

## Artix™ ESB

## Managing Artix Solutions with JMX, C++ Runtime

Version 5.0, July 2007

Making Software Work Together™

IONA Technologies PLC and/or its subsidiaries may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this publication. Except as expressly provided in any written license agreement from IONA Technologies PLC, the furnishing of this publication does not give you any license to these patents, trademarks, copyrights, or other intellectual property. Any rights not expressly granted herein are reserved.

IONA, IONA Technologies, the IONA logos, Orbix, Artix, Making Software Work Together, Adaptive Runtime Technology, Orbacus, IONA University, and IONA XMLBus are trademarks or registered trademarks of IONA Technologies PLC and/or its subsidiaries.

Java and J2EE are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. CORBA is a trademark or registered trademark of the Object Management Group, Inc. in the United States and other countries. All other trademarks that appear herein are the property of their respective owners.

IONA Technologies PLC makes no warranty of any kind to this material including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. IONA Technologies PLC shall not be liable for errors contained herein, or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

#### COPYRIGHT NOTICE

No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, photocopying, recording or otherwise, without prior written consent of IONA Technologies PLC. No third party intellectual property right liability is assumed with respect to the use of the information contained herein. IONA Technologies PLC assumes no responsibility for errors or omissions contained in this book. This publication and features described herein are subject to change without notice.

Copyright © 2001-2007 IONA Technologies PLC. All rights reserved.

All products or services mentioned in this manual are covered by the trademarks, service marks, or product names as designated by the companies who market those products.

Updated: September 3, 2007

## Contents

List of Tables	7
	-
Preface	9
What is Covered in this Book	9
Who Should Read this Book	9
How to Use this Book	9
The Artix Documentation Library	9
Chapter 1 Monitoring and Managing an Artix Runtime	11
Introduction	12
Managed Bus Components	17
Managed Service Components	24
Artix Locator Service	29
Artix Session Manager Service	31
Managed Port Components	32
Chapter 2 Configuring JMX in an Artix Runtime	37
Artix JMX Configuration	38
Chapter 3 Managing Artix Services with JMX Consoles	41
Managing Artix Services with JConsole	42
Managing Artix Services with the JMX HTTP adaptor	46
Managing Artix Services with MC4J	49
Managing Logging Levels with MC4J	60
Chapter 4 Managing WS-RM Persistence with JMX	69
WS-RM Persistence Management	70
Viewing Messages in the WS-RM Persistence Database	72
Index	79

CONTENTS

# List of Figures

Figure 1: Artix JMX Architecture	13
Figure 2: Managed Service in JConsole	43
Figure 3: Managed Port in JConsole	44
Figure 4: Managed Locator in JConsole	45
Figure 5: HTTP Adaptor Main View	47
Figure 6: HTTP Adaptor Bus View	48
Figure 7: Connecting to a Server	50
Figure 8: Server Connection Details	51
Figure 9: Creation of Server Connection	52
Figure 10: New Server Connection	53
Figure 11: Viewing Service Properties	54
Figure 12: Viewing Service Counters Properties	55
Figure 13: Stopping a Service	55
Figure 14: Deactivated Service	56
Figure 15: Activated a Service	57
Figure 16: Viewing Port Properties	58
Figure 17: Viewing Interceptor Properties	59
Figure 18: Logging Viewing Wizard	61
Figure 19: Entering a Logging Subsystem	62
Figure 20: Displayed Logging Level	63
Figure 21: Setting a Logging Level	64
Figure 22: Logging Level Set Successfully	65
Figure 23: Propagating a Logging Level	66
Figure 24: Connecting to a JMX Agent	73
Figure 25: WS-RM Persistence Enabled Endpoint	74
Figure 26: WS-RM Sequence Attributes	76

LIST OF FIGURES

Figure 27: Messages in the WS-RM Persistence database

77

# List of Tables

Table 1: Managed Bus Attributes	18
Table 2: Managed Bus Methods	19
Table 3: Managed Service Attributes	25
Table 4: serviceCounters Attributes	26
Table 5: Managed Service Attributes	27
Table 6: Locator MBean Attributes	29
Table 7: Session Manager MBean Attributes	31
Table 8: Supported Service Attributes	32

LIST OF TABLES

## Preface

#### What is Covered in this Book

Managing Artix Solutions with JMX, C++ Runtime explains how to monitor and manage Artix services in a runtime environment using Java Management Extensions (JMX). It applies to services written using both C++ and Java API for XML-Based Remote Procedure Call (JAX-RPC).

#### Who Should Read this Book

The main audience of *Managing Artix Solutions with JMX* is Artix system administrators. However, anyone involved in designing a large scale Artix solution will find this book useful.

#### How to Use this Book

This book includes the following:

- Chapter 1 introduces the Artix JMX architecture and describes the Artix components that can be managed using JMX.
- Chapter 2 explains how to configure an Artix runtime for JMX.
- Chapter 3 explains how to manage and monitor Artix services using JMX consoles.
- Chapter 4 explains how to manage Web services Reliable Messaging persistence in Artix using JMX.

#### 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.

PREFACE

### CHAPTER 1

# Monitoring and Managing an Artix Runtime

This chapter explains how to monitor and manage an Artix C++ runtime using Java Management Extensions (JMX). It applies to applications written using both C++ and Java API for XML-Based Remote Procedure Call (JAX-RPC).

This chapter discusses the following topics:

Introduction	page 12
Managed Bus Components	page 17
Managed Service Components	page 24
Managed Port Components	page 32

In this chapter

### Introduction

#### Overview

You can use Java Management Extensions (JMX) to monitor and manage key Artix runtime components both locally and remotely. For example, using any JMX-compliant client, you can perform the following tasks:

- View bus status.
- Stop or start a service.
- Change bus logging levels dynamically.
- Monitor service performance details.
- View the interceptors for a selected port.

#### How it works

Artix has been instrumented to allow runtime components to be exposed as JMX Managed Beans (MBeans). This enables an Artix runtime to be monitored and managed either in process or remotely with the help of the JMX Remote API.

Artix runtime components can be exposed as JMX MBeans, out-of-the-box, for both C++ and JAX-RPC Artix servers. In addition, support for registering custom MBeans is also available. Java developers can create their own MBeans and register them either with their MBeanServer of choice, or with a default MBeanServer created by Artix (see "Relationship between runtime and custom MBeans" on page 14).

Figure 1 shows an overview of how the various components interact. The Java custom MBeans are optional components that can be added as required.

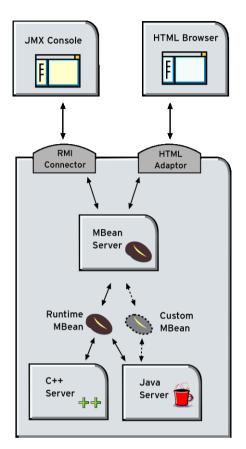


Figure 1: Artix JMX Architecture

#### What can be managed Both Artix C++ and JAX-RPC servers can have their runtime components exposed as JMX MBeans. The following components can be managed: Bus Service • Port All runtime components are registered with an MBeanServer as Open Dynamic MBeans. This ensures that they can be viewed by third-party management consoles without any additional client-side support libraries. All MBeans for Artix runtime components conform with Sun's JMX Best Practices document on how to name MBeans (see http://java.sun.com/products/JavaManagement/best-practices.html). Artix runtime MBeans use com.iona.instrumentation as their domain name when creating ObjectNames. **Note:** An MBeanServerConnection, which is an interface implemented by the MBeanServer is used in the examples in this chapter. This ensures that the examples are correct for both local and remote access. See also "Further information" on page 16 for details of how to access MBean Server hosting runtime MBeans either locally and remotely. Relationship between runtime The Artix runtime instrumentation provides an out-of-the-box JMX view of and custom MBeans C++ and JAX-RPC services. Java developers can also create custom JMX MBeans to manage Artix Java components such as services. You may choose to write custom Java MBeans to manage a service because the Artix runtime is not aware of the current service's application semantics. For example, the Artix runtime can check service status and update performance counters, while a custom MBean can provide details on the status of a business loan request processing. It is recommended that custom MBeans are created to manage application-specific aspects of a given service. Ideally, such MBeans should not duplicate what the runtime is doing already (for example, calculating service performance counters).

It is also recommended that custom MBeans use the same naming convention as Artix runtime MBeans. Specifically, runtime MBeans are named so that containment relationships can be easily established. For example:

```
// Bus :
com.iona.instrumentation:type=Bus,name=demos.jmx_runtime
Service :
com.iona.instrumentation:type=Bus.Service,name="{http://ws.iona.
    com}SOAPService",Bus=demos.jmx_runtime
// Port :
com.iona.instrumentation:type=Bus.Service.Port,name=SoapPort,Bus
    .Service="{http://ws.iona.com}SOAPService",Bus=demos.jmx_runt
    ime
```

Using these names, you can infer the relationships between ports, services and buses, and display or process a complete tree in the correct order. For example, assuming that you write a custom MBean for a loan approval Java service, you could name this MBean as follows:

```
com.iona.instrumentation:type=Bus.Service.LoanApprovalManager,na
me=LoanApprovalManager,Bus.Service="{http://ws.iona.com}SOAPS
ervice",Bus=demos.jmx runtime
```

For details on how to write custom MBeans, see Developing Artix Applications in Java.

Artix runtime support for JMX is enabled using configuration settings only. You do not need to write any additional Artix code. When configured, you can use any third party console that supports JMX Remote to monitor and manage Artix servers.

If you wish to write your own JMX client application, this is also supported. To access Artix runtime MBeans in a JMX client, you must first get a handle to the MBeanServer. The following code extract shows how to access the MBeanServer locally:

```
Bus bus = Bus.init(args);
MBeanServer mbeanServer =
  (MBeanServer)bus.getRegistry().getEntry(ManagementConstants.M
  BEAN SERVER INTERFACE NAME);
```

### Accessing the MBeanServer programmatically

The following shows how to access the MBeanServer remotely:

```
// The address of the connector server
String url = "service:jmx:rmi://host:1099/jndi/artix";
JMXServiceURL address = new JMXServiceURL(url);
// Create the JMXConnectorServer
JMXConnector cntor = JMXConnectorFactory.connect(address, null);
// Obtain a "stub" for the remote MBeanServer
MBeanServerConnection mbsc = cntor.getMBeanServerConnection();
Please see the following demo for a complete example on how to access,
monitor and manage Artix runtime MBeans remotely:
InstallDir\Version\cxx_java\samples\advanced\management\jmx_runtime
```

**Further information** 

For further information, see the following URLs:

#### JMX

http://java.sun.com/products/JavaManagement/index.jsp

#### JMX Remote

http://www.jcp.org/aboutJava/communityprocess/final/jsr160/

#### **Open Dynamic MBeans**

http://java.sun.com/j2se/1.5.0/docs/api/javax/management/openmbean/pac kage-summary.html

#### **ObjectName**

http://java.sun.com/j2se/1.5.0/docs/api/javax/management/ObjectName.ht ml

#### MBeanServerConnection

http://java.sun.com/j2se/1.5.0/docs/api/javax/management/MBeanServerConnection.html

#### MBeanServer

http://java.sun.com/j2se/1.5.0/docs/api/javax/management/MBeanServer.ht ml

## **Managed Bus Components**

Overview	<ul> <li>This section describes the attributes and methods that you can use to manage JMX MBeans representing Artix bus components. For example, you can use any JMX client to perform the following tasks:</li> <li>View bus attributes.</li> <li>Enable monitoring of bus services.</li> <li>Dynamically change logging levels for known subsystems.</li> <li>If you wish to write your own JMX client, this section describes methods that you can use to access Artix logging levels and subsystems, and provides a JMX code example.</li> </ul>
Bus MBean registration	When an Artix bus is initialized, a corresponding JMX MBean is created and registered for that bus with an MBeanServer.
	JAX-RPC
	For example, in an Artix Java application, this occurs after the following call:
	<pre>String[] args =; Bus serverBus = Bus.init(args);</pre>
	C++
	For example, in an Artix $C++$ application, this occurs after the following call:
	<pre>Bus_var server_bus = Bus.init(argc, argv);</pre>
	When a bus is shutdown, a corresponding MBean is unregistered from the MBeanServer.
Bus naming convention	An Artix bus ObjectName uses the following convention:
	<pre>com.iona.instrumentation:type=Bus,name=busIdentifier</pre>

#### **Bus attributes**

The following bus component attributes can be managed by any JMX client:

Name	Description	Туре	Read/Write
scope	Bus scope used to initialize a bus.	String	No
identifier	Bus identifier, typically the same as its scope.	String	No
arguments	Bus arguments, including the executable name.	String[]	No
servicesMonitoring	Used to enable/disable services performance monitoring.	Boolean	Yes
services	A list of object names representing services on this bus.	ObjectName[]	No

 Table 1:
 Managed Bus Attributes

servicesMonitoring is a global attribute which applies to all services and can be used to change a performance monitoring status.

**Note:** By default, service performance monitoring is enabled when JMX management is enabled in a standalone server, and disabled in an it container process.

When using a JMX console to manage a <code>it\_container</code> server, you can enable performance monitoring by setting the <code>serviceMonitoring</code> attribute to <code>true</code>.

services is a list of object names that can be used by JMX clients to build a tree of components. Given this list, you can find all other registered service MBeans that belong to this bus.

For examples of bus attributes displayed in a JMX console, see "Managing Artix Services with JMX Consoles" on page 41.

#### **Bus methods**

If you wish to write your own JMX client, you can use the following bus methods to access logging levels and subsystems:

Name	Description	Parameters	Return Type
getLoggingLevel	Returns a logging level for a subsystem.	subsystem (String)	String
setLoggingLevel	Sets a logging level for a subsystem.	subsystem(String), level(String)	Boolean
setLoggingLevelPropagate	Sets a logging level for a subsystem with propagation.	subsystem (String), level (String), propagate (Boolean)	Boolean

 Table 2:
 Managed Bus Methods

All the attributes and methods described in this section can be determined by introspecting  ${\tt MBeanInfo}$  for the  ${\tt Bus}$  component (see <code>http://java.sun.com/j2se/1.5.0/docs/api/javax/management/MBeanInfo.html</code>).

#### Example JMX client

The following code extract from an example JMX client application shows how to access bus attributes and logging levels:

```
MBeanServerConnection mbsc = ...;
String busScope = ...;
ObjectName busName = new ObjectName("com.iona.instrumentation:type=Bus,name=" + busScope);
if (mbsc.isRegistered(busName)) {
    throw new MBeanException("Bus mbean is not registered");
}
// MBeanInfo can be used to check for all known attributes and methods
MBeanInfo info = mbsc.getMBeanInfo(busName);
// bus scope
String scope = (String)mbsc.getAttribute(busName, "scope");
// bus identifier
String identifier = (String)mbsc.getAttribute(busName, "identifier");
// bus arguments
String[] busArgs = (String[])mbsc.getAttribute(busName, "arguments");
```

```
// check servicesMonitoring attribute, then disable and reenable it
Boolean status = (Boolean)mbsc.getAttribute(busName, "servicesMonitoring");
if (!status.equals(Boolean.TRUE)) {
     throw new MBeanException ("Service monitoring should be enabled by default");
}
mbsc.setAttribute(busName, new Attribute("servicesMonitoring", Boolean.FALSE));
status = (Boolean)mbsc.getAttribute(busName, "servicesMonitoring");
if (!status.equals(Boolean.FALSE)) {
    throw new MBeanException ("Service monitoring should be disabled now");
mbsc.setAttribute(busName, new Attribute("servicesMonitoring", Boolean.TRUE));
status = (Boolean)mbsc.getAttribute(busName, "servicesMonitoring");
if (!status.equals(Boolean.TRUE)) {
    throw new MBeanException ("Service monitoring should be reenabled now");
// list of service MBeans
ObjectName[] serviceNames = (ObjectName[])mbsc.getAttribute(busName, "services");
// logging
String level = (String)mbsc.invoke(
                               busName,
                                "getLoggingLevel",
                               new Object[] {"IT BUS"},
                               new String[] {"subsystem"});
if (!level.equals("LOG ERROR")) {
    throw new MBeanException ("Wrong IT BUS logging level");
level = (String)mbsc.invoke(
                          busName,
                          "getLoggingLevel",
                          new Object[] {"IT BUS.INITIAL REFERENCE"},
                          new String[] {"subsystem"});
if (!level.equals("LOG ERROR")) {
    throw new MBeanException ("Wrong IT BUS.INITIAL REFERENCE logging level");
level = (String)mbsc.invoke(
                          busName,
                          "getLoggingLevel",
                          new Object[] {"IT BUS.CORE"},
                          new String[] {"subsystem"});
if (!level.equals("LOG INFO LOW")) {
    throw new MBeanException ("Wrong IT BUS.CORE logging level");
```

```
// check servicesMonitoring attribute, then disable and reenable it
Boolean status = (Boolean)mbsc.getAttribute(busName, "servicesMonitoring");
if (!status.equals(Boolean.TRUE)) {
     throw new MBeanException ("Service monitoring should be enabled by default");
}
mbsc.setAttribute(busName, new Attribute("servicesMonitoring", Boolean.FALSE));
status = (Boolean)mbsc.getAttribute(busName, "servicesMonitoring");
if (!status.equals(Boolean.FALSE)) {
    throw new MBeanException ("Service monitoring should be disabled now");
}
mbsc.setAttribute(busName, new Attribute("servicesMonitoring", Boolean.TRUE));
status = (Boolean)mbsc.getAttribute(busName, "servicesMonitoring");
if (!status.equals(Boolean.TRUE)) {
    throw new MBeanException ("Service monitoring should be reenabled now");
// list of service MBeans
ObjectName[] serviceNames = (ObjectName[])mbsc.getAttribute(busName, "services");
// logging
String level = (String)mbsc.invoke(
                               busName,
                               "getLoggingLevel",
                               new Object[] {"IT BUS"},
                               new String[] {"subsystem"});
if (!level.equals("LOG ERROR")) {
    throw new MBeanException ("Wrong IT BUS logging level");
}
level = (String)mbsc.invoke(
                          busName,
                          "getLoggingLevel",
                          new Object[] {"IT BUS.INITIAL REFERENCE"},
                          new String[] {"subsystem"});
if (!level.equals("LOG ERROR")) {
    throw new MBeanException ("Wrong IT BUS.INITIAL REFERENCE logging level");
level = (String)mbsc.invoke(
                          busName,
                          "getLoggingLevel",
                          new Object[] {"IT BUS.CORE"},
                          new String[] {"subsystem"});
if (!level.equals("LOG INFO LOW")) {
    throw new MBeanException ("Wrong IT BUS.CORE logging level");
```

```
Boolean result = (Boolean)mbsc.invoke(
                          busName,
                          "setLoggingLevel",
                          new Object[] {"IT BUS", "LOG WARN"},
                          new String[] {"subsystem", "level"});
level = (String)mbsc.invoke(
                          busName,
                          "getLoggingLevel",
                          new Object[] {"IT BUS"},
                          new String[] {"subsystem"});
if (!level.equals("LOG WARN")) {
    throw new MBeanException ("IT BUS logging level has not been set properly");
level = (String)mbsc.invoke(
                         busName,
                         "getLoggingLevel",
                         new Object[] {"IT BUS.INITIAL REFERENCE"},
                         new String[] {"subsystem"});
if (!level.equals("LOG WARN")) {
    throw new MBeanException ("IT BUS.INITIAL REFERENCE logging level has not been set
   properly");
3
level = (String)mbsc.invoke(
                         busName,
                         "getLoggingLevel",
                         new Object[] {"IT BUS.CORE"},
                         new String[] {"subsystem"});
if (!level.equals("LOG INFO LOW")) {
    throw new MBeanException ("IT BUS.CORE logging level should not be changed");
// propagate
result = (Boolean)mbsc.invoke(
                             busName,
                             "setLoggingLevelPropagate",
                             new Object[] {"IT BUS", "LOG SILENT", Boolean.TRUE},
                             new String[] {"subsystem", "level", "propagate"});
level = (String)mbsc.invoke(
                         busName,
                         "getLoggingLevel",
                         new Object[] {"IT BUS"},
                         new String[] {"subsystem"});
```

```
if (!level.equals("LOG SILENT")) {
    throw new MBeanException ("IT BUS logging level has not been set properly");
}
level = (String)mbsc.invoke(
                         busName,
                         "getLoggingLevel",
                         new Object[] {"IT BUS.INITIAL REFERENCE"},
                         new String[] {"subsystem"});
if (!level.equals("LOG SILENT")) {
    throw new Exception ("IT BUS.INITIAL REFERENCE logging level has not been set
   properly");
}
level = (String)mbsc.invoke(
                         busName,
                         "getLoggingLevel",
                         new Object[] {"IT BUS.CORE"},
                         new String[] {"subsystem"});
if (!level.equals("LOG SILENT")) {
    throw new MBeanException ("IT BUS.CORE logging level shouldve been set to LOG SILENT");
```

**Further information** 

For information on Artix logging levels and subsystems, see Configuring and Deploying Artix Solutions.

### **Managed Service Components**

#### Overview

This section describes the attributes and methods that you can use to manage JMX MBeans representing Artix service components. For example, you can use any JMX client to perform the following tasks:

- View managed services.
- Dynamically change a service status.
- Monitor service performance data.
- Manage service ports.

The Artix locator and session manager services, have also been instrumented. These provide an additional set of attributes on top of those common to all services. For information on WS-RM persistence instrumentation, see Chapter 4.

If you wish to write your own JMX client, this section describes methods that you can use and provides a JMX code example.

Service MBean registration

When an Artix servant is registered for a service, a JMX Service MBean is created and registered with an MBeanServer.

#### JAX-RPC

For example, in an Artix Java application, this occurs after the following call:

```
Bus bus = Bus.init(args);
QName bankServiceName = new
   QName("http://www.iona.com/bus/tests", "BankService");
Servant servant = new SingleInstanceServant(new BankImpl(),
   serviceWsdlURL, bus);
```

bus.registerServant(servant, bankServiceName, "BankPort");

#### C++

For example, in an Artix C+ + application, this happens after the following call:

```
Bus_var server_bus = Bus.init(argc, argv);
BankServiceImpl servant;
bus->register_servant(
    servant,
    wsdl_location,
    QName("http://www.iona.com/bus/tests", "BankService")
);
```

When a service is removed, a corresponding MBean is unregistered from the MBeanServer.

Service naming convention An Artix service ObjectName uses the following convention:

com.iona.instrumentation:type=Bus.Service,name="{namespace}local
 name",Bus=busIdentifier

In this format, a  $_{name}$  has an expanded service QName as its value. This value includes double quotes to permit for characters that otherwise would not be allowed.

Service attributes

The following service component attributes can be managed by any JMX client:

Table 3:         Managed Service Attribute	Table 3:	Managed S	Service	Attributes
--	----------	-----------	---------	------------

Name	Description	Туре	Read/Write
name	Service QName in expanded form.	String	No
state	Service state.	String	No
serviceCounters	Service performance data.	CompositeData	No
ports	A list of ObjectNames representing ports for this service.	ObjectName[]	No

name is an expanded QName, such as

{http://www.iona.com/bus/tests}BankService.

 ${\tt state}$  represents a current service state that can be manipulated by stop and start methods.

ports is a list of ObjectNames that can be used by JMX clients to build a tree of components. Given this list, you can find all other registered Port MBeans which happen to belong to this Service.

#### serviceCounters attributes

The following service performance attributes can be retrieved from the serviceCounters attribute:

Name	Description	Туре
averageResponseTime	Average response time in milliseconds.	Float
requestsOneway Total number of oneway requests to this service.		Long
requestsSinceLastCheck	Number of requests happened since last check.	Long
requestsTotal	Total number of requests (including oneway) to this service.	Long
timeSinceLastCheck	Number of seconds elapsed since last check.	Long
totalErrors	Total number of request-processing errors.	Long

Table 4:	serviceCounters	Attributes
	001110000000000000000000000000000000000	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,

For examples of service attributes displayed in a JMX console, see "Managing Artix Services with JMX Consoles" on page 41

#### Service methods

If you wish to write your own JMX client, you can use the following service methods to manage a specific service:

 Table 5:
 Managed Service Attributes

Name	Description	Parameters	Return Type
name	Start (activate) a service.	None	Void
state	Stop (deactivate) a service.	None	Void

All the attributes and methods described in this section can be accessed by introspecting MBeanInfo for the Service component.

#### Example JMX client

The following code extract from an example JMX client application shows how to access service attributes and methods:

```
MBeanServerConnection mbsc = ...;
String busScope = ...;
ObjectName serviceName = new ObjectName("com.iona.instrumentation:type=Bus.Service" +
                     ",name=\"{http://www.iona.com/hello world soap http}SOAPService\""
   +",Bus=" + busScope);
if (!mbsc.isRegistered(serviceName)) {
    throw new MBeanException ("Service MBean should be registered");
// MBeanInfo can be used to check for all known attributes and methods
MBeanInfo info = mbsc.getMBeanInfo(serviceName);
// service name
String name = (String)mbsc.getAttribute(serviceName, "name");
// check service state attribute then reset it by invoking stop and start methods
String state = (String)mbsc.getAttribute(serviceName, "state");
if (!state.equals("ACTIVATED")) {
    throw new MBeanException ("Service should be activated");
mbsc.invoke(serviceName, "stop", null, null);
```

```
state = (String)mbsc.getAttribute(serviceName, "state");
if (!state.equals("DEACTIVATED")) {
   throw new MBeanException ("Service should be deactivated now");
}
mbsc.invoke(serviceName, "start", null, null);
state = (String)mbsc.getAttribute(serviceName, "state");
if (!state.equals("ACTIVATED")) {
   throw new MBeanException ("Service should be activated again");
// check service counters
CompositeData counters = (CompositeData)mbsc.getAttribute(serviceName, "serviceCounters");
Long requestsTotal = (Long)counters.get("requestsTotal");
Long requestsOneway = (Long)counters.get("requestsOneway");
Long totalErrors = (Long)counters.get("totalErrors");
Float averageResponseTime = (Float)counters.get("averageResponseTime");
Long requestsSinceLastCheck = (Long)counters.get("requestsSinceLastCheck");
Long timeSinceLastCheck = (Long)counters.get("timeSinceLastCheck");
// ports
ObjectName[] portNames = (ObjectName[])mbsc.getAttribute(serviceName, "ports");
```

**Further information** 

#### MBeanInfo

http://java.sun.com/j2se/1.5.0/docs/api/javax/management/MBeanInfo.html

#### **CompositeData**

http://java.sun.com/j2se/1.5.0/docs/api/javax/management/openmbean/CompositeData.html

### **Artix Locator Service**

#### Overview

The Artix locator can also be exposed as a JMX MBean. A locator managed component is a service managed component that can be managed like any other bus service with the same set of attributes and methods. The Artix locator also exposes it own specific set of attributes.

Locator attributes

An Artix locator MBean exposes the following locator-specific attributes:

Name	Description	Туре
registeredEndpoints	Number of registered endpoints.	Integer
registeredServices	Number of registered services, less or equal to number of endpoints.	Integer
serviceLookups	Number of service lookup requests.	Integer
serviceLookupErrors	Number of service lookup failures.	Integer
registeredNodeErrors	Number of node (peer ping) failures.	Integer

#### Example JMX client

The following code extract from an example JMX client application shows how to access locator attributes and methods:

### **Artix Session Manager Service**

#### Overview

The Artix session manager can also be exposed as a JMX MBean. A session manager component is a service managed component that can be managed like any other bus service with the same set of attributes and methods. The Artix session manager also exposes it own specifc set of attributes.

#### Session manager attributes

An Artix session manager MBean exposes the following session manager-specific attributes:

Name	Description	Туре
registeredEndpoints	Number of registered endpoints.	Integer
registeredServices	Number of registered services, less or equal to number of endpoints.	Integer
serviceGroups	Number of service groups.	Integer
serviceSessions	Number of service sessions	Integer

 Table 7:
 Session Manager MBean Attributes

#### Example JMX client

The following code extract from an example JMX client application shows how to access session manager attributes and methods:

```
MBeanServerConnection mbsc = ...;
String busScope = ...;
ObjectName serviceName = new ObjectName("com.iona.instrumentation:type=Bus.Service" +
    ",name=\"{http://ws.iona.com/sessionManager}SessionManagerService\"" +",Bus=" +
    busScope);
// use common attributes and methods, see an example above
// SessionManager specific attributes
Integer regServices = (Integer)mbsc.getAttribute(serviceName, "registeredServices");
Integer serviceGroups = (Integer)mbsc.getAttribute(serviceName, "serviceGroups");
Integer serviceSessions = (Integer)mbsc.getAttribute(serviceName, "serviceGroups");
```

## **Managed Port Components**

Overview	<ul> <li>This section describes the attributes that you can use to manage JMX</li> <li>MBeans representing Artix port components. For example, you can use any</li> <li>JMX client to perform the following tasks:</li> <li>Monitor managed ports.</li> <li>View message and request interceptors.</li> </ul>	
	If you wish to write your own JMX client, this section also shows an example of accessing these attributes in JMX code.	
Port MBean registration	Port managed components are typically created as part of a service servant registration. When service is activated, all supported ports will also be registered as MBeans.	
	When a service is removed, a corresponding Service MBean, as well as all its child Port MBeans are unregistered from the MBeanServer.	
Naming convention	An Artix port ObjectName uses the following convention:	
	<pre>com.iona.instrumentation:type=Bus.Service.Port,name=portName,Bus     .Service="{namespace}localname",Bus=busIdentifier</pre>	

Port attributes The following bus component attributes can be managed by any JMX client:

Name	Description	Туре	Read/Write
name	Port name.	String	No
address	Transport specific address representing an endpoint.	String	No
interceptors	List of interceptors for this port.	String[]	No

Name	Description	Туре	Read/Write
transport	An optional attribute representing a transport for this port.	ObjectName[]	No

#### Table 8: Supported Service Attributes

#### interceptors

The interceptors attribute is a list of interceptors for a given port. Internally, interceptors is an instance of TabularData that can be considered an array/table of CompositeData. However, due to a current limitation of CompositeData, (no insertion order is maintained, which makes it impossible to show interceptors in the correct order), the interceptors are currently returned as a list of strings, where each String has the following format:

[name]: name [type]: type [level]: level [description]: optional
 description

In this format, type can be CPP or Java; level can be Message or Request.

It is most likely that this limitation will be fixed in a future JDK release, probably JDK 1.7 because the enhancement request has been accepted by Sun. In the meantime, interceptors details can be retrieved by parsing a returned string array.

For examples of port attributes displayed in a JMX console, see "Managing Artix Services with JMX Consoles" on page 41

#### Example JMX client

The following code extract from an example JMX client application shows how to access port attributes and methods:

```
MBeanServerConnection mbsc = ...;
String busScope = ...;
ObjectName portName = new ObjectName ("com.iona.instrumentation:type=Bus.Service.Port" +
                     ",name=SoapPort" +
   ",Bus.Service=\"{http://www.iona.com/hello world soap http}SOAPService\"" +",Bus=" +
   busScope);
if (!mbsc.isRegistered(portName)) {
    throw new MBeanException ("Port MBean should be registered");
}
// MBeanInfo can be used to check for all known attributes and methods
MBeanInfo info = mbsc.getMBeanInfo(portName);
// port name
String name = (String)mbsc.getAttribute(portName, "name");
// port address
String address = (String)mbsc.getAttribute(portName, "address");
// check interceptors
String[] interceptors = (String[])mbsc.getAttribute(portName, "interceptors");
if (interceptors.length != 6) {
    throw new MBeanException ("Number of port interceptors is wrong");
handleInterceptor(interceptors[0],
                  "MessageSnoop",
                  "Message",
                  "CPP");
handleInterceptor(interceptors[1],
                  "MessagingPort",
                  "Request",
                  "CPP");
handleInterceptor(interceptors[2],
                  "http://schemas.xmlsoap.org/wsdl/soap/binding",
                  "Request",
                  "CPP");
```

```
handleInterceptor(interceptors[3],
                                   "TestInterceptor",
                               "Request",
                               "Java");
handleInterceptor(interceptors[4],
                               "bus_response_monitor_interceptor",
                               "Request",
                             "CPP");
handleInterceptor(interceptors[5],
                          "ServantInterceptor",
                         "Request",
                              "CPPF");
```

For example, the handleInterceptor() function may be defined as follows:

CHAPTER 1 | Monitoring and Managing an Artix Runtime

### CHAPTER 2

# Configuring JMX in an Artix Runtime

This chapter explains how to configure an Artix runtime to be managed with Java Management Extensions (JMX).

In this chapter

This chapter discusses the following topic:

Artix JMX Configuration

page 38

## **Artix JMX Configuration**

Overview	This section explains the Artix configuration variable settings that you must configure to enable JMX monitoring of the Artix runtime, and access for remote JMX clients.
Enabling the management plugin	To expose the Artix runtime using JMX MBeans, you must enable a bus_management plug-in as follows:
	<pre>jmx_local {     plugins:bus_management:enabled="true"; };</pre>
	This setting enables local access to JMX runtime MBeans. The <pre>bus_management plug-in wraps runtime components into Open Dynamic</pre> MBeans and registers them with a local MBeanServer.
Configuring remote JMX clients	To enable remote JMX clients to access runtime MBeans, use the following configuration settings:
	<pre>jmx_remote {     plugins:bus_management:enabled="true";     plugins:bus_management:connector:enabled="true"; };</pre>
	These settings allow for both local and remote access.
	<b>Specifying a remote access URL</b> Remote access is performed through JMX Remote, using an RMI Connector on a default port of 1099. Using this configuration, you can use the following JNDI-based JMXServiceURL to connect remotely:

service:jmx:rmi:///jndi/rmi://host:1099/artix

o r ·				
Configuring	а	remote	access	nort
Soundaring	ч	10111010	400000	POIL

To specify a different port for remote access, use the following configuration variable:

plugins:bus management:connector:port="2000";

You can then use the following JMXServiceURL:

service:jmx:rmi:///jndi/rmi://host:2000/artix

Configuring a stub-based JMXServiceURL

You can also configure the connector to use a stub-based JMXServiceURL as follows:

```
jmx_remote_stub
{
    plugins:bus_management:enabled="true";
    plugins:bus_management:connector:enabled="true";
    plugins:bus_management:connector:registry:required="false";
};
```

See the javax.management.remote.rmi package for more details on remote JMX.

Publishing the JMXServiceURL to<br/>a local fileYou can also request that the connector publishes its JMXServiceURL to a<br/>local file:

plugins:bus management:connector:url:publish="true";

The following entry can be used to override the default file name:

plugins:bus management:connector:url:file="../../service.url";

#### **Further information**

For further information, see the following:

#### **RMI Connector**

http://java.sun.com/j2se/1.5.0/docs/api/javax/management/remote/rmi/RMI Connector.html

#### JMXServiceURL

http://java.sun.com/j2se/1.5.0/docs/api/javax/management/remote/JMXServ iceURL.html

http://java.sun.com/j2se/1.5.0/docs/api/javax/management/remote/rmi/package-summary.html

### CHAPTER 3

## Managing Artix Services with JMX Consoles

You can use third-party management consoles that support JMX Remote to monitor and manage Artix servers (for example, JConsole and MC4J). You can view the status of a bus instance, stop or start a service, change bus logging levels, or view interceptor chains.

This chapter discusses the following topics:

Managing Artix Services with JConsole	page 42
Managing Artix Services with the JMX HTTP adaptor	page 46
Managing Artix Services with MC4J	page 49

In this chapter

## Managing Artix Services with JConsole

Overview	You can also use JConsole, which is provided with JDK 1.5, to monitor and manage Artix applications. JConsole displays Artix runtime managed components in a hierarchical tree, as shown in Figure 2.		
Using JConsole	To use JConsole, perform the following steps:		
	1. Start up JConsole using the following command:		
	JDK_HOME/bin/jconsole		
	2. Select the <b>Advanced</b> tab.		
	<ol> <li>Enter or paste a JMXServiceURL (either the default URL, or one copied from a published connector.url file).</li> </ol>		
Managing services	Figure 2 shows the attributes displayed for a managed service component (for example, the serviceCounters performance metrics displayed in the right pane). For detailed information on these attributes, see "Service attributes" on page 25.		

🔹 J2SE 5.0 Monitoring & Management Console: service:jmx:rmi:	///jndi/rmi://s	beryoz:5008/	/artix		
Connection					
Summary Memory Threads Classes MBeans VM					
MBeans					
Tree		0	N-41541	V Infa 1	
	Attributes	Operations	Notifications	Info	
	Nam			Valu	
← 🗂 JMImplementation	name				ntime}SOAPService
e 📑 com.iona.instrumentation	ports	jav	ax.management	.ObjectN	ame[1]
P-☐ Bus ↓			< Tabular Na	avigation	>
			<< Composi	te Naviga	
P			Name	<b>-</b>	Value 0.023500001
🕈 📑 demos.jmx_runtime.server	serviceCounte		erageResponseT questsOneway	IIme	0
			questsSinceLast	Chook	0
🦳 🧐 SoapPort			questsTotal	CHECK	8
			reSinceLastChec	12	610
🔶 🔚 demos.jmx_runtime.server			alErrors	/R	0
	state	AC	TIVATED		
			Refresh		

Figure 2: Managed Service in JConsole

#### Managing ports

Figure 3 shows the attributes displayed for a managed port component (for example, the interceptors list displayed in the right pane). For detailed information on these attributes, see "Port attributes" on page 32.

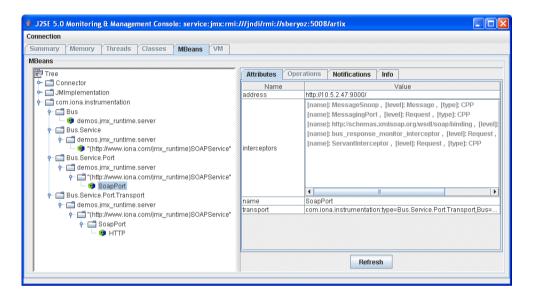


Figure 3: Managed Port in JConsole

#### Managing containers

Figure 4 shows an example of a locator service deployed into an Artix container. For more information, see "Locator attributes" on page 29.

👙 J2SE 5.0 Monitoring & Management Console: service:jmx:rmi:///jndi	/rmi://sberyo	:5008/artix	
Connection			
Summary Memory Threads Classes MBeans VM			
MBeans			
Tree	Attributes	Operations	Notifications Info
🔶 🗂 Connector	N	ame	Value
🕶 🗂 JMImplementation	name	ame	{http://ws.iona.com/2005/11/locator}LocatorService
🗣 🗂 com.iona.instrumentation	ports		javax.management.ObjectName[1]
🕈 🗂 Bus	registeredEn		1
🗏 🧐 demos.locator_load_balancing.locator	registeredNo		0
🕈 🥅 Bus.Service	registeredSer	vices	1
Bus Service     demos locator_load_balancing locator     demos locator_load_balancing locator     demos locator_load_balancing locator     demos locator_load_balancing locator     demos locator_locatorService"     demos locator.com/container/ContainerService"     demos locator.com/peer_manager/PeerManagerService"     demos locator.com/peer_manager/PeerManagerService	serviceCounters		< Tabular Navigation Composite
	serviceLooku	- Freeze	0
	serviceLooku		1
	state	20	ACTIVATED
			Refresh

Figure 4: Managed Locator in JConsole

**Note:** When using a JMX console to manage a service running in an Artix container, set the serviceMonitoring attribute to true to enable service performance monitoring (see "Bus attributes" on page 18).

Further information

For more information on using JConsole, see the following: http://java.sun.com/developer/technicalArticles/J2SE/jconsole.html

## Managing Artix Services with the JMX HTTP adaptor

Overview	You can also manage Artix services using the default HTTP adaptor console that is provided with the JMX reference implementation. This console is browser-based, as shown in Figure 5.		
Using the JMX HTTP adaptor	To use the JMX HTTP adaptor, perform the following steps: 1. Specify following configuration settings:		
	<pre>plugins:bus_management:http_adaptor:enabled="true"; plugins:bus_management:http_adaptor:port="7659";</pre>		
	<ol> <li>Enter the following URL in your browse: http://localhost:7659</li> <li>This displays the main HTTP adaptor management view, as shown in Figure 5.</li> </ol>		

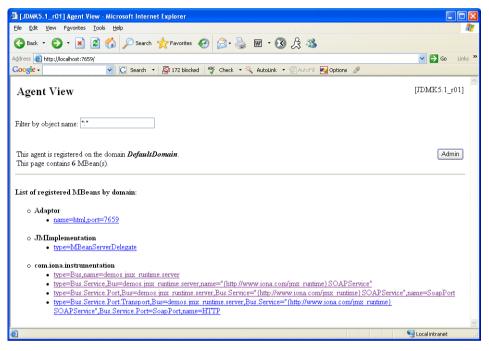


Figure 5: HTTP Adaptor Main View

Figure 6 shows the attributes displayed for a managed bus component (for example, the services that it includes). For detailed information on these attributes, see "Bus attributes" on page 18.

Figure 6: HTTP Adaptor Bus View

#### Further information

For further information on using the HTTP JMX adaptor, see the following: http://java.sun.com/developer/technicalArticles/J2SE/jmx.html

## Managing Artix Services with MC4J

Overview	You can use the open source MC4J management console to view service attributes and operations, stop or start a service, view interceptor chains, and change bus logging levels dynamically. This section uses the <code>jmx_runtime</code> Artix demo to show a detailed walk-through example of how to use MC4J to monitor and manage an Artix server.		
Running the JMX demo	<ul> <li>Before creating a new server connection in the MC4J console, perform the following steps:</li> <li>1. Change to the demo directory:</li> <li>cd InstallDir\Version\cxx_java\samples\advanced\management\jmx_runtime</li> </ul>		
	<ul> <li>2. Build the C++ or Java demo:</li> <li>C++ nmake</li> <li>Java ant</li> <li>2. Due the O is is a large series</li> </ul>		
	3. Run the C++ or Java server: C++ run_cxx_server.bat		

Java run\_java\_server.bat

#### Creating a new server connection

To create a new server connection in the MC4J console, perform the following steps:

1. Select **MC4J Connections,** and right click, as shown in Figure 7.

MC-U 1.2 beta 9			- E E 🛛
Management Tools Window Help			000000000
197 14.924.640			
Currention			
MC4J Connections - Properties 4 ×			
<li>dia Propertes&gt;</li>			
ME-L4 Donnections Maritaris the lot of all IM Monogenent connections			
Output - MC4J Errors			¥ X
Inappropriate call from Arm-PromoThread at erg.mcd; consola.ban.MReadWorks.loadActLives(TheadWork Inappropriate call from Arm-PromoThread at erg.mcd; consola.ban.MReadWorkJini.actLinifeThread[TR Thappropriate call from Arm-PromoThread at erg.mcd; consola.ban.MReadWorkJini.actLinifeThread[TR Thappropriate call from Arm-PromoThread at erg.mcd; consola.ban.MReadWorkJini.actLinifeThread[TR C	anNode.java:542)		× ×
🛃 start 🔰 🔭 Moldbeik 🛸 Vindows 🎯 Installiknywh 🖬 Command Pr 🖬 Clycnald, 13	Jac MC4112 Set	/ 102 IQ 😤	🚯 10:17 AM

Figure 7: Connecting to a Server

2. Click **Connection server...** to launch the **My wizard** dialog, as shown in Figure 8.

🐃 My Wizard	duauduauduauduauduauduaudu	
Steps	Panel Name wizard (1	of 4)
1. Connect to Server	JSR160	<b>V</b>
Select Server Installation     Customize classpath     Choose connector	Start by selecting your s	erver connection type above
	Name	JMX demo
•	JNDI Name	
💼 🚸	Initial Context Factory	i.jndi.rmi.registry.RegistryContextFactory 💌
	Server URL	x:rmi://dli670/jndi/rmi://DLI670:5008/artix
	Principle	
MC4J	Credentials	
		Advanced
Steps Help		
	< Back Net	xt > Einish Cancel Help

Figure 8: Server Connection Details

- 3. In the **My Wizard** dialog, select JSR160 as your server connection type.
- 4. Enter JMX demo as your connection Name.
- 5. Enter the contents of the following file as the **Server URL**:

...samples/advanced/management/jmx\_runtime/etc/connector.url

🗠 My Wizard	
Steps 1. Connect to Server 2. Select Server Instalation 3. Customize classpath 4. Choose connector	Panel Name witzard (3 of 4) Cuttom desspath and server libraries Add Remove
	Additional mc4jbb directory libraries Ciljonal(9_13_best1)articl(4.1)mc4jmc4jbb/arim3tion-1.1.3.jsr Ciljonal(9_13_best1)articl(4.1)mc4jmc4jbb/area72,jsr Ciljonal(9_13_best1)articl(4.1)mc4jmc4jbb/area72,dsr Ciljonal(9_13_best1)articl(4.1)mc4jmc4jbb/area0_d-a4,jsr Ciljonal(4.1)mc4jmc4jbb/area0_d-a4,jsr Ciljonal(4.1)mc4jmc4jbb/area0_d-a4,jsr Ciljonal(4.1)mc4jmc4jbb/area0_d-a4,jsr Ciljonal(4.1)mc4jmc4jbb/area0_d-a4,jsr Ciljonal(4.1)mc4jmc4jbb/area0_d-a4,jsr Ciljonal(4.1)mc4jmc4jbb/area0_d-a4,jsr Ciljonal(4.1)mc4jmc4jbb/area0_d-a4,jsr Ciljonal(4.1)mc4jmc4jbb/area0_d-a4,jsr Ciljonal(4.1)mc4jmc4jbb/area0_d-a4,jsr Ciljonal(4.1)mc4jmc4jbb/area0_d-a4,jsr Ciljonal(4.1)mc4jmc4jbb/area0_d-a4,jsr Ciljonal(4.1)mc4jmc4jbb/area0_d-a4,jsr Ciljonal(4.1)mc4jmc4jb
Steps (100)	<betk next=""> Finish Cancel Hop</betk>

6. Click **Next** to go to next screen, as shown in Figure 9.

Figure 9: Creation of Server Connection

7. Click **Finish** to finish the creation of a new server connection.

8. In the left panel of the MC4J console, a new server connection named JMX demo is created, as shown in Figure 10:

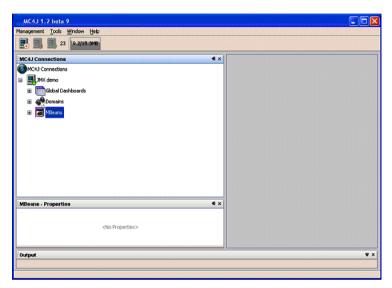


Figure 10: New Server Connection

Monitoring and managing a service

To monitor and manage an example service in the Mc4J console, perform the following steps:

- 1. Expand the **MBeans** tree node in the left panel of MC4J.
- 2. Double click on the following tree node, as shown in Figure 11:

Name='{http://www.iona.com/jmx\_runtime}SOAPService',type=Bus.
Service

This displays the attributes and operations of the SOAPService in the service properties dialog.

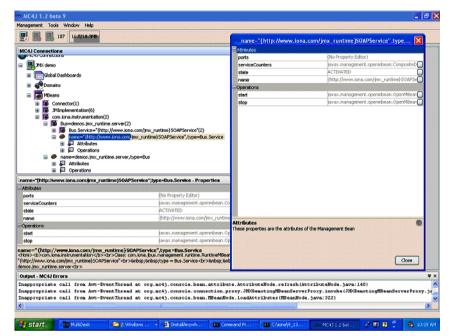


Figure 11: Viewing Service Properties

3. Click the ... button at the right of the serviceCounters attribute in the service properties dialog. This displays the details for the serviceCounters attribute, as shown in Figure 12.

	0.0
averageResponseTime	0
requestsOneway	
requestsSinceLastCheck	0
requestsTotal	150
timeSinceLastCheck totalErrors	0
rvice counters	

Figure 12: Viewing Service Counters Properties

4. Click the ... button at right of the **stop** operation on the service properties dialog. This displays a dialog for the **stop** operation, as shown in Figure 13.

🔲 nam	e=" 🔀
Exe	cute
	Close

Figure 13: Stopping a Service

5. Click **Execute...** to stop the service. In the SOAPSERVICE properties dialog, the state attribute of the service becomes DEACTIVATED, as shown in Figure 14.

Attributes	
ports	(No Property Editor)
serviceCounters	javax.management.openmbean.CompositeD
state	DEACTIVATED
name	{http://www.iona.co
Operations	DEACTIVATED
start	javax.management.openmbean.OpenMBean
stop	javax.management.openmbean.OpenMBean
top ause service	

Figure 14: Deactivated Service

6. Click the ... button at the right of **start** operation on SOAP service properties. This displays a dialog for the **start** operation, which is the same as the one shown in Figure 13.

 Click Execute... to restart the service. In the service properties dialog, the state of the SOAPService becomes ACTIVATED, as shown in Figure 15.

Attributes	na.com/jmx_runtime}SOAPService",type 🔀
ports	(No Property Editor)
serviceCounters	javax.management.openmbean.CompositeD
state	ACTIVATED
name	{http://www.iona.com/jmx_runtime}50AP5e
Operations	
start	avax.management.openmbean.OpenMBean
stop	javax.management.openmbean.OpenMBean

Figure 15: Activated a Service

#### Monitoring a service port

To monitor an example service port in the Mc4J console, perform the following steps:

1. Click the following node in the left panel of the MC4J console:

name=SoapPort,tyoe=Bus.Service.Port

This displays the attributes for SoapPort, as shown in Figure 16.

MC4J 1.2 beta 9				
anagement Tools Window Help		name-So	apPort,type-Bus.Service.Port - Propertie	5
<b>I I I</b> 149 8.8/22.5748		Attricutes		
		address	http://10.65.4.139:9000/	
MC4J Connections		interceptors	[name]: MessageSnoop , [level]: Message , [type	e]: CPP, [name]: Messaging
MC43 Connections		name	SoapPort	
I JMX demo		transport	com.iona.instrumentation:Bus-demos.jmc_run	kime.server,Bus.Servi
		8		
Global Dashboards				
Domains				
MBeans		8		
E 🗃 Connector(1)				
IMImplementation(6)		8		
😑 👼 com.iona.instrumentation(2)				
🖃 📕 Bus=demos.jmx_runtime.				
	www.iona.com/jmc_runtime}SOAP5ervice*(2)			
	e=Bus.Service.Port.Transport,Bus.Service.Port=SoapP type=Bus.Service.Port	ovt 📲		
	<pre>kype=bus.bervice.rort .kona.com/jmx_runtime}SOAPService",type=Bus.Service</pre>			
inter-demos.jmx_runtim				
Attributes	erserver/vitheore			
me-SoapPort.type=Bus.Service.Port -				
tributes	ri opei des	1000 B		
address	http://10.65.4.139:9000/			
nterceptors	[name]: MessageSnoop , []er	(e)		
name	SoapPort			
ransport	com.iona.instrumentation	But		
anapar.	Connection of Connection			
		國		
		transport		
me=SoapPort,type=Bus.Service.Port		port transport (	javax.management.ObjectName)	
ml> <b>com.iona.instrumentation</b> <br< td=""><th>&gt;Class: com.iona.jbus.management.runtime.RuntimeM</th><td></td><td></td><td></td></br<>	>Class: com.iona.jbus.management.runtime.RuntimeM			
pPort %nbsp;%nbsp;type = Bus.Servic ttp://www.iona.com/jmx_runtime}SOAPSer	e.Port br>  Bus = demos.jmx_runtime.set	ver		
· · · · · · · · · · · · · · · · · · ·		-		
ntput - MC4J Errors				Close
	tThread at org.mc4j.console.bean.HBear			
	tThread at org.mc4j.console.bean.HBear			
appropriate call from Aut-Even				
appropriate call from Aut-Even	tThread at org.mc4j.console.bean.HBear	Node.access#10	0(HBeanNode.java:88)	
appropriate call from Aut-Even	tThread at org.nc4j.console.bean.fBeam	Node.access#10	0 (MBeanNode.java:88)	
appropriate call from Aut-Even	tThread at org.ac4j.console.bean.HBean	Node.access#10	0(HBeanNode.java:88)	

Figure 16: Viewing Port Properties

2. Click the ... button at the right of the interceptors attribute in Figure 16. This displays the interceptors properties for the selected bus, as shown in Figure 17.

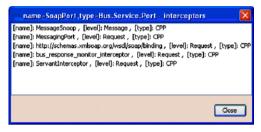


Figure 17: Viewing Interceptor Properties

**Further information** 

For full details on using the MC4J management console, see the MC4J documentation:

http://mc4j.org/confluence/display/MC4J/User+Guide

## Managing Logging Levels with MC4J

Overview	This section uses the jmx_runtime Artix demo to show a detailed walk-through example of how to use the MC4J console to manage Artix bus logging levels dynamically at runtime. The following logging configuration is defined in the demos.jmx_runtime configuration scope:	
Defined demo logging configuration		
	Logging Subsystem	Logging Level
	IT_BUS	LOG_ERROR
	IT_BUS.CORE	LOG_INFO_LOW
		ging level for IT_BUS, and all of its child subsystems, exception is IT_BUS.CORE, which has a logging level
Viewing logging levels for a subsystem	To view logging levels for a specified Artix logging subsystem in MC4J, perform the following steps:	
	1. Expand the followi	ng tree node in the left panel of MC4J:
	<ul> <li>name=demos.jmx_runtime.server,type=Bus</li> <li>2. Expand the Operations node.</li> <li>3. Double click getLoggingLevel. This displays the My Wizard screen shown in Figure 18.</li> </ul>	

You can use this wizard to view the logging level of a specified subsystem.

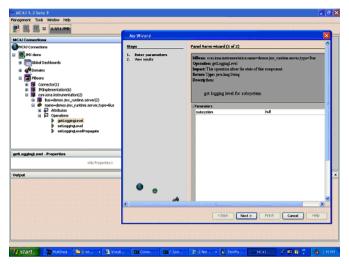


Figure 18: Logging Viewing Wizard

- My Wizard × Panel Name wizard (1 of 2) Steps 1. Enter parameters View results MBean: com iona instrumentation name=demos jmx\_runtime server, type=Bus 2. Operation: getLoggingLevel Impact: This operation alters the state of this component Return Type: jevalang String Description: get logging level for subsystem Parameters IT\_BUS subsystem < <u>B</u>ack Next > Finish Cancel Help
- 4. Enter the IT BUS subsystem, as shown in Figure 19.

Figure 19: Entering a Logging Subsystem

- 5. Click Next. This displays the logging level of IT\_BUS as LOG\_ERROR, as shown in Figure 20.
- 6. Click Finish.

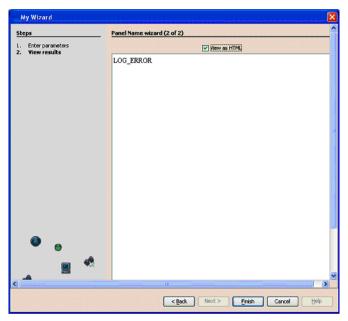


Figure 20: Displayed Logging Level

- 7. Similarly, use the **My Wizard** screen to enter a logging subsystem of IT\_BUS.INITIAL\_REFERENCE.
- Click Next. The logging level for the IT\_BUS.INITIAL\_REFERENCE subsystem is also displayed as LOG\_ERROR. The IT\_BUS.INITIAL\_REFERENCE subsystem inherits the same logging level from its IT\_BUS parent.
- 9. Finally, use the **My Wizard** screen to enter a logging subsystem of IT\_BUS.CORE.
- Click Next. The logging level for IT\_BUS.CORE is displayed as LOG\_INFO\_LOW. The logging level for IT\_BUS.CORE has been configured differently from its IT\_BUS parent (see "Defined demo logging configuration" on page 60).

## Setting the logging level for a subsystem

To set the logging level for a specified logging subsystem, perform the following steps:

- 1. Double click the setLoggingLevel node in the left panel of the MC4J console. This displays the **My Wizard** screen, as show in Figure 21.
- 2. Enter IT\_BUS for the subsystem, and LOG\_WARN for the logging level, as as show in Figure 21.

🗠 My Wizard		X	
Steps	Panel Name wizard (1 of 2)		
Enter parameters     Wiew results	MBean: com iona instrumentation name=demos jmx_runtime server,type=Bus Operation: setLoggingLevel Impact: This operation alters the state of this component. Return Type: jeva.lang.Boolean Description: set logging level for subsystem		
	- Parameters		
	subsystem	IT_BUS	
	level	LOG_WARN	
• •			
<	сЩ <sub>отототот</sub>		
	< Back	Next > Einish Cancel Help	

Figure 21: Setting a Logging Level

3. Click **Next**. This displays true, as shown in Figure 22, which means that the logging level is set successfully.

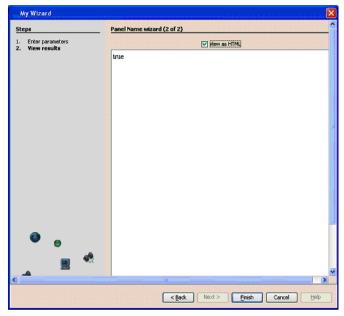


Figure 22: Logging Level Set Successfully

- 4. View the logging level of the IT\_BUS subsystem to verify your setting (as described in "Viewing logging levels for a subsystem" on page 60). The logging level for IT BUS is now LOG WARN.
- 5. View the logging level for the IT\_BUS.INITIAL\_REFERENCE subsystem. The logging level for IT\_BUS.INITIAL\_REFERENCE is also LOG\_WARN.
- 6. View the logging level for IT\_BUS.CORE. The logging level of IT\_BUS.CORE is still LOG\_INFO\_LOW. It does not inherit the LOG\_WARN level from its parent because its logging level has been configured separately (see "Defined demo logging configuration" on page 60).

### Setting the logging level for a subsystem with propagation

To set a logging level to override a child subsystem with a separately configured logging level, perform the following steps:

1. Double click the setLoggingLevelPropagate tree node in left panel of MC4J. This displays the **My Wizard** screen, as shown in Figure 22.

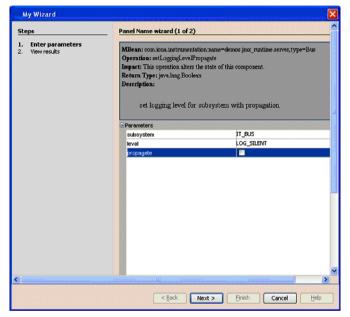


Figure 23: Propagating a Logging Level

- 2. Enter IT BUS as the subsystem, and LOG SILENT as the logging level.
- 3. Click **Next**. The returned value is true, which means that the logging level is set successfully.
- View the logging level for IT\_BUS (as described in "Viewing logging levels for a subsystem" on page 60). The logging level for IT\_BUS is LOG\_SILENT.

	5.	View the logging level for IT_BUS.INITIAL_REFERENCE. The logging level for IT_BUS.INITIAL_REFERENCE is also LOG_SILENT.
	6.	View the logging level for IT_BUS.CORE. The logging level for IT_BUS.CORE is also LOG_SILENT. Specifying propagation overrides log levels for all child logging subsystems.
Further information		detailed information on Artix logging, see Configuring and Deploying x Solutions.
	For	more information on using MC4J, see:
	http	o://mc4j.org/confluence/display/MC4J/Home

CHAPTER 3 | Managing Artix Services with JMX Consoles

#### CHAPTER 4

## Managing WS-RM Persistence with JMX

You can manage Web Services Reliable Messaging persistence in Artix using any JMX console.

In this chapter

This chapter discusses the following topics:

WS-RM Persistence Managementpage 70Viewing Messages in the WS-RM Persistence Databasepage 72

## **WS-RM Persistence Management**

Overview	database both persistence er client's RM so	any JMX console to view messages in the WS n locally and remotely. You also can monitor nabled endpoint, the WS-RM acksTo endpoin purce endpoint. This section explains the WS nat can be managed in a JMX console.	the WS-RM nt URI, and the
Managed WS-RM persistence components	console: Managed (RMEndpo Managed	WS-RM persistence components can be ma d WS-RM persistence endpoints pintPersistentStore) d WS-RM persistence sequences encePersistentStore)	naged in a JMX
Managed WS-RM persistence endpoints	WS-RM persis destination er WS-RM persis unregistered f	stence endpoint managed components are u stence enabled endpoints. When a WS-RM p adpoint is created, it is registered as an MBe stence destination endpoint is closed, the M rom the MBeanServer. aming convention is as follows:	persistence an. When an
<pre>com.iona.instrumentation:type=Bus.Service.Port.EndpointP name=WSRM_ENDPOINT_PERSISTENCE, Bus.Service.Port=portName, Bus.Service="{namespace}localname", Bus=busIdentifier</pre>		pintPersistent,	
	You can view the following attributes for a WS-RM persistence endpoint in JMX console:		ence endpoint in a
	Name	Description	Туре
	service name	WS-RM persistence enabled service name	String
	port name	WS-RM persistence enabled port	String

### Managed WS-RM persistence sequences

WS-RM persistence sequence managed components are used to represent WS-RM sequences. A destination sequence with a unique ID is created for each client. When a WS-RM persistence destination sequence is created, it is registered as an MBean. When a WS-RM persistence destination sequence is recovered from database, it is also registered as an MBean. When a WS-RM persistence destination sequence is terminated, it is unregistered from the MBeanServer.

The MBean naming convention is as follows:

com.iona.instrumentation:type=Bus.Service.Port.EndpointPersistent.SequencePersistent,
name=sequenceName,

```
Bus.Service.Port.EndpointPersistent=WSRM ENDPOINT PERSISTENCE,
```

```
Bus.Service.Port=portName,
```

Bus.Service="{namespace}localName",

```
Bus=busIdentifier
```

In this syntax, *sequenceName* includes the string sequence\_id and the sequence ID.

#### WS-RM persistence sequence attributes

You can view the following attributes for a WS-RM persistence sequence in a JMX console:

Name	Description	Туре
acksto uri	WS-RM acknowledgement URI	String
messages	Messages in the WS-RM persistence database	String[]
sequence id	Sequence unique ID representing a client	String

The messages attribute is a list of messages in the WS-RM persistence database. The messages are returned as a list of strings, where each string has the following format:

[message id]: messageId [message]: soapMessage

## Viewing Messages in the WS-RM Persistence Database

Overview	Before you start viewing in the WS-RM persistence database, you must set your Artix configuration to enable JMX management for WS-RM persistence. This section uses the Artix WS-RM example demo (demos\advanced\wsrm) to explain how to view and monitor messages in the WS-RM persistence database.
Enable JMX management fo WS-RM persistence	<ul> <li>To enable JMX management for WS-RM persistence in your Artix configuration file, perform the following steps:</li> <li>1. Open thedemos\advanced\wsrm\etc\wsrm.cfg file.</li> <li>2. Edit the demos.wsrm_persistence_enabled.server scope as follows:</li> </ul>
	<pre>server {     plugins:artix:db:home = "./server_db";     plugins:bus_management:enabled="true";     plugins:bus_management:connector:enabled="true";     plugins:bus_management:connector:url:file="//etc/connector.url";     # optional port, default is 1099     plugins:bus_management:connector:port="5008"; };</pre>

**Note:** Enabling JMX management for WS-RM persistence is similar to enabling JMX management for other Artix components.

Start the server	<ol> <li>To start the server, go to the demos\advanced\wsrm\bin directory, and run the following command:</li> </ol>
	<pre>run_cxx_server_persistence.bat This starts the server using following example command:</pre>
	<pre>start server.exe -ORBname demos.wsrm_persistence_enabled.server</pre>
	When the server runs, a file named connector is created in thedemos\advanced\wsrm\etc\ directory.
Start a JMX console	You can start any JMX console. For example, to start JConsole, execute the following command:
	%jdk1.5_home%\bin\jconsole.exe
	This displays the <b>JConsole: Connect to Agent</b> dialog, as shown in Figure 24.

👙 JConsole: Connect to Agent	$\mathbf{X}$
Local Remote Advanced	
JMX URL: service:jmx:rmi:///jndi/rmi://DLI670:5008/artix	
User Name:	
Password:	
Connect Cancel	

Figure 24: Connecting to a JMX Agent

Copy the contents of the connector file into the JMX URL field, and click **Connect**. This displays the **J2SE 5.0 Monitoring and Management Console**, as shown in Figure 25.

View WS-RM persistence enabled endpoints You can view a WS-RM persistence enabled endpoint in the **MBeans** tab of the JMX console, as shown in Figure 25:

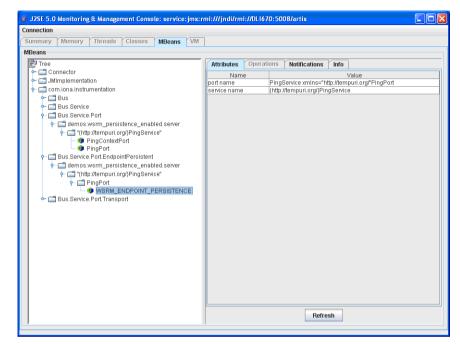


Figure 25: WS-RM Persistence Enabled Endpoint

In this example, PingPort is a WS-RM persistence enabled port. You can view the port and service name in the **Attributes** tab on the right of the console.

### View messages in the WS-RM persistence database

To view messages in the WS-RM persistence database, perform the following steps:

1. Edit the client code in

...\demos\advanced\wsrm\cxx\client\PingClientSample.cxx as
follows:

```
int
  run_persistence_client(
    int argc,
    char* argv[]
)
....
for (int i=0; i < 10; i++)
    {
      cout << "Invoking PingOneway " << i << endl;
      PingType paraml;
      paraml.setText("PingOneway message from client");
      client1.PingOneway(param1);
      cout << i << " PingOneway invoked" << endl;
    }
....</pre>
```

This adds a loop to the client that invokes the server 10 times in order to easily view messages in WS-RM persistence database.

2. Start the client. For example, go to the demos\advanced\wsrm\bin directory, and run the following command:

run\_cxx\_client\_persistence.bat

 You can view the attributes for the WS-RM sequence in the JMX console, as shown in Figure 26. The WS-RM sequence name consists of the sequence\_guid string and a sequence ID.

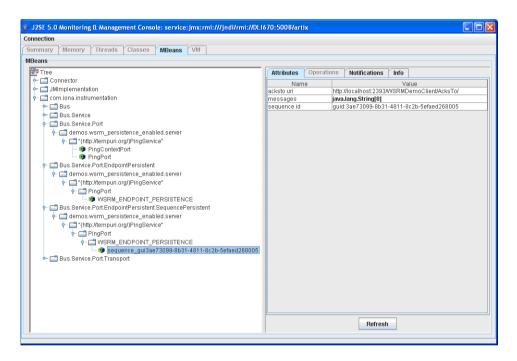


Figure 26: WS-RM Sequence Attributes

 You can view all the messages in WS-RM persistence database by clicking in the Attributes tab on the right of the console, as shown in Figure 27. Each message consists of a message ID and a SOAP message.

onnection		
Summary Memory Threads Classes MBeans VM		
1Beans		
Tree	Attributes	Operations Notifications Info
🗠 🗂 Connector	Name	Value
🕶 🚍 JMImplementation	acksto uri	http://localhost:2393/WSRMDemoClient/AcksTo/
🕈 🚍 com.iona.instrumentation		[message id]: 3 [message]: xml version='1.0' encod</td
🕶 🗂 Bus		[message id]: 4 [message]: xml version='1.0' encod</td
🗠 🛄 Bus.Service		[message id]: 5 [message]: xml version='1.0' encod</td
🕈 🗂 Bus.Service.Port		[message id]: 6 [message]: xml version='1.0' encod</td
demos.wsrm_persistence_enabled.server	messages	[message id]: 7 [message]: xml version='1.0' encod</td
🛉 🚍 "(http://tempuri.org/)PingService"	messages	[message id]: 8 [message]: xml version='1.0' encod</td
- 🧐 PingContextPort - 📵 PingPort		[message id]: 9 [message]: xml version='1.0' encod</td
		[message id]: 10 [message]: xml version='1.0' enco</td
Genos.wsrm_persistence_enabled.server		
* [] "{http://tempuri.org/PingService"		
Cal PingPort	sequence id	guid:3ae73099-8b31-4811-8c2b-5efaed268005
Ell Bus Service Port EndpointPersistent SequencePersistent		
- demos.wsrm_persistence_enabled.server		
←  ☐ "(http://tempuri.org/)PingService"		
- 🗂 PingPort		
- C WSRM_ENDPOINT_PERSISTENCE		
Image: sequence_gui3ae73099-8b31-4811-8c2b-5efaed268005		
🖕 🚍 Bus.Service.Port.Transport		
		Refresh

Figure 27: Messages in the WS-RM Persistence database

You can click the **Refresh** button to view the current messages in WS-RM persistence database.

#### CHAPTER 4 | Managing WS-RM Persistence with JMX

## Index

#### Α

acksto uri 71 address 32 arguments 18 Attributes tab 74 averageResponseTime 26

#### В

bus attributes 18 ObjectName 17 bus management 38

#### С

CompositeData 33 connector.url 42 custom JMX MBeans 14

#### G

getLoggingLevel 19

#### Н

HTTP adaptor 46

#### I

identifier 18 interceptors 32, 44

#### J

J2SE 5.0 Monitoring and Management Console 73 Java Management Extensions 11, 37 JConsole 42, 73 Connect to Agent 73 JMX 11, 37 JMX HTTP adaptor 46 JMX Remote 15 JMXServiceURL 38

#### L

locator

managed attributes 29 logging levels 19 subsystems 19

#### Μ

Managed Beans 12 management consoles 41 MBeans 12 MBeanServer 12, 70 MBeanServerConnection 14 MBeans tab 74 messages 71

#### Ρ

persistence endpoint 70 plugins:artix:db home 72 plugins:bus management:connector enabled 72 plugins:bus management:connector:enabled 38 plugins:bus management:connector:port 72 plugins:bus management:connector:registry:require d 39 plugins:bus management:connector:ur file 72 plugins:bus management:connector:url:file 39 plugins:bus management:connector:url:publish 39 plugins:bus management:enabled 38, 72 plugins:bus management:http adaptor:enabled 46 plugins:bus management:http adaptor:port 46 port name 32 ObjectName 32 port name 70 ports 25

#### R

registeredEndpoints 29, 31 registeredNodeErrors 29 registeredServices 29, 31 remote access port 39 remote JMX clients 38 requestsOneway 26 requestsSinceLastCheck 26 requestsTotal 26 RMEndpointPersistentStore 70 RMI Connector 38 RMSequencePersistentStore 70 runtime MBeans 14

#### S

scope 18 sequence\_guid 76 sequence ID 76 sequence id 71 service attributes 25 managed components 24 methods 27 name 25 ObjectName 25 serviceCounters 25 serviceGroups 31 serviceLookupErrors 29 serviceLookups 29 service name 70 services 18 serviceSessions 31 servicesMonitoring 18 session manager managed attributes 31 setLoggingLevel 19 setLoggingLevelPropagate 19 state 25

#### T

TabularData 33 timeSinceLastCheck 26 totalErrors 26 transport 33

#### W

WS-RM persistence 70