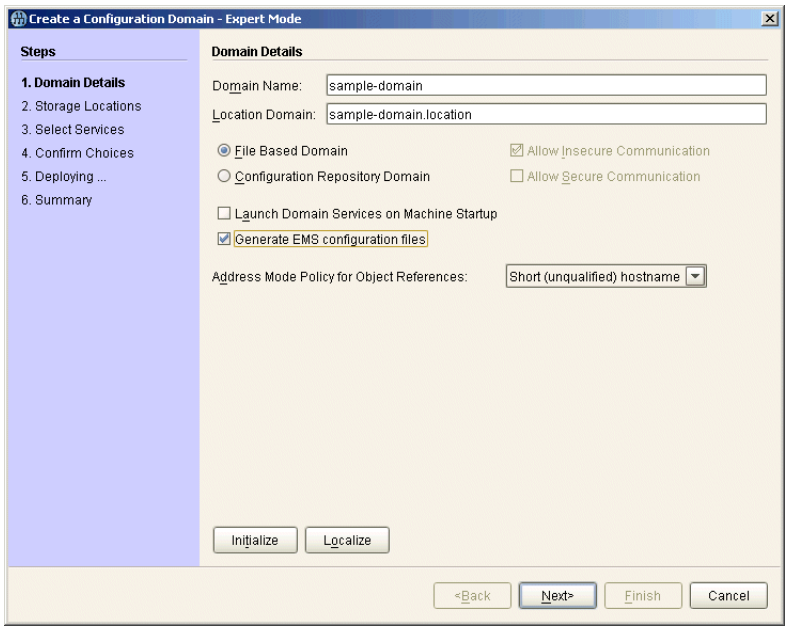

Figure 5: *Selecting Tivoli Agent Configuration*

3. Proceed as normal following the steps in the wizard until you get to the **Select Services** screen (see "Configuring performance logging").

Configuring performance logging

To configure performance logging, take the following steps:

1. In the **Select Services** screen, click **Settings** to launch the **Domain Defaults** dialog, shown in Figure 6.
2. Select the **Performance Logging** option in the **Other Properties** box, shown in Figure 6. This ensures that, by default, all your selected services are configured for monitoring.

   If you want to enable Tivoli to start, stop, or restart your servers, also select the **Launch Service on Domain Startup** option, in the **Service Launching** box.

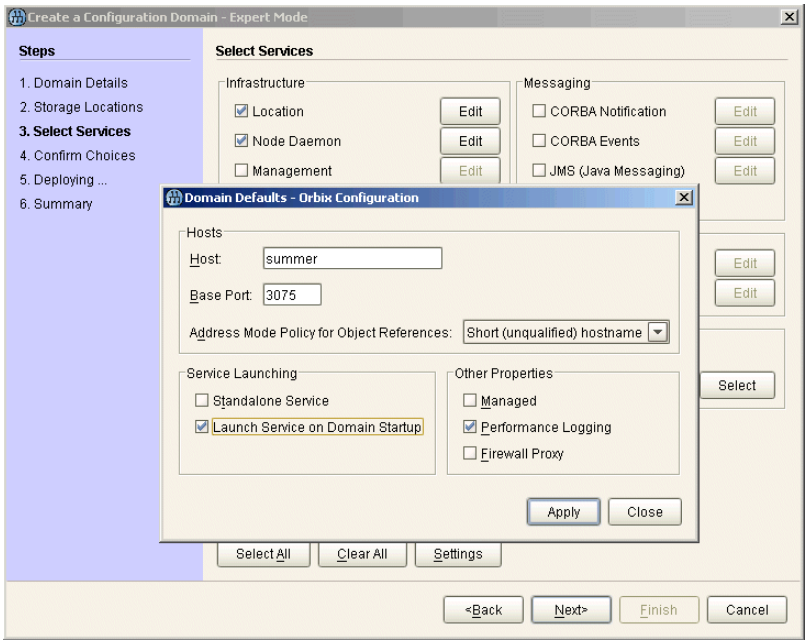

Figure 6: *Selecting Performance Logging*

Alternatively, you can configure these settings separately for each service by selecting the service, and clicking the **Edit** button.

3. Click **Apply**, and then **Close**.

4. Click **Next** to view a **Confirmation** screen for your selected configuration.
5. Click **Next** to deploy your configuration.
6. Click **Finish** to exit.

Note: When configuring Tivoli integration, you must also configure performance logging. This step is not optional. However, you can configure performance logging without Tivoli integration. For full details, see the *Orbix Management User's Guide.*

Tivoli configuration files

When the domain is created, you can start it like any other domain, using the start script in your *OrbixInstall*`/etc/bin` directory. Selecting the performance logging feature has enabled some extra configuration and logging. In your *OrbixInstall*`/var/domain-name` directory, you will find the following Tivoli configuration files:

`servers.conf`	Used by the IONA Tivoli resource model.
`server_commands.txt`	Used by the IONA Tivoli task library.

The servers.conf file

When you open the `servers.conf` file, you will see a number of entries in the following form:

ServerName`, `*number*`, /Path/to/a/Log/File`

For example:

```
mydomain_locator_myhost, 1,
   /opt/iona/var/mydomain/logs/locator_myhost_perf.log
```

The `servers.conf` file lists the servers that you want Tivoli to monitor on a particular host. To begin with, assume that you are running all services in the domain on one host. For example, assume your `servers.conf` file has the above entry. When you have started your domain, you should see a log file in the following location:

`/opt/iona/var/mydomain/logs/locator_perf.log`

There will be one of these files for each server that you want to monitor. The IONA Tivoli resource model uses the `servers.conf` file to locate these logs and then scans the logs for information about the server's key performance indicators.

The server_commands.txt file

When you open the `server_commands.txt` file, you will see a number of entries of the form:

`ServerName,Action=/Path/to/Script`

For example:

```
mydomain_locator_myhost,start
=/opt/iona/var/mydomain/locator_myhost_start.sh
```

Each entry in this file contains a pointer to a script that implements an action on a particular server. In this example, the action is a start action for the server `mydomain_locator_myhost`. When the IONA Tivoli task library receives an instruction to start the locator in a domain named `mydomain` on a host named `myhost`, it looks up the `server_commands.txt` file on `myhost`, and execute the script pointed to in this entry.

Further information

For details of how to manually configure servers to use the performance logging plugins, see "Extending to a Production Environment" on page 45.

For a complete explanation of performance logging configuration, see the *Orbix Management User's Guide*.

CHAPTER 3

Configuring your Tivoli Environment

This chapter explains the steps that you must perform in your IBM Tivoli environment. It assumes that you already have a good working knowledge of the IBM Tivoli Management Framework and IBM Tivoli Monitoring (formerly known as Distributed Monitoring).

In this chapter

This chapter contains the following sections:

Creating a Tivoli Installation Bundle

Overview

Your Tivoli integration comes in a `.tar` file called `tivoli_integration.tar`. This file is located in the following directory:

ArtixInstall\artix*Version*\management\Tivoli

This section explains how to create a Tivoli install bundle from the `tivoli_integration.tar` file. You will create an install bundle named `tivoli_install.tar`.

Creating an install bundle

To create a Tivoli install bundle, perform the following steps:

1. Untar the `tivoli_integration.tar` file into any directory on the host that you want to monitor, using the following command:

   ```
   tar xvf tivoli_integration.tar
   ```

   There should be three subdirectories:

   ```
   bin
   resource-model
   task-library
   ```

2. Go into the `bin` directory and run the `create_tivoli_install_bundle` shell script.

   Note: This is a bash script. On Windows (with Tivoli installed), you must use the bash environment that is installed with Tivoli. If you invoke the script with no arguments, it prints out a page of instructions.

The `create_tivoli_install_bundle` script takes the following arguments:

Configuration directory	The configuration directory where the `servers.conf` and `server_commands.txt` files are located: **Artix** The directory you specified when generating these files using **Artix Designer** (for example, `c:\my_ems_files`). **Orbix** *`OrbixInstall`*`/var/`*`DomainName`* **Note:** On Windows, you must use a forward slash character (`/`) when specifying this location.
Region name	The name of the Tivoli administrative region that you want this host/application to be in.
Profile manager	The name of the Tivoli profile manager that you want the IONA profile to be installed in.

3. Decide which region to use in your Tivoli deployment, and which profile manager you want the IONA profile to be installed in.
4. Run the `create_tivoli_install_bundle` shell script with all three values specified. This results in a new tar file called `tivoli_install.tar`.

Installing the Resource Model in the Tivoli Server

Overview

This section explains how to install the IONA Tivoli resource model from the `tivoli_install.tar` file that you created.

Installing the IONA Tivoli resource model

To install the IONA Tivoli resource model and task library into your Tivoli server, perform the following steps:

1. Transfer the `tivoli_install.tar` file to your Tivoli region server, and untar it to a temporary location, using the following command:

```
tar xvf tivoli_install.tar
```

2. Start a Tivoli shell environment (see your Tivoli documentation for details). On Windows, type `bash`, to run in a bash shell. Change to your temporary location, and you will see a new directory structure starting with a directory named `iona`.
3. Change directory into `iona/bin`. This contains the following shell scripts:

```
import_tll.sh
create_profile.sh
```

4. Run the `create_profile.sh` script. This adds the IONA Tivoli resource model to the resource model database and creates a new profile named `IONAProfile`.
5. Open the Tivoli **Desktop** and select the region that you specified when you created the install bundle, followed by the profile manager that you specified. In the **Profile Manager** GUI, you will see a new profile called `IONAProfile`, as shown in Figure 7.

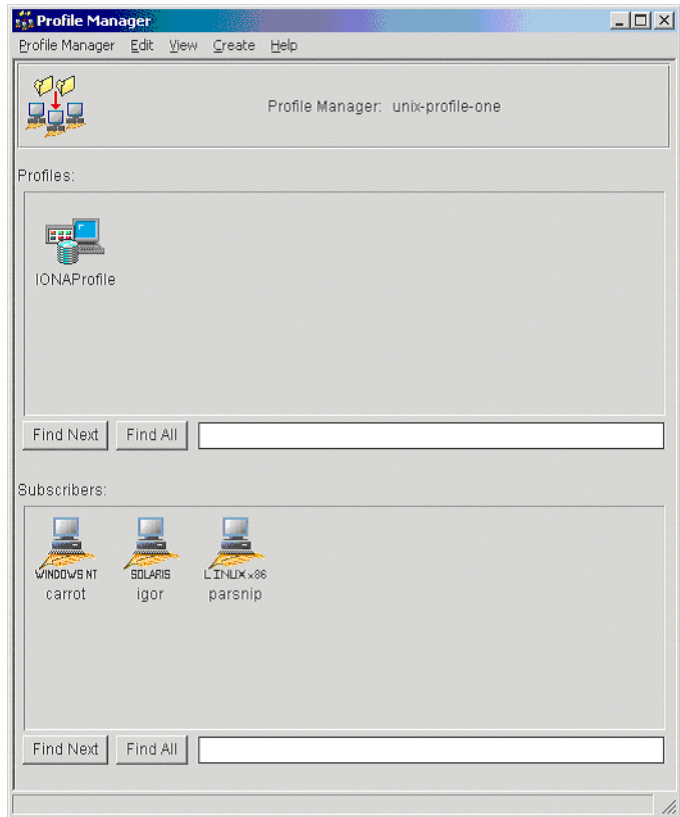

Figure 7: *Tivoli Profile Manager*

6. Open the `IONAProfile`, and then open the resource model `IONAServer Monitor`. You will see a resource model with default thresholds and indications, as shown in Figure 8.

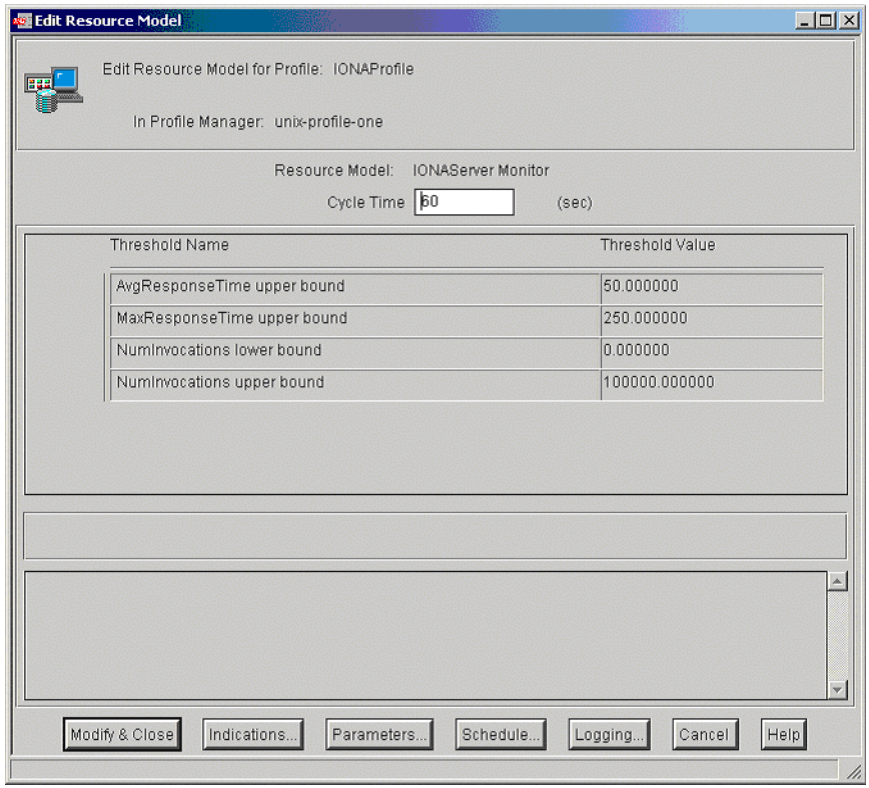

Figure 8: *Edit Resource Model*

Figure 8 shows that the profile has been initialized with default threshold values. Appendix A describes these thresholds in detail; you do not need to be concerned with these now.

7. If you want Tivoli to log historical data on the attributes of each server, click the **Logging...** button for the profile, and then check the box marked **Enable Data Logging** to put logging into effect. This will record historical data for each attribute.
8. Click **Modify & Close**.

Pushing the Resource Model out to your Host

Overview

This section explains how to push the IONA Tivoli resource model out to the endpoint where your IONA product is running (Orbix or Artix).

Pushing out the resource model

To push the IONA Tivoli resource model out to the endpoint where your IONA product is running, perform the following steps:

1. Add the Tivoli endpoint where your IONA product is installed as a subscriber to the profile manager, and distribute the `IONAProfile` to this endpoint.

   The resource model should now be running on the endpoint, but it will not yet be able to collect any meaningful data because it needs to be pointed to the `servers.conf` file.
2. Return to the directory where you untarred `tivoli_install.tar`, and change directory to `iona/bin`.
3. Run the `import_tll.sh` script. This installs the task library.
4. Reopen the Tivoli region that you are using on the desktop. You should now see a task library called `IONAServerTaskLibrary`.
5. Open the task library. It contains the following four tasks (also shown in Figure 9):

```
check_deployment
configure_provider
list_server_commands
server_command
```

6. Run the `check_deployment` task on the endpoint that contains your correctly configured Artix or Orbix installation. It prints out diagnostics to indicate that it has found your `servers.conf` file and your `server_commands.txt` file. This task also verifies the contents of these files.

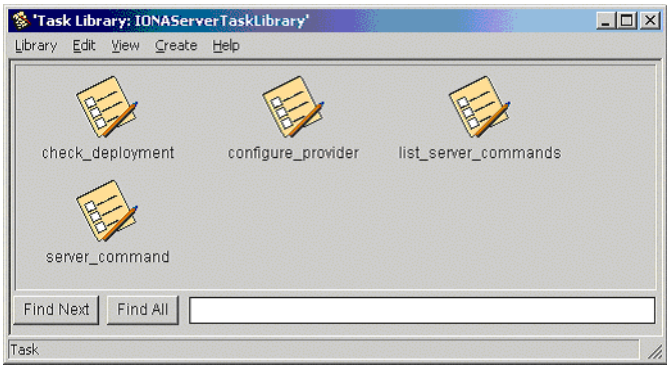

Figure 9: *Contents of the IONA Server Task Library*

7. If the `check_deployment` task runs successfully, try running `list_server_commands`. This shows a list of actions that you can run on each server, for example:

```
mydomain_locator_myhost,stop
```

Executing this command stops the Orbix locator in `mydomain`. You can execute any of these server commands by running the `server_command` task. This is an exercise for later (described in "Configuring the resource model" on page 42).

Configuring the Resource Model for your Endpoint

Overview

This section explains how to configure the IONA Tivoli resource model for your endpoint host, and how to test that your integration is configured correctly. It includes the following topics:

- "Configuring the resource model".
- "Testing your Tivoli integration".
- "Further information".

Configuring the resource model

To configure the resource model for your endpoint, perform the following steps:

1. First, verify that the `configure_provider` script uses the correct location for the `servers.conf` file, and then execute the `configure_provider` task.
2. You must restart the Tivoli Monitoring Engine on the endpoint to pick up this new information. You can do this using the following command:

```
wdmcmd -e endpoint_name -stop
```

> **Note:** On Windows, you might need to allow some time before restarting. This is because it takes time to shut down the Tivoli M12 provider, which hosts the resource model.

3. When this process has finished, it is safe to execute a restart using the following command:

```
wdmcmd -e endpoint_name -restart
```

4. View the status of your deployed resource model by opening the Tivoli **Web Health Console**, and view the data for your host. If all the servers in your domain are running, everything should be fine with no errors.

5. Verify that monitoring is working correctly by killing one of your servers. The effect is not immediately visible on the **Web Health Console**. The delay depends on cycle time setting in the profile. The default is `60` seconds. However, the **Web Health Console** can take longer to refresh.

   The quickest way to check the status is by executing the following command:

```
wdmlseng -e endpoint_name -verbose
```

   This shows the status of any errors in the deployed resource models.

6. You should be able to start the server again using the task library. Go to the task library and execute the `server_command` task. Fill in the name of the server and the action to perform on it (for example, in this case, `mydomain_locator_myhost, start`). You should see your server start and your health return to 100% in the **Web Health Console** soon after that.

Testing your Tivoli integration

When you have checked that you can start and stop servers and monitor their liveness, you can try some of the other available thresholds.

For example, the IONA Tivoli resource model provides a threshold called **NumInvocations upper bound**. This emits an event when the number of operations that a server receives exceeds a certain threshold, which can indicate that an overload is in progress. You can set the threshold, redistributing the profile. You can test this by writing clients to frequently contact the server in question until the threshold is exceeded.

Further information

For full details on how to use the Tivoli Monitoring product, see the *Tivoli Monitoring User Guide*.

CHAPTER 4

Extending to a Production Environment

This section describes how to extend an IONA TivoIi integration from a test environment to a production environment.

In this chapter

This chapter contains the following sections:

Configuring an Artix Production Environment

Overview

When you have performed the basic setup steps, then you can move on to the deployment-based production tasks. These include:

- "Monitoring your own Artix applications".
- "Monitoring an Artix application on multiple hosts".
- "Monitoring multiple Artix applications on the same host".
- "Further information".

Monitoring your own Artix applications

Using the **Artix Designer** GUI to enable Tivoli to manage your applications is straightforward. **Artix Designer** generates all the correct configuration for you. For details, see "Setting up your Artix Environment" on page 24.

Manual configuration

If you do not use **Artix Designer**, you must add the following settings to your Artix server's configuration file:

```
my_application {

# Ensure that bus_response_monitor is in your orb_plugins list.
orb_plugins = [ ...,"bus_response_monitor"];

# Collector period (in seconds). How often performance information is logged.
plugins:it_response_time_collector:period = "60";

# Set the name of the file which holds the performance log
plugins:it_response_time_collector:filename =
"/opt/myapplication/log/myapplication_perf.log";

}
```

Note: The specified `plugins:it_response_time_collector:period` should divide evenly into your cycle time (for example, a `period` of `20` and a cycle time of `60`).

Monitoring an Artix application on multiple hosts

The same principles apply when monitoring your server on multiple hosts. Each host has one `servers.conf` file. In the following example, assume that you want to run the `prdserver` on an endpoint host called `dublin`:

1. Create the `servers.conf` and `server_commands.txt` files for the servers that you want to monitor on the `dublin` host. You can write these files manually or use **Artix Designer** (see "Setting up your Artix Environment" on page 24 for details).
2. Run the `configure_provider` task selecting the `dublin` endpoint. Enter the location of the `servers.conf` file on the `dublin` host, shown in Figure 10.
3. Restart the Tivoli Monitoring Engine on `dublin` as described in "Configuring the Resource Model for your Endpoint" on page 42.

Now you should be able to monitor `prdserver` on `dublin`.

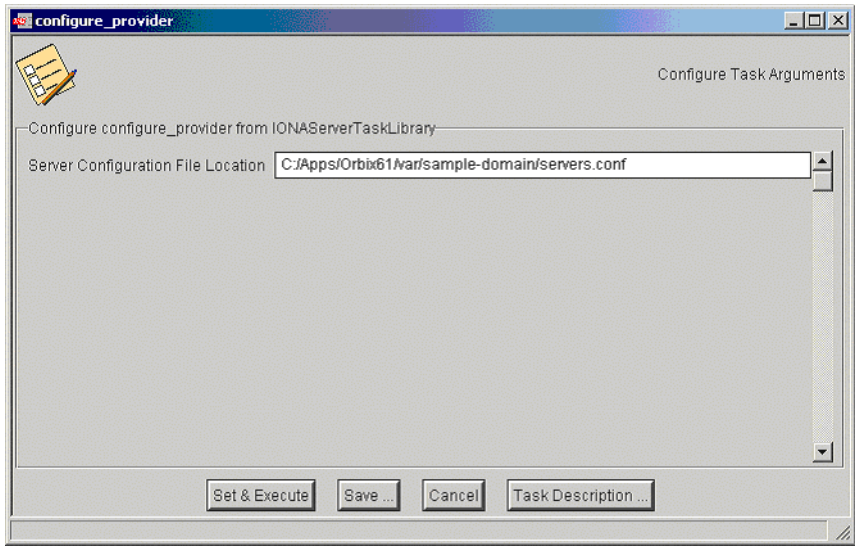

Figure 10: *The configure_provider Task*

Example task Suppose you want to execute the stop script for the `prdserver` on the `dublin` endpoint. Assuming your `server_commands.txt` file is complete, you can open the `server_command` task selecting the `dublin` endpoint. The script takes the following parameters:

```
Server Command File Location, (location of server_commands.txt)
Server Name,
Server Action,
Server Id
```

The `Server Name` and `Server Id` is `myapplication_prdserver`. The action is `stop`, but the `Server Command File Location` defaults to whatever this location was on the host where you first generated the `tivoli_install.tar`.

You must retype this location so that it points to the correct location on the `dublin` host. Use the same path for your `servers.conf` and `server_commands.txt` files on all hosts, if possible. If not, enter a new location each time that you want to invoke an action on a different host.

Alternatively, you can use the `server_command` task as a template for a new task. After changing the value of `server_commands.txt`, and filling in the other fields, instead of clicking **Set & Execute**, click **Save...**. You can rename this task as, for example, `stop_prdserver_on_dublin`.

If you want more flexibility in deciding which parameters to default and which to leave open, you can create a custom task library based on the IONA Tivoli task library. A description of how this is done is beyond the scope of this document. If you need to do this, contact IONA Professional Services.

Monitoring multiple Artix applications on the same host

Sometimes you may need to deploy multiple separate Artix applications on the same host. However, the **Artix Designer** only generates a `servers.conf` and `server_commands.txt` file for a single application.

The solution is to merge the `servers.conf` and `server_commands.txt` files from each of the applications into single `servers.conf` and `server_commands.txt` files.

For example, if the `servers.conf` file from the `UnderwriterCalc` application looks as follows:

```
UnderwriterCalc,1,/opt/myAppUnderwritierCalc/log/UnderwriterCalc_perf.log
```

And the `servers.conf` file for the `ManagePolicy` application looks as follows:

```
ManagePolicy, 1, /opt/ManagePolicyApp/log/ManagePolicy_perf.log
```

The merged `servers.conf` file will then include the following two lines:

```
UnderwriterCalc,1,/opt/myAppUnderwritierCalc/log/UnderwriterCalc_perf.log
ManagePolicy, 1, /opt/ManagePolicyApp/log/ManagePolicy_perf.log
```

Exactly the same procedure applies to the `server_commands.txt` file.

Further information

For full details on how to use the Tivoli Monitoring product, see your *Tivoli Monitoring User Guide.*

Configuring an Orbix Production Environment

Overview

When you have performed the basic setup steps, then you can move on to the deployment-based production tasks. These include:

- "Monitoring your own Orbix applications".
- "Monitoring your Orbix servers on multiple hosts".
- "Monitor multiple Orbix domains on the same host".
- "Further information".

Monitoring your own Orbix applications

You can use the **Orbix Configuration** tool to enable Tivoli management of Orbix services. Enabling Tivoli to manage your own Orbix applications involves the following steps:

1. You must configure your application to use performance logging (see the *Orbix Management User's Guide* for a full description). For example, suppose you have a server executable named `myapplication_prdserver` that executes with the ORB name `myapplication.prdserver`. The typical configuration would be as follows:

C++ applications

```
myapplication {
  prdserver {
     binding:server_binding_list = ["it_response_time_logger+OTS", ""];
     plugins:it_response_time_collector:period = "30";
     plugins:it_response_time_collector:server-id
        ="myapplication_prdserver";
     plugins:it_response_time_collector:filename =
        "/opt/myapplication/logs/prdserver/prdserver_perf.log";
     }
}
```

Java applications

```
myapplication {
  prdserver {
    binding:server_binding_list = ["it_response_time_logger+OTS", "..."];
    plugins:it_response_time_collector:period = "30";
    plugins:it_response_time_collector:server-id = "myapplication_prdserver";
    plugins:it_response_time_collector:log_properties = ["log4j.rootCategory=INFO, A1",
       "log4j.appender.A1=com.iona.management.logging.log4jappender.TimeBasedRollingFile
       Appender",
    "log4j.appender.A1.File=/opt/myapplications/logs/prdserver_perf.log",
    "log4j.appender.A1.layout=org.apache.log4j.PatternLayout",
    "log4j.appender.A1.layout.ConversionPattern=%d{ISO8601} %-80m %n"];
    }
}
```

Note: The specified `plugins:it_response_time_collector:period` should divide evenly into your cycle time (for example, a `period` of `20` and a cycle time of `60`).

2. The most important configuration values are the `server-id` and the C++ `filename` or Java `log_properties` used by the `response_time_collector`. You can add these values to the `servers.conf` file to make the IONA Tivoli resource model aware of your application as follows:

```
myapplication_prdserver, 1,
   /opt/myapplication/logs/prdserver/prdserver_perf.log
```

3. Restart your endpoint. Now Tivoli will monitor the execution of the `myapplication_prdserver`.
4. To control the `myapplication_prdserver` server through the `server_command` task, edit the `server_commands.txt` file. For example you could add the following entries to the `server_commands.txt` file:

```
myapplication_prdserver,start =
   /opt/myapplication/scripts/prdserver_start.sh
myapplication_prdserver,stop =
   /opt/myapplication/scripts/prdserver_stop.sh
myapplication_prdserver,restart =
   /opt/myapplication/scripts/prdserver_restart.sh
```

The `prdserver_start.sh`, `prdserver_stop.sh` and `prdserver_restart.sh` scripts will be written by you.

Monitoring your Orbix servers on multiple hosts

The same principles apply when monitoring your Orbix servers on multiple hosts. Each host has one `servers.conf` file. In the following example, assume that you want to run the `prdserver` on an endpoint host called `dublin`:

1. Write the `servers.conf` and `server_commands.txt` files for the servers that you want to monitor on the `dublin` host (see "Setting up your Orbix Environment" on page 28 for details).
2. Run the `configure_provider` task selecting the `dublin` endpoint. Enter the location of the `servers.conf` file on the `dublin` host, shown in Figure 10.
3. Restart the Monitoring Engine on `dublin` as described in "Configuring the Resource Model for your Endpoint" on page 42.

Now you should be able to monitor `prdserver` on `dublin`.

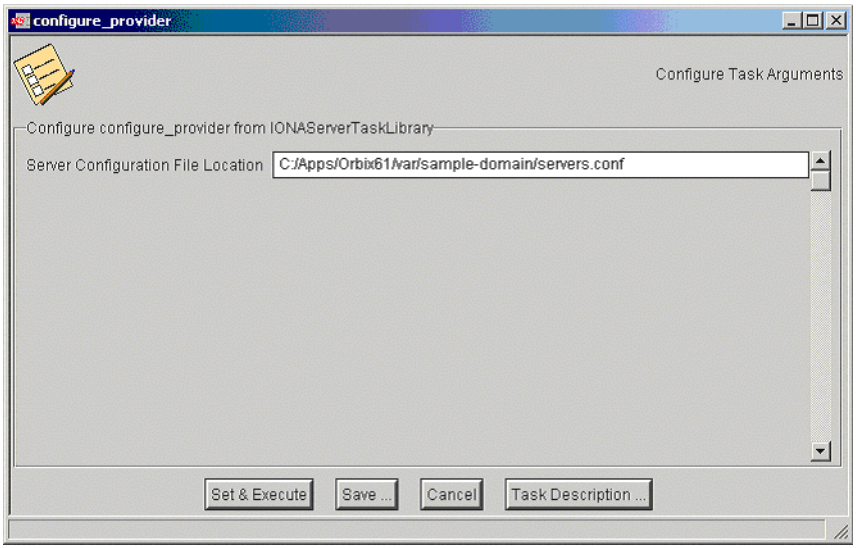

Figure 11: *The configure_provider Task*

Example task: Suppose you want to execute the stop script for the `prdserver` on the `dublin` endpoint. Assuming your `server_commands.txt` file is complete, you can open the `server_command` task selecting the `dublin` endpoint. The script takes the following parameters:

```
Server Command File Location, (location of server_commands.txt)
Server Name,
Server Action,
Server Id
```

The `Server Name` and `Server Id` is `myapplication_prdserver`. The action is `stop`, but the `Server Command File Location` defaults to whatever this location was on the host where you first generated the `tivoli_install.tar`.

You must retype this location so that it points to the correct location on the `dublin` host. Use the same path for your `servers.conf` and `server_commands.txt` files on all hosts, if possible. If not, enter a new location each time that you want to invoke an action on a different host.

Alternatively, you can use the `server_command` task as a template for a new task. After changing the value of `server_commands.txt`, and filling in the other fields, instead of clicking **Set & Execute**, click **Save...**. You can rename this task as, for example, `stop_prdserver_on_dublin`.

If you want more flexibility in deciding which parameters to default and which to leave open, you can create a custom task library based on the IONA Tivoli task library. A description of how this is done is beyond the scope of this document. If you need to do this, contact IONA Professional Services.

Monitor multiple Orbix domains on the same host

You may have more than one Orbix configuration domain running on the same host. Tivoli is not aware of concepts like Orbix configuration domains and the current solution for this is to have the IONA Tivoli resource model perform monitoring of all domains on the same host. This means having only one `servers.conf` or `server_commands.txt` file for each host.

This could potentially cause problems if you have servers on the same host that have the same ORB name and by extension the same default value for the following variable:

`plugins:it_response_time_collector:server-id`

This is why, by default, the server IDs are generated with the domain name added as prefix and the host name added as suffix (for example, `mydomain_locator_myhost`).

A typical `servers.conf` file with two domains (`mydomain` and `yourdomain`) would look as follows:

```
mydomain_locator, 1,
/opt/iona/var/domains/mydomain/logs/locator_myhost_perf.log
...
yourdomain_locator, 1,
/opt/iona/var/domains/yourdomain/logs/locator_yourhost_perf.log
```

Similarly for the task library:

```
mydomain_locator_myhost , start,
/opt/iona/etc/bin/mydomain_locator_start.sh
...
yourdomain_locator_yourhost , start,
/opt/iona/etc/bin/yourdomain_locator_start.sh
```

Further information

For full details on how to use the Tivoli Monitoring product, see your *Tivoli Monitoring User Guide*.

CHAPTER 5

Using the IONA Tivoli Integration

This chapter explains how to perform common tasks using the IONA Tivoli integration. For example, how to access historical data, or detect when a server is down.

In this chapter

This chapter contains the following sections:

Detecting Common Server Problems

Overview

This section explains how to detect common server problems using the IONA Tivoli integration. It includes the following:

- "Detecting possible server crashes".
- "Detecting problems with response times".
- "Detecting heavy traffic".
- "Enabling data logging for your servers".
- "Further information".

Detecting possible server crashes

An `Ev_IONAServer_ServerStatus_matches` event is sent when a server listed in `servers.conf` fails to log a `status=running` message since the beginning of the last cycle. The `Ev_IONAServer_ServerStatus_matches` event contains information about the identity of the server that has stopped running.

The cycle time can be set appropriately before you distribute your profile. It is important that the configured value of the `plugins:it_response_time_collector:period` is always less than the cycle time. Otherwise, you may get spurious events of this type. The specified period should divide evenly into your cycle time (for example, a period of `20` and a cycle time of `60`).

For more details on configuration variables, see "Extending to a Production Environment" on page 45.

Detecting problems with response times

If the average response time of a server exceeds the average response time threshold (`Thr_IONAServer_Resource_Model_AvgResponseTime_gt`), an event is emitted to warn the user. A higher than expected response time may indicate a heavy load or possibly a failure that is causing an unexpectedly slow response for users. This threshold should be set appropriately for the servers that you are monitoring. This can be done in a profile or a policy.

There is also a threshold for maximum response times (`Thr_IONAServer_Resource_Model_MaxResponseTime_gt`). The maximum response time refers to the slowest operation that took place on a server during the last collection cycle. Typically, this value can vary a lot more than the average response time, so you might want to set this threshold higher than the average response time.

Detecting heavy traffic

The `NumInvocations` parameter tracks the number of invocations being processed by the server during each cycle. You must treat this metric with caution because it is not normalized and can be prone to sampling errors.

For example, small differences in the actual cycle time could mean that you pick up an extra log entry during the lifetime of a particular cycle. This can lead to a spike in the data.

The effect of this is lessened when the ratio of cycle time/collector period increases. For example, if the performance logging plugin logs data every 60 seconds and the cycle time is 60 seconds, the error could be as much as +/- 100%. If the ratio of cycle time/collector period is 10, the error for this parameter is +/- 10%.

Enabling data logging for your servers

Before you distribute your `IONAProfile`, or indeed any profile based on the IONA Tivoli resource model, it is recommended that you enable logging in the profile as follows:

1. In the Tivoli **Monitoring Profile** window, double click on **IONAServer Monitor** in the top pane.
2. This launches an **Edit Resource Model** window, click on the **Logging...** button in this window.
3. Ensure that the **Enable Data Logging** button is checked.
4. Click **Apply Changes and Close**.
5. Click **Modify & Close** in the **Edit Resource Model** window.

If you do this before distributing the profile, the Tivoli Agent will track and summarize data for all of the attributes in the resource model. You can use these historical logs for a number of tasks (for example, server downtime, explained in the next section).

Further information

For descriptions of all the events, thresholds, and parameters in the IONA Tivoli resource model, see Appendix A.

Tracking Server Performance Metrics

Overview

This section explains how to track key server performance metrics (for example, server downtime and response time). It includes:

- "Examining server downtime".
- "Tracking other server performance metrics".

Examining server downtime

To examine server downtime, perform the following steps:

1. Open the **Web Health Console** and connect to a machine that is running your profile.
2. In the top pane (the one labelled **Resource Models on Hostname**), select the **Historical Data** radio button.
3. In the bottom pane, choose the `IONAServer_Resource_Model` in the left-hand drop-down box, and `IONAServer_Resource_Model_Availability` in the right-hand drop-down box.
4. In the left-hand selection, choose the name of the server that you want to examine.
5. In the right-hand selection, choose `ServerStatus`.

A table is displayed that shows when the server was running, and for what periods (if any) that its status was unknown. This will most likely be because the server was not running.

Tracking other server performance metrics

Follow steps 1-4 listed for "Examining server downtime". But this time, choose a different metric on the right.

For example, to view a history of the average response time of your server, choose `AvgResponseTime (AVG)`. The data is displayed in tabular form for the last hour, by default. However, you can choose to view data for longer periods. The range of graphical presentation options, such as line and bar charts, can give a useful insight into your server usage patterns.

Another metric of interest is `NumOperations`. This tracks the throughput of your server. Viewing the history can help you identify times when the server usage peaks.

Stopping, Starting, and Restarting Servers

Overview

This section explains how to use the `IONAServerTaskLibrary` to perform actions on servers (for example, stop, start, or restart). It includes:

- "Establishing which servers and operations are tracked".
- "Example of starting the locator service".

Establishing which servers and operations are tracked

The `IONAServerTaskLibrary` enables you to stop, start or restart your servers. To check what servers are recognized by the system and what operations are defined for them, perform the following steps:

1. Double click on the `IONAServerTaskLibrary`.
2. Double click on `list_server_commands`.
3. Click the **Display on Desktop** checkbox.
4. Click the endpoint on which to execute the task.
5. Select **Execute & Dismiss**.
6. Verify that the **Server Command File Location** is correct (this is the `server_commands.txt` file).
7. Click **Set & Execute**.

A list of recognized servers and the operations supported for those servers is displayed.

Example of starting the locator service

To start an Orbix locator service (for example, in the domain `foo`, on the host `patrick`) perform the following steps:

1. Double click on the `IONAServerTaskLibrary`.
2. Double click on `server_command`.
3. Click the **Display on Desktop** checkbox. Select the endpoint on which to execute the task.
4. Click **Execute & Dismiss**.

5. Verify that the **Server Commands File Location** is correct (this is for the `server_commands.txt` file)
6. Fill in the name and ID of the server (`foo_locator_patrick`) and the action (`start`).
7. Click **Set & Execute**.

APPENDIX A

IONA Tivoli Resource Model

This appendix describes the contents of the IONA Tivoli resource model. It includes descriptions of the thresholds, events, and parameters used in this model, along with a WBEM/CIM definition.

In this appendix

This chapter contains the following sections:

Thresholds

This section describes the thresholds in the IONA Tivoli resource model. It lists an internal name and description of each threshold.

Thr_IONAServer_Resource_Model_AvgResponseTime_gt

When the `AvgResponseTime` counter exceeds this threshold, the `Ev_IONAServer_Resource_Model_AvgResponseTime_too_high` event is generated.

The default value is `50`.

This threshold corresponds to the **AvgResponseTime upper bound** threshold displayed in the **Profile Manager** GUI.

Thr_IONAServer_Resource_Model_MaxResponseTime_gt

When the `MaxResponseTime` counter exceeds this threshold, the `Ev_IONAServer_Resource_Model_MaxResponseTime_too_high` event is generated.

The default value is `250`.

This threshold corresponds to the **MaxResponseTime upper bound** threshold displayed in the **Profile Manager** GUI.

Thr_IONAServer_Resource_Model_NumInvocations_lt

When the `NumInvocations` counter is lower than this threshold, the `Ev_IONAServer_Resource_Model_NumInvocations_too_low` event is generated.

The default value is `0`.

This threshold corresponds to the **NumInvocations lower bound** threshold displayed in the **Profile Manager** GUI. This threshold is useful for detecting server hangs when used in conjunction with a ping client that is run at regular intervals.

Thr_IONAServer_Resource_Model_NumInvocations_gt

When the `NumInvocations` counter exceeds this threshold the `Ev_IONAServer_Resource_Model_NumInvocations_too_high` event is generated.

The default value is `100000`.

This threshold corresponds to the **NumInvocations upper bound** threshold displayed in the **Profile Manager** GUI.

Events

This section describes the events in the IONA Tivoli resource model. It lists an internal name and description of each event.

Ev_IONAServer_Resource_Model_AvgResponseTime_too_high

This event is generated when the `AvgResponseTime` counter exceeds the **AvgResponseTime upper bound** threshold.

Ev_IONAServer_Resource_Model_MaxResponseTime_too_high

This event is generated when the `MaxResponseTime` counter exceeds the **MaxResponseTime upper bound** threshold.

Ev_IONAServer_Resource_Model_NumInvocations_too_low

This event is generated when the `NumInvocations` counter is lower than the **NumInvocations lower bound** threshold.

Ev_IONAServer_Resource_Model_NumInvocations_too_high

This event is generated when the server is receiving a large number of invocations, and when the `NumInvocations` counter exceeds the **NumInvocations upper bound** threshold. This can be an indication of overload.

Ev_IONAServer_Server_Status_matches

This event is generated when the status of the server is unknown.

Parameters

This section describes the parameter in the IONA Tivoli resource model. It lists an internal name and description.

Par_Problematic_Status_Values_eqs

This specifies values that indicate a problem with the server status. Possible values are as follows:

- `unknown`
- `shutdown_started`
- `shutdown_complete`

WBEM/CIM Definition

WBEM/CIM refers to Web-Based Enterprise Management/Common Information Model. The WBEM/CIM definition for the IONA Tivoli resource model is as follows:

```
#pragma namespace ("\\\\.\\ROOT\\CIMV2")

[
Dynamic,
M12_Instrumentation("Java.com.iona.management.provider.tivoli.AR
   TILTProviderImpl | | ENUM"),
Provider("M12JavaProvider")
]
class IONAServer
{
    [Key, Description("The unique name of an IONA Server
   Replica")]
    string Identifier;

    [
    Dynamic,

   M12_Instrumentation("Java.com.iona.management.provider.tivoli
   .ARTILTProviderImpl | | GET"),
    Provider("M12JavaProvider")
    ]
    uint32 NumInvocations;

    [
    Dynamic,

   M12_Instrumentation("Java.com.iona.management.provider.tivoli
   .ARTILTProviderImpl | | GET"),
    Provider("M12JavaProvider")
    ]
    uint32 MaxResponseTime;
```

```
    [
     Dynamic,

    M12_Instrumentation("Java.com.iona.management.provider.tivoli
    .ARTILTProviderImpl | | GET"),
     Provider("M12JavaProvider")
     ]
     uint32 MinResponseTime;

     [
     Dynamic,

    M12_Instrumentation("Java.com.iona.management.provider.tivoli
    .ARTILTProviderImpl | | GET"),
     Provider("M12JavaProvider")
     ]
     uint32 AvgResponseTime;

     [
     Dynamic,

    M12_Instrumentation("Java.com.iona.management.provider.tivoli
    .ARTILTProviderImpl | | GET"),
     Provider("M12JavaProvider")
     ]
     string ServerStatus;

};
```

Index