Artix™

Progress Actional Integration Guide

Version 4.2, May 2008

Making Software Work Together™

Updated: June 23, 2008

Contents

List of Figures

Preface

What is covered in this book

Artix supports integration with the following Progress Actional SOA management products:

- Actional for SOA Operations
- Actional Continuous Service Optimization (Actional CSO)

This guide explains how to enable Artix solutions to be monitored by these Actional products. This guide applies to Artix Web service applications written using Java APIs for XML-Based Remote Procedure Call (JAX-RPC).

Who should read this book

This guide is aimed at system administrators using Actional to monitor SOA environments, system architects designing SOA environments, and developers writing SOA applications with Artix. System 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 working knowledge of Actional SOA management products. For more information, see http://www.actional.com.

Organization of this book

This book contains the following chapter:

- Chapter 1 describes the architecture of the Artix integration with Actional.
- Chapter 2 explains how to configure the Artix integration with Actional, and shows examples from Artix–Actional integration demos.
- Chapter 3 gives guidelines on deploying Artix–Actional integration in some example production environments.

The Artix Documentation Library

For information on the organization of the Artix library, the document conventions used, and where to find additional resources, see Using the Artix Library.

CHAPTER 1

Artix–Actional Integration

Artix provides support for integration with Progress Actional SOA management products.

In this chapter

This chapter includes the following section:

Artix–Actional Interaction Architecture

Overview

Integration between Artix and Actional enables Artix services to be monitored by Actional SOA management products. For example, you can use Actional SOA management tools to perform monitoring, auditing, and reporting on Artix services. You can also correlate and track messages through your network to perform dependency mapping and root cause analysis.

The Artix–Actional integration is deployed on Artix service endpoints to enable reporting of management data back to the Actional server. The data reported back to Actional includes system administration metrics such as response time, fault location, auditing, and alerts based on policies and rules.

This Artix–Actional integration can be used with Artix Web service applications written in Java (JAX-RPC).

Artix–Actional integration architecture

The Actional SOA management system includes an Actional server and an Actional agent. An Actional agent is run on each Artix service endpoint node that you wish to manage.

The Artix service endpoint to be managed by Actional uses Actional's interceptor API to send monitoring data to the Actional agent. The Actional server pings the Actional agent periodically to retrieve the monitoring data. It analyzes this data and represents it in the Actional SOA management GUI tools. In addition, any alerts triggered at the Actional agent are sent immediately to the Actional server.

Figure 1 shows how Artix Web service applications are integrated with Actional using this architecture.

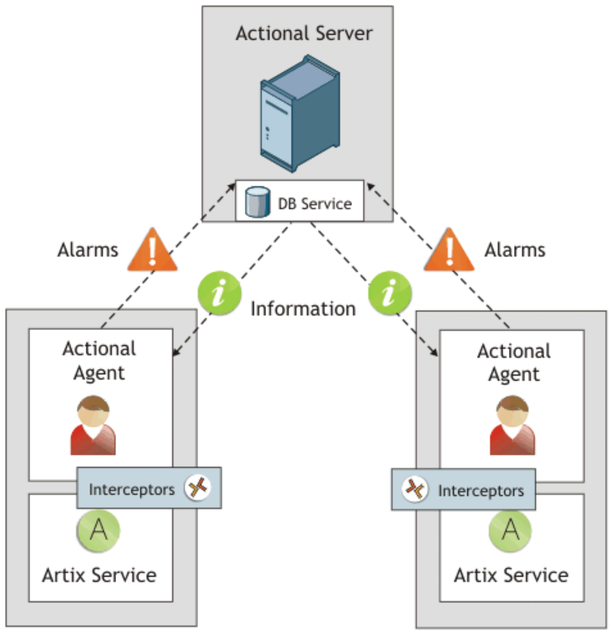


Figure 1: *Artix–Actional Integration Architecture*

The main components in this architecture are:

- "Actional server"
- "Actional agent"
- "Artix interceptors"
- "Actional agent interceptor API"
- "Artix service endpoints"
- "Service consumers"

Actional server

The Actional server is a central management server that manages service endpoint nodes containing an Actional agent.

The Actional server hosts a database and pings Actional agents to obtain management data at configured time intervals. It analyzes the management data and displays it in an Actional console—for example, the **Actional Server Administration Console**. This is a Web application deployed on Apache Tomcat, runtime management and agent configuration modes.

By default, the Actional server uses port `4040`. The default Actional server database is Apache Derby.

Actional agent

An Actional agent is run on each Artix service endpoint node that you wish to manage. Actional agents are used to provide instrumentation data back to the Actional server.

Actional agents are provisioned from the Actional server to establish initial contact and send configuration to the Actional agent. There is one Actional agent per service endpoint node. By default, the Actional agent uses port `4041`.

Artix interceptors

At the level of an endpoint node, Artix interceptors send the instrumentation data to the Actional agent using an Actional-specific API. These interceptors essentially push events to the Actional agent.

The data is analyzed and stored in the Actional agent for retrieval later by the Actional server. However, any alerts triggered at the Actional agent are sent immediately to the Actional server.

Artix Java handlers

In Artix Java, interceptors are also known as *Java handlers*. For example, at the implementation level, Java handlers are used as follows:

1. Artix initializes a Java plug-in that loads a Java handler factory.
2. The handler factory creates client-side request and message handlers, and server-side request and message handlers.

3. When the Artix client-side request handler is invoked, Artix initiates an client Actional interaction object, and sets the following data on this object:
   - Service name
   - Port name
   - Operation name
   - Endpoint URL
   - IP address
   - Correlation ID
4. When client message handler is invoked, Artix gets the message payload and sets it in the client Actional interaction object.
5. On the Artix server side, when the request reaches the server-side message handler, Artix starts a server Actional interaction object and sets the message payload.
6. When the server-side request handler is invoked, Artix sets the same data listed in step 3 on the server Actional interaction object.

Actional agent interceptor API

The Actional Agent Interceptor SDK is an Actional-specific API used to send the management instrumentation data from the service endpoint to the Actional agent.

The Artix service application to be managed by Actional must use the Actional Agent Interceptor SDK to send monitoring data to the Actional agent. For detailed information on how to use this API, see the Actional product documentation.

Artix service endpoints

An Artix service endpoint is a service built using Artix, and described using WSDL. The endpoint can be implemented in Java (JAX-RPC). However, the main characteristic of an Artix service endpoint is that it can be described in WSDL, and classified as a service, which can be consumed.

Service consumers

Service consumers are clients that consume service endpoints by exchanging messages based on the service interface. Consumers can be built using Artix, or any product that supports the technology used by the endpoint. For example, a pure CORBA client could be a consumer for a CORBA endpoint. A .NET client could be a consumer for an Artix SOAP endpoint.

Actional SOA management system

In this document, Actional is the general term used to describe the Actional SOA management system in which all data is stored and viewed. This simplifies the architecture of Actional for the sake of this discussion.

Figure 2 shows an example of the **Actional Server Administration Console**. Managed endpoint nodes are displayed as orange boxes, and unmanaged nodes are displayed as grey boxes. The green arrow indicates the message flow through various nodes.

Clicking on each of the nodes shows more in-depth information regarding the response time, alerts and warnings, and so on. The organization of the information in this web console is in the form of *Node–Group–Service–Operation*. In Artix, this translates to *Node–Service–Port–Operation*.

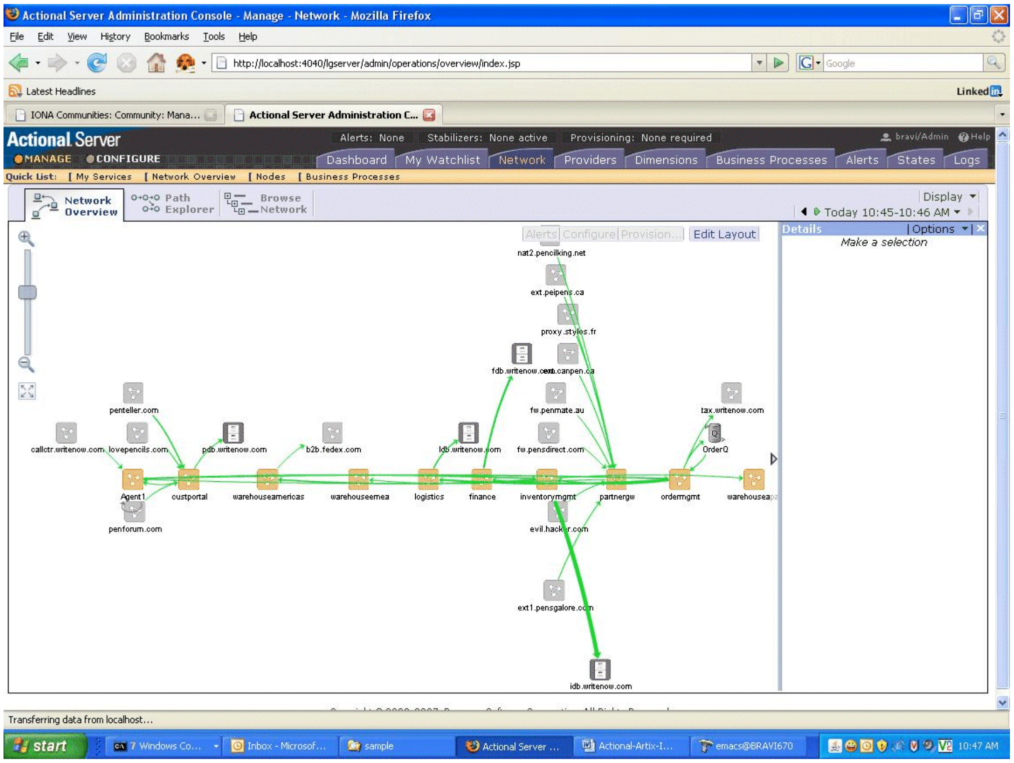


Figure 2: *Actional Server Administration Console*

Further information

For detailed information on using Actional features, see the Actional product documentation.

For more information on Artix Java request/response handlers, see the Developing Artix Applications in Java.

CHAPTER 2

Configuring Artix–Actional Integration

This chapter explains how to configure integration between Artix and Actional SOA management products, and shows examples from Artix–Actional integration demos.

In this chapter

This chapter includes the following sections:

Prerequisites

Overview

This section describes prerequisites for integration between Artix and Actional SOA management products.

Supported product versions

You must have the following version of Artix installed:

- Artix 4.2 (patch 20080620).

This supports integration with the following Actional product versions:

- Actional for SOA Operations 7.1 and 7.2.
- Actional Continuous Service Optimization (Actional CSO) 7.1 and 7.2.

Supported protocols and transports

The following protocols and transports are supported:

- SOAP over HTTP
- SOAP over JMS

Actional agents

You must ensure that Actional agents have been set up on each Artix service endpoint node that you wish to manage. The provisioning of Actional agents is performed using the Actional server. For some basic details, see "Configuring Actional for Artix Integration" on page 19.

For full details of how to set up Actional agents on endpoint nodes, see the Actional product documentation.

Further information

For information on the full range of platform versions and database versions supported by Actional, see the Actional product documentation.

The Artix integration supports the full range of operating system platforms supported by Artix 4.2. For more details, see the Artix Installation Guide.

Configuring Actional for Artix Integration

Overview

These section provides some basic configuration guidelines for Actional agent and server configuration. For full details, see the Actional product documentation.

This basic configuration will help to set up the Artix–Actional integration demos. For information on how to run these demos, see the `readme.txt` files in the following directories:

```
ArtixInstallDir/Version/demos/advanced/management/monitoring/actional_http_handler
ArtixInstallDir/Version/demos/advanced/management/monitoring/actional_jms_handler
```

Actional agent configuration

No specific Actional agent configuration settings are required for integration with Artix. For example, for the purposes of the Actional–Artix integration demos, the Actional agent can be started with the default configuration settings.

Actional server configuration

The following sample configuration steps describe how to set up the Actional server to run an simple Artix–Actional demo:

1. Install the Actional server with typical installation options, and select the Apache Derby database.
2. Specify the following URL in your browser:

   `localhost:4040/lgserver`.
3. If this is a new installation click **Start**, and follow new the Actional server setup steps.

   Otherwise, if the Actional server is already installed, perform the following steps:

   i. In the Actional console Web interface, select the **Configure** radio button in the top left of the screen.

   ii. Select **Platform** tab. This displays the general configuration settings.

Creating a managed node

To create a managed node for a simple Artix demo, perform the following steps:

1. In the Actional **Configure** view menu bar, open the **Network** tab. This displays the **Network Nodes**.
2. Select **Add**. This displays **Node Creation / Managing Agents**.
3. Click **Managed Node**.

Configuring a new node

To configure a managed node for the demo, perform the following steps in the wizard:

Step 1: New Node - Identification

1. Specify the **Name** as `agent1`.
2. Specify the **Display icon** as `auto-discover` (you can select `IONA Artix` from the drop down list, if desired).
3. Click **Next**.

Step 2: New Node - Management

1. Specify the **Transport** as `HTTP/S`.
2. Supply the Actional agent user name and password.
3. Ensure that **Override Agent Database** is checked.
4. Click **Next**.

Step 3: New Node - Agents

1. Specify the following URL:

   ```
   http://HostName:4041/lgagent
   ```

   You can specify a host name or an `IP_ADDRESS`.
2. Ensure that **Limit functionality to Operational Visibility** is not checked.
3. Click **Add**. The agent URL is added.
4. Click **Next**.

Step 4: New Node - Endpoints

1. For **Endpoints**, add the hostname, fully qualified hostname, and IP address.
2. Click **Next**.

Step 5: New Node - Filters

1. Do not specify any filters for the demo.
2. Click **Next**.

Step 6: New Node - Trust Zone

1. Do not specify a trust zone the demo.
2. Click **Finish**

The node is created, and needs to be provisioned.

Provisioning a new node

To provision the new node, perform the following steps:

1. Select the **Deployment** tab from the **Configure** menu bar.
2. The **Provisioning** page is displayed, and `agent1` is listed as not provisioned.
3. Select the `agent1` check box.
4. Click **Provision**. This displays a message when complete: `Successfully provisioned`.
5. Click the **Manage** radio button on the Actional Web interface. You should see `agent1` added to the **Network Overview** screen.

Configuring Artix Java Services for Actional Integration

Overview

This section explains how to configure Artix Java (JAX-RPC) services for integration with Actional. It shows some examples from the Artix–Actional integration demos:

```
ArtixInstallDir/Version/demos/advanced/management/monitoring/actional_http_handler
ArtixInstallDir/Version/demos/advanced/management/monitoring/actional_jms_handler
```

Configuring the Artix monitoring plug-in

Configuring the Artix monitoring plug-in includes the following steps:

- Specifying the plug-in name
- Adding the Java handlers to the interceptor chain
- Configuring the monitoring tool

You can configure the monitoring plug-in by editing your Artix configuration (`artix.cfg`) file.

Specifying the plug-in name

To set the monitoring plug-in factory class, and load the plug-in name, add the following settings:

```
# Configure the plug-in factory class:
plugins:monitoring_plugin:classname =
    "com.iona.jbus.management.monitoring.interceptors.MonitoringPlugInFactory";

# Load the java plug-in:
orb_plugins = ["soap", "java"];

# Load the monitoring plug-in:
java_plugins = ["monitoring_plugin"];
```

Adding the monitoring handlers to the interceptor chain

You must specify monitoring handlers to the request-level and message-level interceptor lists, on both the client side and server side:

```
# Add the client-side handlers to the interceptors chain.
binding:artix:client_request_interceptor_list= "monitoring_handler";
binding:artix:client_message_interceptor_list= "monitoring_handler";

# Add the server-side handlers to the interceptors chain.
binding:artix:server_request_interceptor_list= "monitoring_handler";
binding:artix:server_message_interceptor_list= "monitoring_handler";
```

For more details on configuring binding lists and interceptors, see Artix Configuration Reference.

Configuring the monitoring tool

You must also configure the name of the reporting tool (in this case, `actional`). `actional` is currently the only supported value. For example:

```
plugins:monitoring_plugin:know_report_tool= "actional";
```

Optimizing your Actional integration

Artix provides the following configuration options to enable you to fine-tune the behavior of the monitoring plug-in.

Reporting the message payload

You can enable reporting of the message payload on the server side (for example, a SOAP message over HTTP). If this option is set to `false`, only the payload size is reported. The default value is:

```
plugins:monitoring_plugin:enable_si_payload = "true";
```

Specifying the maximum size of the payload

You can specify the maximum size in bytes of the message payload to report. If a message payload exceeds this value, only its size is reported, regardless of the value of the `enable_si_payload` option. An example setting is:

```
plugins:monitoring_plugin:max_reported_payload_size= "1024";
```

The default value is `-1` (unlimited).

Enabling a service facade

The service facade feature enables reporting of all interactions with an extra representation of the target service on the client side. This is also known informally as an *extra hop*. This is useful when it is impossible to report what service is being invoked by the client (for example, where a JMS queue exists in the invocation chain). The default value is:

```
plugins:monitoring_plugin:show_service_facade= "false";
```

Sample configuration

The following sample configuration shows some example settings in a `my_app` configuration scope:

```
my_app {

  monitoring_jms_handler {

    plugins:monitoring_plugin:classname =
    "com.iona.jbus.management.monitoring.interceptors.MonitoringPlugInFactory";

    orb_plugins = ["soap", "java"];
    java_plugins = ["monitoring_plugin"];

    # Name of the report tool
    plugins:monitoring_plugin:know_report_tool= "actional";

    # Enable the report of the payload (default = "true")
    plugins:monitoring_plugin:enable_si_payload = "true";

    # Maximum size of the reported payload (default is -1 unlimited)
    plugins:monitoring_plugin:max_reported_payload_size= "-1";

    client {
       binding:artix:client_request_interceptor_list= "monitoring_handler";
       binding:artix:client_message_interceptor_list= "monitoring_handler";

       # Enable service Facade representation
       plugins:monitoring_plugin:show_service_facade= "true";
     };

    server {
       binding:artix:server_request_interceptor_list= "monitoring_handler";
       binding:artix:server_message_interceptor_list= "monitoring_handler";
     };
};
```

Viewing Artix Endpoints in Actional

Overview

When your Artix service endpoints and consumers have been configured for integration with Actional, they can be monitored using the Actional SOA management tools.

For example, when you run the Artix–Actional SOAP over JMS demo, the **Actional Server Administration Console** displays the server queues and agent nodes. Invocations are displayed as arrows flowing to and from the queues. For details on how to run this demo, see the `readme.txt` file in the following directory:

```
ArtixInstallDir/Version/demos/advanced/management/monitoring/actional_jms_handler
```

Network overview

Figure 3 shows a running SOAP over JMS demo displayed in the **Network Overview** screen of the **Actional Server Administration Console**.

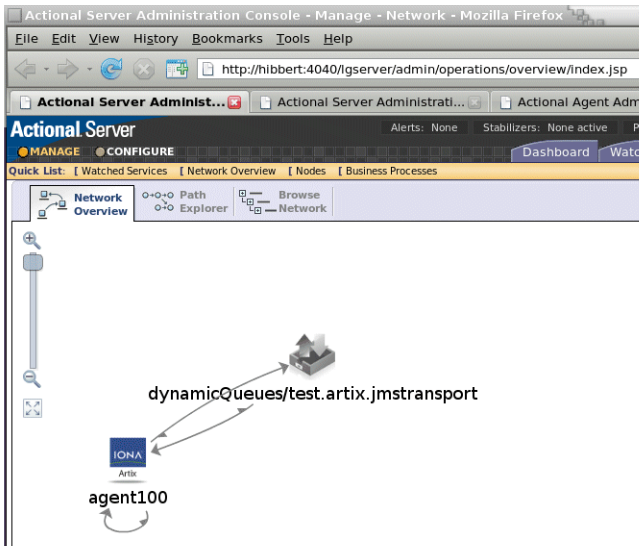

Actional Server Administration Console - Manage - Network - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://hibbert:4040/lgserver/admin/operations/overview/index.jsp
Actional Server Administ...
Actional Server Administrati...
Actional Agent Adm
Actional Server
Alerts: None
Stabilizers: None active
MANAGE
CONFIGURE
Dashboard
Watc
Quick List: [Watched Services [Network Overview [Nodes [Business Processes
Network Overview
Path Explorer
Browse Network
dynamicQueues/test.artix.jmstransport
IONA
Artix
agent100

Figure 3: *Actional Server Network Overview*

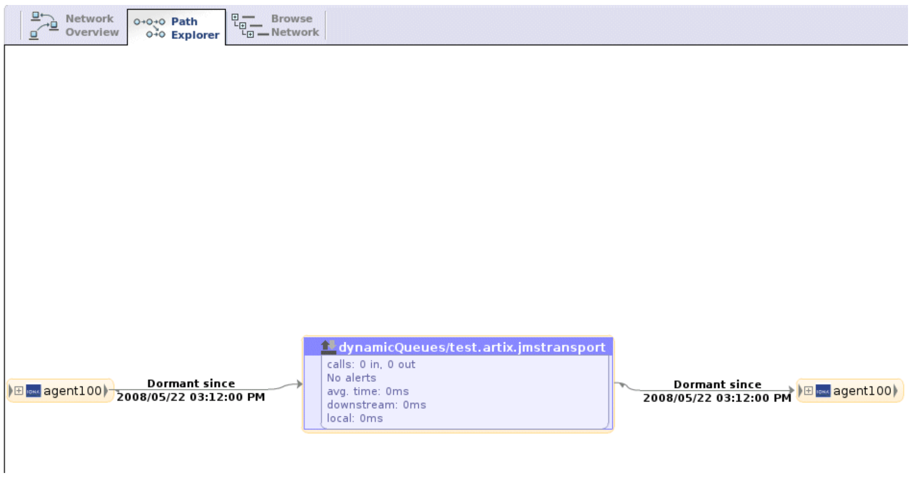
In Figure 3, the JMS queue is displayed on top, and the instrumented Artix application is displayed below. The interactions between the client and server applications are recorded by `agent100`, which is installed on the machine that runs the demo. This agent reports monitoring data back to the Actional server.

The arrows between the agent and the JMS queue represent the invocations: out to the queue from the client application, and back from the queue showing the message received by the service from the queue.

The arrow that loops from `agent100` back to itself is the extra hop, or service facade call. For more details, see "Enabling a service facade" on page 24.

Path Explorer

Figure 4 shows the example JMS queue displayed in the **Path Explorer** screen of the **Actional Server Administration Console**.



calls: 0 in, 0 out
No alerts
avg. time: 0ms
downstream: 0ms
local: 0ms

Figure 4: *Actional Server Path Explorer*

To view this screen, select the JMS queue object in Figure 3, and select the **Path Explorer** tab at the top left. This example shows the invocations from the point of view of the JMS queue.

Viewing a service facade

Figure 5 shows what is displayed when you expand `agent100` in Figure 4, and select the `FACADEsayHI` operation.

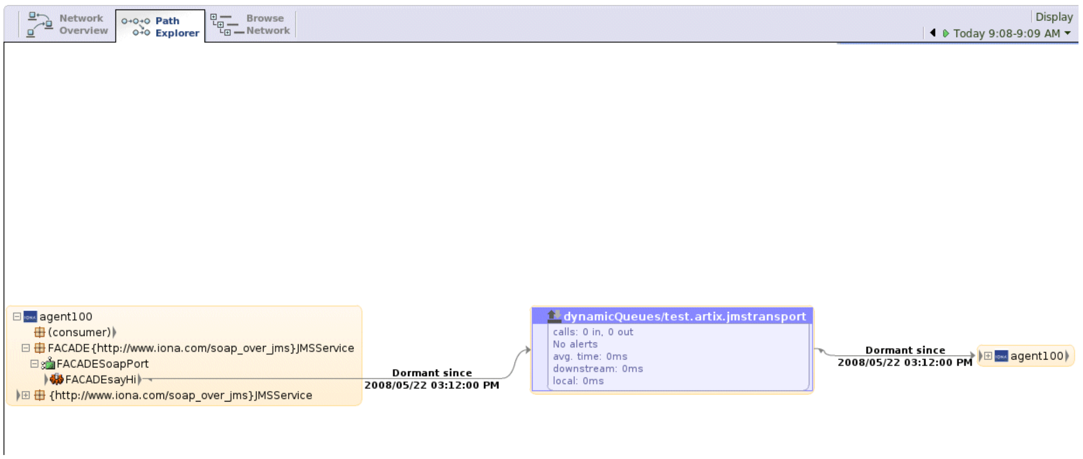

Network Overview
Path Explorer
Browse Network
Display
Today 9:08-9:09 AM
agent100
(consumer)
FACADE {http://www.iona.com/soap_over_jms}JMSService
FACADESoapPort
FACADEsayHi
{http://www.iona.com/soap_over_jms}JMSService
Dormant since 2008/05/22 03:12:00 PM
dynamicQueues/test.artix.jmstransport
calls: 0 in, 0 out
No alerts
avg. time: 0ms
downstream: 0ms
local: 0ms
Dormant since 2008/05/22 03:12:00 PM
agent100

Figure 5: *Service Facade in Path Explorer*

In Figure 5, the facade service appears to make a call to the JMS queue. However, the consumer is the actor that performs the invocation. Without the facade, you would only see an arrow from the consumer directly to the JMS queue. You would not know what service port or operation is invoked through the queue. This is because the queue acts as an opaque buffer for all messages.

Adding the service facade enables you to represent where the call is going. The small arrow head displayed to the right of `consumer` is the extra hop invocation from the consumer to the facade. The small arrow head displayed to the left of the `FACADEsayHi` operation is the other end of the invocation. The Actional console displays the call to the JMS queue as if it originates from the service facade.

Figure 6 shows the display when you expand the `agent100` object on the right of Figure 5. This represents the call from the JMS queue to the service being invoked. The names used for the facade in Figure 5 are constructed from the names for the service, endpoint and operation by prefixing them with `FACADE`.

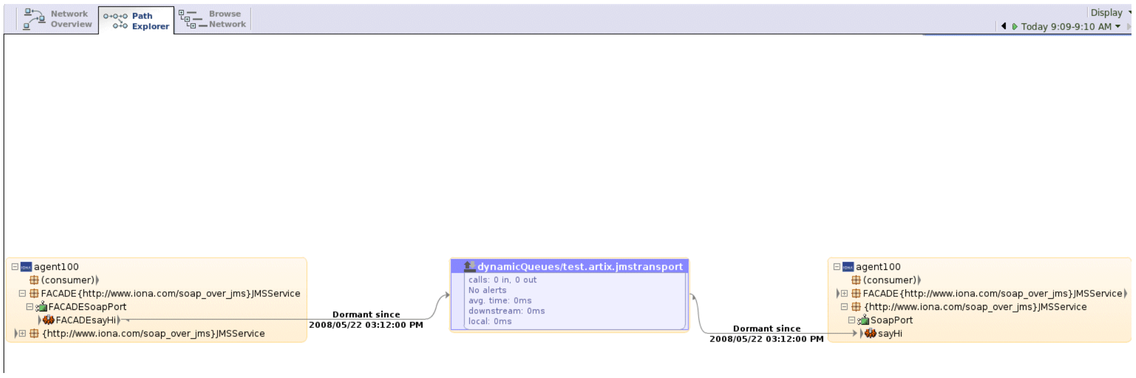

Network Overview
Path Explorer
Browse Network
Display
Today 9:09-9:10 AM
agent100
(consumer)
FACADE{http://www.iona.com/soap_over_jms}JMSService
FACADESoapPort
FACADEsayHi
{http://www.iona.com/soap_over_jms}JMSService
Dormant since 2008/05/22 03:12:00 PM
dynamicQueues/test.artix.jmstransport
calls: 0 in, 0 out
No alerts
avg. time: 0ms
downstream: 0ms
local: 0ms
Dormant since 2008/05/22 03:12:00 PM
agent100
(consumer)
FACADE{http://www.iona.com/soap_over_jms}JMSService
{http://www.iona.com/soap_over_jms}JMSService
SoapPort
sayHi

Figure 6: *Service Facade in More Detail*

Figure 7 shows the interaction from the point of view of `agent100`. In this example, the agent has central position, and calls to and from the JMS queue are displayed. The arrow looping back to `agent100` is the internal extra hop interaction with the service facade.

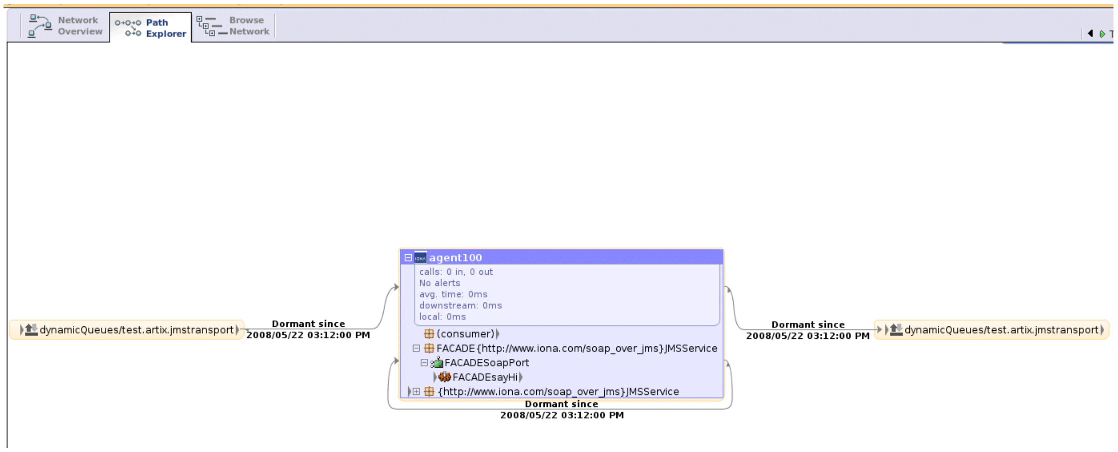

Network Overview
Path Explorer
Browse Network
agent100
calls: 0 in, 0 out
No alerts
avg. time: 0ms
downstream: 0ms
local: 0ms
(consumer)
FACADE{http://www.iona.com/soap_over_jms}JMSService
FACADESoapPort
FACADEsayHi
{http://www.iona.com/soap_over_jms}JMSService
dynamicQueues/test.artix.jmstransport
Dormant since 2008/05/22 03:12:00 PM
Dormant since 2008/05/22 03:12:00 PM
dynamicQueues/test.artix.jmstransport
Dormant since 2008/05/22 03:12:00 PM

Figure 7: *Service Facade Extra Hop*

For more information on service facades, see "Enabling a service facade" on page 24.

Further information

Actional

For information on how to set up and run the Actional server, Actional agent, and Actional Server Administration Console, see the Actional product documentation.

Artix

For more information on Artix configuration, see the following:

- Configuring and Deploying Artix Solutions
- Artix Configuration Reference

CHAPTER 3

Deployment Scenarios

This chapter gives general guidelines on deploying Artix–Actional integration in example production environments.

In this chapter

This chapter includes the following sections:

Deployment with IBM WebSphere and J2EE Connector

Overview

This section gives basic guidelines for integrating Artix and Actional in an example deployment scenario that includes an IBM WebSphere 5.1 environment with an Artix J2EE Connector.

Artix J2EE Connector is a resource adaptor that enables J2EE applications and Artix Web services to talk to each other. It is also used to manage connections, transactions, and security.

For more details on WebSphere, see the IBM WebSphere product documentation. For more details on Artix J2EE Connector, see Artix for J2EE.

IBM WebSphere deployment

Follow these general guidelines when deploying IBM WebSphere:

- Before starting WebSphere, ensure that the Artix environment script (`artix_env`) has been sourced.
- Deploy the Artix J2EE Connector resource adapter archive (`artix.rar`) on WebSphere. Instructions for deploying on WebSphere are described in Artix for J2EE.
- Follow the instructions in the *Actional Interceptor Guide* to enable WebSphere instrumentation.
- Ensure that a copy of `actional-sdk.jar` is present in your `$WAS_HOME/classes` directory.

Artix J2EE Connector deployment

In this example deployment scenario, which includes an Artix J2EE Connector, you also need to update your Artix J2EE Connector classloader firewall configuration file (`artix_j2ee_ce.xml`).

- Add the following entries under the `ce:environment` element:

```
<ce:filter type="pattern" > com.actional. </ce:filter>
<ce:filter type="negative-pattern"> javax.xml.soap. </ce:filter>
<ce:filter type="negative-pattern"> com.iona.jbus.jms. </ce:filter>
<ce:filter type="negative-pattern"> com.iona.jbus.management. </ce:filter>
```

- Add the following entries under the `ce:loader` element (using the fully qualified path):

```
<ce:location> path/to/it_bus_management_monitoring.jar </ce:location>
<ce:location> $IT_PRODUCT_DIR/lib/sun/saaj/1.2.1/saaj-api.jar </ce:location>
```

Artix deployment

When deploying an Artix and Actional integration, you need to add some configuration entries to your Artix configuration file (`artix.cfg`). For example, you must configure the Artix monitoring plug-in; you can also set additional options such as payload reporting and service facade.

For full details, see "Configuring Artix Java Services for Actional Integration" on page 22.

Example Artix configuration scope

The following example shows a `j2ee` configuration scope from an Artix configuration file (`.cfg`):

```
j2ee {

   plugins:monitoring_plugin:classname="com.iona.jbus.management.monitoring.intercept
   ors.MonitoringPlugInFactory";

    plugins:monitoring_plugin:know_report_tool= "actional";

    event_log:filters = ["*=*"];

    orb_plugins = ["xmlfile_log_stream", "iiop_profile", "giop", "iiop", "soap",
   "java"];

    java_plugins = ["monitoring_plugin"];

    binding:artix:client_request_interceptor_list= "monitoring_handler";
    binding:artix:client_message_interceptor_list= "monitoring_handler";
    binding:artix:server_request_interceptor_list= "monitoring_handler";
    binding:artix:server_message_interceptor_list= "monitoring_handler";

    plugins:monitoring_plugin:enable_si_payload="true";
    plugins:monitoring_plugin:max_reported_payload_size="-1";
    plugins:monitoring_plugin:show_service_facade="true";

    tx {

    orb_plugins = ["local_log_stream", "iiop_profile", "giop", "iiop",

    "ws_coordination_service", "soap", "ots", "java"];

     plugins:bus:default_tx_provider:plugin = "wsat_tx_provider";

        xa {
            poa:j2ee_rm:direct_persistent="true";
            poa:j2ee_rm:well_known_address:port="58502";
            initial_references:TransactionFactory:plugin = "ots_encina";
        };
    };
};
```

Native Deployment with IBM WebSphere

Overview

This section gives basic guidelines for integrating Artix and Actional in an example deployment scenario that includes a native IBM WebSphere 5.1 environment (without Artix J2EE Connector).

For more details on WebSphere, see the IBM WebSphere product documentation.

Native WebSphere deployment

Using Artix natively with WebSphere requires specific WebSphere configuration settings. For example, WebSphere classloaders do not use the system classpath. This means that you must specify classloader configuration to WebSphere without causing class clashes—WebSphere has some common components with Artix; but different versions.

Here are some important deployment considerations:

- The WebSphere server runs as a single process with multiple threads. Because Artix limits creation of only one Artix bus per process by default, you must initialize different Artix buses in WebSphere with different ORB IDs. Different ORB names are not sufficient.
- WebSphere reads the shared library path from the starting shell. You must start a shell environment, source your Artix environment, and then start WebSphere from that shell. If your Artix environment is not set before starting the WebSphere server, this results in failure.
- “WebSphere configuration steps” on page 36 describes how to add core Artix JARs to the WebSphere extended classloader, but it does not include all Artix JARs. If you use a non-standard Artix subsystem (for example, AmberPoint), you may need to add additional JARs to the list.

Before you begin

Follow these general guidelines:

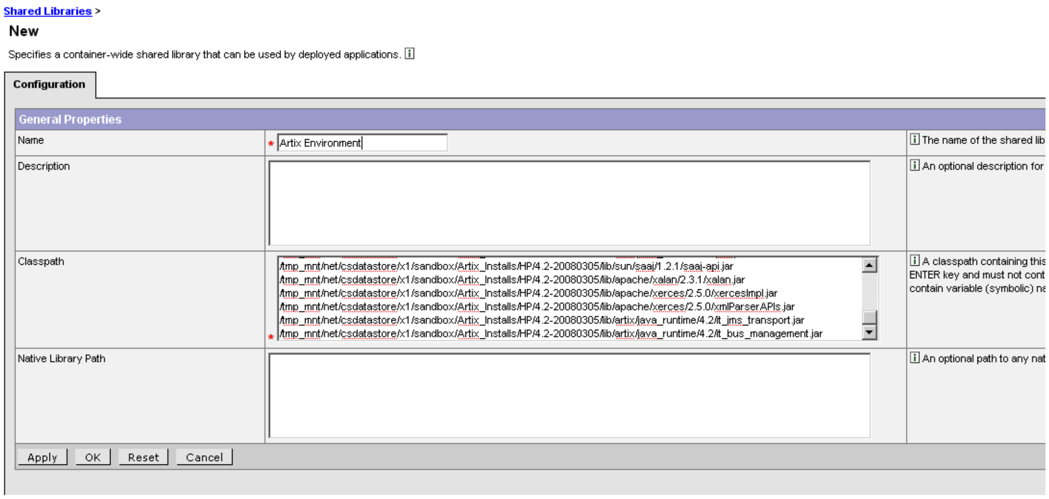
- Before starting WebSphere, ensure that the Artix environment script (`artix_env`) has been sourced.
- Follow the instructions in the *Actional Interceptor Guide* to enable WebSphere instrumentation.
- Ensure that a copy of `actional-sdk.jar` is present in your `$WAS_HOME/classes` directory.

WebSphere configuration steps

In a native IBM WebSphere deployment, perform the following steps.

1. Start WebSphere server, from a console that has already has the Artix environment set (using the `artix_env` script).
2. Launch the WebSphere administration console. This is generally available on: `*HostName*:9090/admin`. If no security is turned on, enter any user name to login
3. In the main menu on the left, select **Environment|Shared Libraries**.
4. On the **Shared Libraries** screen, in the `Scope` definition, select the `Server` radio button. This causes the shared library definition to applicable only the server level.
5. Click **Apply**.
6. To create a new shared library entry, select **New**.

7. In the **Configuration** tab, enter a shared library name in **Name** text box; for example, `Artix Environment` (see Figure 8).



Shared Libraries >

New

Specifies a container-wide shared library that can be used by deployed applications.

Configuration

General Properties

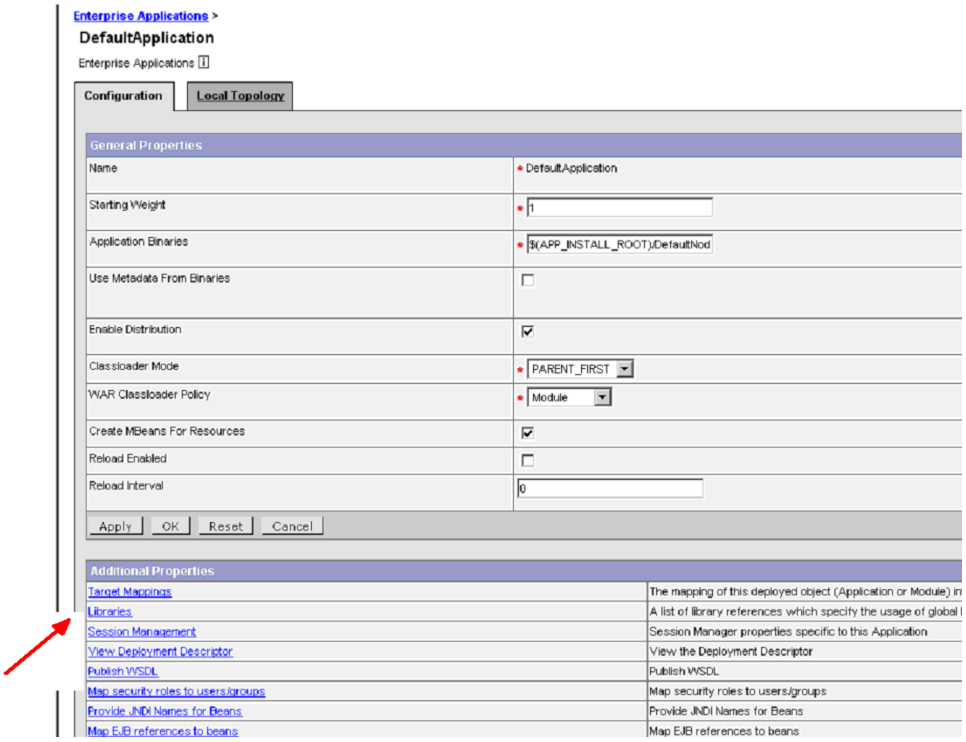
Name	* Artix Environment	The name of the shared lib
Description		An optional description for
Classpath	/tmp_mnt/net/csdatastore/x1/sandbox/Artix_Installs/HP/4.2-20080305/lib/sun/saaj/1.2.1/saaj-api.jar /tmp_mnt/net/csdatastore/x1/sandbox/Artix_Installs/HP/4.2-20080305/lib/apache/xalan/2.3.1/xalan.jar /tmp_mnt/net/csdatastore/x1/sandbox/Artix_Installs/HP/4.2-20080305/lib/apache/xerces/2.5.0/xercesImpl.jar /tmp_mnt/net/csdatastore/x1/sandbox/Artix_Installs/HP/4.2-20080305/lib/apache/xerces/2.5.0/xmlParserAPIs.jar /tmp_mnt/net/csdatastore/x1/sandbox/Artix_Installs/HP/4.2-20080305/lib/artix/java_runtime/4.2/it_jms_transport.jar * /tmp_mnt/net/csdatastore/x1/sandbox/Artix_Installs/HP/4.2-20080305/lib/artix/java_runtime/4.2/it_bus_management.jar	A classpath containing this ENTER key and must not cont contain variable (symbolic) na
Native Library Path		An optional path to any nat

Figure 8: *Specifying a New Shared Library*

8. In the **Classpath** text box, paste the full path to the following Artix JAR libraries:

```
ArtixInstallDir/etc
ArtixInstallDir/lib/apache/jakarta-log4j/1.2.6/log4j.jar
ArtixInstallDir/artix/4.2/etc
ArtixInstallDir/lib/common/ifc/1.3/ifc.jar
ArtixInstallDir/lib/artix/java_runtime/4.2/it_bus.jar
ArtixInstallDir/lib/artix/java_runtime/4.2/it_bus-api.jar
ArtixInstallDir/lib/artix/java_runtime/4.2/it_context_library.jar
ArtixInstallDir/lib/ws_common/jaxrpc/1.3/it_jaxrpc.jar
ArtixInstallDir/lib/ws_common/saaj/1.3/it_saaj.jar
ArtixInstallDir/lib/ws_common/reflect/1.3/it_ws_reflect.jar
ArtixInstallDir/lib/ws_common/reflect/1.3/it_ws_reflect_types.jar
ArtixInstallDir/lib/ws_common/wsdl/1.3/it_wsdl.jar
ArtixInstallDir/lib/jaxrpc/jaxrpc/1.1/jaxrpc-api.jar
ArtixInstallDir/lib/artix/java_runtime/4.2/jms.jar
ArtixInstallDir/lib/sun/saaj/1.2.1/saaj-api.jar
ArtixInstallDir/lib/apache/xalan/2.3.1/xalan.jar
ArtixInstallDir/lib/apache/xerces/2.5.0/xercesImpl.jar
ArtixInstallDir/lib/apache/xerces/2.5.0/xmlParserAPIs.jar
ArtixInstallDir/lib/artix/java_runtime/4.2/it_jms_transport.jar
ArtixInstallDir/lib/artix/java_runtime/4.2/it_bus_management.jar
ArtixInstallDir/lib/artix/java_runtime/4.2/it_bus_management_monitoring.jar
ArtixInstallDir/lib/activemq/activemq/4.0.1/incubator-activemq-4.0.1.jar
```

9. Select **Apply** and **Save**.
10. In the main menu, select **Application|Enterprise Applications**.
11. Select the application that will use Artix natively (for example, `DefaultApplication` in Figure 9).
12. On the **Configuration** tab, in **General Properties**, set the **Classloader Mode** to `PARENT_LAST`.
13. Set the **WAR Classloader Policy** to `Application`.
14. Scroll down to **Additional Properties**, and select **Libraries**.



Enterprise Applications >
DefaultApplication
Enterprise Applications

General Properties

Name	* DefaultApplication
Starting Weight	* 1
Application Binaries	* $(APP_INSTALL_ROOT)/DefaultNod
Use Metadata From Binaries	☐
Enable Distribution	✓
Classloader Mode	* PARENT_FIRST
WAR Classloader Policy	* Module
Create MBeans For Resources	✓
Reload Enabled	☐
Reload Interval	0

Additional Properties

Target Mappings	The mapping of this deployed object (Application or Module) in
Libraries	A list of library references which specify the usage of global l
Session Management	Session Manager properties specific to this Application
View Deployment Descriptor	View the Deployment Descriptor
Publish WSDL	Publish WSDL
Map security roles to users/groups	Map security roles to users/groups
Provide JNDI Names for Beans	Provide JNDI Names for Beans
Map EJB references to beans	Map EJB references to beans

Figure 9: *Specifying Application Properties*

15. Select **Add**. This displays a list of predefined shared libraries, including the one you defined earlier.
16. Select the shared library you defined (for example, `Artix Environment`).
17. Select **Apply** and **Save**.

18. In the main menu, select **Servers | Application Servers | server1 | Process Definition**.
19. In **Generic JVM Arguments**, add the following:

```
-Dorg.apache.commons.logging.LogFactory=org.apache.commons.logging.impl.LogFactoryImpl
```

20. Select **Apply** and **Save**.

Artix deployment

When deploying an Artix and Actional integration, you must specify some configuration entries in your Artix configuration file (`artix.cfg`). For example, you must configure the Artix monitoring plug-in; you can also set additional options such as payload reporting and service facade.

For full details, see "Configuring Artix Java Services for Actional Integration" on page 22.

Example Artix configuration scope

The following shows an example configuration scope from an Artix configuration file (`.cfg`):

```
demos {

    hello_world_soap_http {

   plugins:monitoring_plugin:classname="com.iona.jbus.management.monitoring.
   interceptors.MonitoringPlugInFactory";

    plugins:monitoring_plugin:know_report_tool= "actional";

    orb_plugins = ["local_log_stream", "xmlfile_log_stream", "soap", "java"];

    java_plugins = ["monitoring_plugin"];

    binding:artix:client_request_interceptor_list= "monitoring_handler";
    binding:artix:client_message_interceptor_list= "monitoring_handler";
    binding:artix:server_request_interceptor_list= "monitoring_handler";
    binding:artix:server_message_interceptor_list= "monitoring_handler";

    plugins:monitoring_plugin:enable_si_payload="true";
    plugins:monitoring_plugin:max_reported_payload_size="-1";
    plugins:monitoring_plugin:show_service_facade="false";
    };
};
```

Further information

Actional

For information on how to set up and run the Actional server, Actional agent, and Actional Server Administration Console, see the Actional product documentation.

Artix

For more information on Artix configuration, see the following:

- Configuring and Deploying Artix Solutions
- Artix Configuration Reference

IBM WebSphere

For more details on WebSphere, see the IBM WebSphere product documentation.

Artix J2EE Connector

For more details on Artix J2EE Connector, see Artix for J2EE.

Index